A Direct Georeferencing Imaging Technique to Identify Earth Surface Temperatures Using Oblique Angle Airborne Measurements

by
Ryan Andrew Everett Byerlay

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Applied Science
in
Engineering

Guelph, Ontario, Canada

ABSTRACT

A DIRECT GEOREFERENCING IMAGING TECHNIQUE TO IDENTIFY EARTH SURFACE TEMPERATURES USING OBLIQUE ANGLE AIRBORNE MEASUREMENTS

Ryan A. E. Byerlay
University of Guelph, 2019

Advisors:
Dr. Amir A. Aliabadi
Dr. Mohammad Biglarbegian

This thesis describes a novel, open-source image processing method that directly georeferences oblique angle thermal images of the Earth's surface and calculates Earth surface temperatures at a high spatiotemporal resolution. Images were collected from a thermal camera mounted on the Tethered And Navigated Air Blimp 2 (TANAB2). Median surface temperatures are represented spatially in six four-hour time interval plots to display diurnal surface temperature variation. The technique is applied to two data sets collected during two separate field campaigns, one from a northern Canadian mining facility and one from the University of Guelph, Guelph, Ontario, Canada. A comparison between surface temperatures for images recorded from the mining facility and a MODerate resolution Imaging Spectroradiometer (MODIS) image is completed with a resulting median absolute error of 0.64 K, bias of 0.5 K, and Root Mean Square Error (RMSE) of 5.45 K. Based on the findings, the developed direct georeferencing oblique angle thermal image processing method is capable of calculating surface temperatures with an accuracy of approximately 5 K at a spatiotemporal resolution that is significantly higher than conventional satellite-based sensors. Further applications of this direct georeferencing workflow are numerous and can be evalu-

ated with other cameras such as Red Green Blue (RGB), multispectral, and hyperspectral imaging systems.

Dedication

I would like to dedicate this work to my family, friends, teachers, and most of all my parents for their encouragement and support throughout my educational pursuit.

Acknowledgements

This work could not have been completed without the exceptional support and guidance provided by my advisor Dr. Amir A. Aliabadi and my co-advisor Dr. Mohammad Biglarbegian. Their thoughts and provision of new ideas were invaluable throughout the duration of my graduate studies. I would also like to thank my colleagues Dr. Manoj K. Kizhakkeniyil, Amir Nazem, Md. Rafsan Nahian, Dr. Mojtaba Ahmadi-Baloutaki, Seyedahmad Kia, and Mohsen Moradi who assisted with the collection of data used in this work and provided helpful support throughout the duration of my research.

Completing this work would not be possible without the technical support of Rowan Williams Davies and Irwin Inc. (RWDI) and financial support from the University of Guelph, the Ed McBean philanthropic fund, the Discovery Grant program (401231) from the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Government of Ontario through the Ontario Centres of Excellence (OCE) under the Alberta-Ontario Innovation Program (AOIP) (053450), and from Emission Reduction Alberta (ERA) (053498).

The Tethered And Navigated Air Blimp 2 (TANAB2) was partially developed by the assistance of Denis Clement, Jason Dorssers, Katharine McNair, James Stock, Darian Vyriotes, Amanda Pinto, and Phillip Labarge. The TANAB2 tether reel system was developed by Andrew F. Byerlay. TANAB2 gondola electrical configuration advice provided by C. Harrison Brodie is appreciated. Steve Nyman, Chris Duiker, Peter Purvis, Manuela Racki, Jeffrey Defoe, Joanne Ryks, Ryan Smith, James Bracken, and Samantha French at the University

of Guelph assisted with field campaign logistics. Special credit is directed toward Amanda Sawlor, Datev Dodkelian, Esra Mohamed, Di Cheng, Randy Regan, Margarent Love, and Angela Vuk at the University of Guelph for administrative support. The computational platforms were set up with the assistance of Jeff Madge, Joel Best, and Matthew Kent at the University of Guelph. Technical discussions with John D. Wilson and Thomas Flesch at the University of Alberta are highly appreciated. Field support from Alison M. Seguin (RWDI), Andrew Bellavie (RWDI), and James Ravenhill at Southern Alberta Institute of Technology (SAIT) is appreciated.

Most of all, I would like to express my sincere gratitude to my parents. Their endless emotional and financial support towards my education contributed immeasurably to this work.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ABI	Advanced Baseline Imager
ALOS	Advanced Land Observing Satellite
AVHRR	Advanced Very High Resolution Radiometer
BBE	BroadBand Emissivity
DEM	Digital Elevation Model
DSM	Digital Surface Model
EDT	Eastern Daylight Time
ETM+	Enhanced Thematic Mapper Plus
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GOES	Geostationary Operational Environmental Satellite
GPS	Global Positioning System
HFOV	Horizontal Field of View
HPR	Horizontal Pixel Range
IMU	Inertial Measurement Unit
LDT	Local Daylight Time
LiDAR	Light Detection And Ranging
LMFIT	Non-Linear Least-Squares Minimization and Curve-Fitting
LWIR	Long Wave Infrared Radiation
MODIS	MODerate resolution Imaging Spectroradiometer
NASA	National Aeronautics and Space Administration
NOAA	National Oceanic and Atmospheric Administration

PCA	Principal Component Analysis
POES	Polar-orbiting Operational Environmental Satellite
PPK	Post-Processing Kinematic
PPP	Precise Point Positioning
PRISM	Panchromatic Remote-sensing for Stereo Mapping
RTK	Real-Time Kinematic
RMSE	Root Mean Square Error
ST	Surface or Skin Temperature
sUAS	small Unmanned Aerial System
TANAB2	Tethered And Navigated Air Blimp 2
TIR	Thermal InfRared
TIRS	Thermal InfRared Sensor
UAV	Unmanned Aerial Vehicle
USGS	United States Geological Survey
VFOV	Vertical Field of View
VPR	Vertical Pixel Range

List of Mathematical Symbols

Latin Symbols

a	Constant
B	Constant
BBE	Emissivity
b	Constant
c	Constant
d_{edge}	Distance
d_{horiz}	Distance
$d_{\text{horiz}_{\text{top}}}$	Distance
d_{launch}	Distance
F	Constant
$HFOV$	Angle
HPR	Pixel
i	Pixel
k	Distance
Lat_1	Latitude
Lat_2	Latitude
lat_1	Latitude
lat_2	Latitude
Lon_1	Longitude
Lon_2	Longitude
lon_1	Longitude

lon_2	Longitude
n	Integer
O	Constant
$P0$	Pixel
$P256$	Pixel
$P512$	Pixel
P_1	Pressure
P_2	Pressure
P_x	Pixel
R_1	Constant
R_2	Constant
R	Constant
z_1	Altitude
z_2	Altitude
z_{AGL}	Altitude
Slope	Slope
T_{Obj}	Temperature
$\overline{T_v}$	Temperature
U_{Atm}	Signal Value
U_{Obj}	Signal Value
U_{Refl}	Signal Value
U_{Tot}	Signal Value
$VFOV$	Angle
VPR	Pixel
X	Pixel
Yaw	Angle
Yb	Pixel
Yc	Pixel
y_{pixel}	Pixel

Yt	Pixel
Ytx	Pixel

Greek Symbols

α	Angle
β	Angle
η	Angle
γ	Angle
κ	Angle
θ	Angle
Δ	Implies difference
ϵ	Emissivity
ϵ_{29}	Emissivity
ϵ_{31}	Emissivity
ϵ_{32}	Emissivity
τ	Transmissivity

Chapter 1

Introduction

Earth surface temperature, otherwise known as Skin Temperature (ST), is an important geophysical variable that has been measured with remote sensing technologies since the 1970s [60, 72]. Accurate quantification of ST is important for many Earth system models, including meteorological, climate, and planetary boundary layer models [58, 77, 89]. Micro-, meso-, and macro-scale climate models all consider ST as a key variable, as noted by Gémes et al. [28]. Land surface temperature is a key variable when quantifying the impacts of urban heat islands. Specifically, the diurnal impact of ST with respect to the surrounding environment are of importance to many researchers [44, 59]. Furthermore, macro- and meso-scale models, including those that model the change of climate, consider ST over both land and waterbodies as a boundary condition [25, 34]. The impact of ST on large freshwater lakes has also been studied as surface water temperature influences thermal stratification in lakes [49, 64].

1.1 Literature Review

1.1.1 Measurement

ST can be quantified as a function of Long Wave Infrared Radiation (LWIR) emitted from the Earth's surface [100]. The emitted LWIR is an important variable when considering the Earth energy budget from incoming solar radiation [100]. Before the advent of satellites and other remote sensing platforms, multiple point sources recording either surface temperature or air temperature were used in conjunction with weighting algorithms and other Geographic Information Systems (GIS) techniques to spatially represent ST [75]. These historical meth-

ods can introduce significant inaccuracies during interpolation of the data, as a result, remote sensing tools have since been utilised to reduce these data analysis errors [75].

Satellite-Based Sensors

Conventionally, ST has been quantified from remote sensing satellites with onboard ST sensors including the MODerate resolution Imaging Spectroradiometer (MODIS), the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), the Advanced Baseline Imager (ABI), the Enhanced Thematic Mapper Plus (ETM+), the Thermal InfRared Sensor (TIRS), and the Advanced Very High Resolution Radiometer (AVHRR) amongst other data sources [36, 89]. These sensors record data within the Thermal InfRared (TIR) or LWIR spectra between 8 and 15 µm [94].

MODIS is located on both the Terra and Aqua satellites which are operated by the National Aeronautics and Space Administration (NASA) [51]. Both satellites have polar, sun-synchronous orbits[1,2] and both MODIS sensors record two distinct images daily at different times, approximately three hours apart [19]. The horizontal resolution of MODIS images is approximately 1 km × 1 km [56]. MODIS records data from 36 spectral bands in total which include wavelengths between 0.41 and 14.4 µm, of specific importance for ST are the bands 20 to 36 which cover the spectral range between 3.66 and 14.4 µm [56, 104]. In addition to MODIS, ASTER is also located onboard the Terra satellite [66]. ASTER images the Earth's surface with a high spatial resolution of 90 m × 90 m and a revisit time of 16 days for each imaged location [30, 32]. ASTER contains three subsystems, including a system which records data within the TIR spectrum between 8.13 and 11.7 µm.[3]

Geostationary Operational Environmental Satellite (GOES) satellites are operated through a collaboration between NASA and the National Oceanic and Atmospheric Administration (NOAA).[4] The GOES-R series of satellites are the most recent geostationary satellites to be developed and launched by this partnership, specifically GOES-16 and GOES-17 which were operational as of December 18, 2017 and February 12, 2019 respectively[5,6] [4]. Onboard both GOES-16 and GOES-17, the ABI records images for 16 spectral bands which range between 0.45 and 13.6 µm[7] [83]. The ABI on GOES-R series satellites records images over

[1]https://aqua.nasa.gov/content/about-aqua
[2]https://terra.nasa.gov/about
[3]https://asterweb.jpl.nasa.gov/characteristics.asp
[4]https://www.nasa.gov/content/goes-overview/index.html
[5]https://www.goes-r.gov/users/transitionToOperations16.html
[6]https://www.goes-r.gov/users/transitionToOperations17.html
[7]https://www.goes-r.gov/spacesegment/ABI-tech-summary.html

the continental United States every 5 minutes for all 16 spectral bands and at a spatial resolution of 2 km × 2 km [17, 82].

Landsat satellites are developed and operated by NASA in collaboration with the United States Geological Survey (USGS).[8] Currently, Landsat 7 ETM+ records images with eight spectral bands, one of which records within the 10.4 and 12.5 µm spectrum. Landsat 8 TIRS records images with 11 spectral bands; one band records data within the 10.6 to 11.19 µm range and another records data within the 11.5 to 12.5 µm range [23]. Landsat 7 ETM+ records TIR data at a spatial resolution of 60 m × 60 m, while Landsat 8 TIRS records TIR data at a spatial resolution of 120 m × 120 m [37]. Both Landsat 7 and 8 have a revisit time period of 16 days [15].

AVHRR is a sensor that is operated by NOAA and is located onboard the Polar orbiting Operational Environmental Satellites (POES) series [41]. The third generation AVHRR is located onboard NOAA-15, -16, -17, -18, and -19, of which NOAA-15, -18, and -19 are still operational [41].[9] AVHRR has 6 channels in total and two channels which image within the LWIR spectrum, one channel records data within the 10.3 to 11.3 µm range while the other channel records data within the 11.5 to 12.5 µm range.[10] The NOAA-15, -18, and -19 satellites are sun-synchronous polar orbiting satellites and AVHRR records images at a spatial resolution of 1.1 km × 1.1 km. These satellites have a short revisit period that varies between four and six AVHRR revisits per day as AVHRR is located on multiple satellites [43, 47, 52].

Although there are many remote sensing instruments which map ST, satellite-based ST sources have limitations, for example, land surface temperature measurements are impacted by the atmosphere and surface emissivity [54]. If atmospheric characteristics are known, an algorithmic model can be used to extract land surface temperature from satellite imagery [62]. Even with corrections to quantify ST, both high spatial and temporal resolution land surface temperature data is not available from satellite sources [11]. High spatial resolution data is associated with low temporal resolution data and high temporal resolution data is associated with low spatial resolution data [11, 106]. Furthermore, satellites are known to be impacted by cloud cover, atmospheric dust, and sensor failure [57, 68]. Miniaturisation of thermal imaging technologies and the development of reliable Unmanned Aerial Vehicles (UAVs) and Unmanned Aerial Systems (UASs), such as drones, kites, and blimps, provide

[8] https://landsat.gsfc.nasa.gov/

[9] https://www.ospo.noaa.gov/Operations/POES/status.html

[10] https://noaasis.noaa.gov/NOAASIS/ml/avhrr.html

another airborne platform to derive ST from images [7, 48]. Furthermore, coupling small Unmanned Aerial System (sUAS) platforms with oblique thermal imaging technology and concurrent image processing methods can result in increased ST coverage.

Unmanned Aerial Systems

Recently it has become increasingly common for sUAS platforms to include thermal cameras [18]. Uncooled thermal cameras are most often used on sUASs as they are physically lighter, inexpensive, and require less power to operate as compared to cooled thermal cameras [74, 80, 84]. There are many types of sUAS devices used to deploy camera systems, including but not limited to fixed wing and multi-rotor drones, kites, blimps, and balloons [21].

Tethered balloons are another aerial platform that have several advantages as compared to conventional sUASs. Tethered balloons can be deployed for hours without changing batteries, be launched in remote and complex environments where drones are unable to fly (e.g. airports), are inexpensive relative to other sUAS platforms, and their altitude can be precisely controlled, amongst other advantages [65, 96]. A few studies have been completed involving thermal imaging and tethered balloons. Vierling et al. [96] deployed a helium-filled, tethered aerostat equipped with a thermal infrared sensor, capable of lifting a payload of 78 kg and flying in a maximum wind speed greater than 11 m s^{-1}. Rahaghi et al. [74] launched a tethered-helium-filled balloon equipped with a FLIR Tau 2 thermal camera over Lake Geneva, Switzerland, "under weak wind conditions."

Airborne sUAS vectors, including drones and balloons, have been noted to be deployed in maximum wind speeds up to 10 m s^{-1}, after which sUAS performance is significantly degraded [78]. von Burern et al. [97] and Hardin et al. [31] reported that many manufacturers claim that UAVs are capable of flying in wind speeds up to 8.3 m s^{-1}. However, Hardin et al. [31] stated that wind speeds greater than 7 m s^{-1} can impact flight time and performance. Ren et al. [79] noted that the DJI Phantom 4 quadcopter, a popular drone produced for the consumer market, has a maximum wind resistance speed of 10 m s^{-1}. Puliti et al. [73] collected earth surface images from a UAV over multiple flights in wind speeds up to 7 m s^{-1}, where each flight lasted approximately 24 minutes. Boon et al. [9] used two types of UAVs (fixed wing and multi-rotor) for an environmental mapping study. Both UAVs were capable of flying in a maximum wind speed of 11.1 m s^{-1}. Rankin and Wolff [76] used a tethered balloon during a field campaign in which the manufacturer recommended use in maximum wind speeds up to 12 m s^{-1}, however the blimp was not flown in wind speeds above 8 m s^{-1}. Hot-air-based balloons experience inflation and positioning difficultly in wind speeds greater

than 4.17 m s^{-1} and helium-filled balloons were found to be destabilised in winds greater than 1.4 m s^{-1} as described by Aber [1].

1.1.2 Direct Georeferencing

With advancing sUAS technology, including the integration of Inertial Measurement Units (IMUs) and Global Navigation Satellite System (GNSS) units, sUAS imaging systems have been able to directly georeference images without the use of ground control [18, 70, 86]. The angular and positioning data provided by these systems can either be used directly or processed with Real-Time Kinematic (RTK), Post-Processing Kinematic (PPK), Precise Point Positioning (PPP), or differential correction techniques prior to being utilised in direct georeferencing methods [6, 61, 86, 109]. Without the use of differential correction for geographical coordinates calculated from direct georeferencing, positional accuracy in the range of 2 m to 5 m is typical [92, 102]. Padró et al. [70] quantified Root Mean Square Error (RMSE) for GNSS direct georeferencing without correction and PPK methods with respect to pre-defined ground control point locations. Planimetric RMSE for the uncorrected GNSS directly georeferenced data was 1.06 m, while vertical error was 4.21 m. The RMSE for the PPK methods were at least one order of magnitude less than that of the uncorrected direct georeferencing method. However, Padró et al. [70] noted that the uncorrected GNSS approach may be appropriate, such as in cases of analysing satellite images with a pixel size greater than 2 m.

1.1.3 Thermal and Oblique Imaging

Thermal cameras use microbolometer focal plane arrays to observe incoming radiant energy [26, 69]. When an image is captured, the microbolometer array represents the observed energy as a signal value (commonly referred to as digital numbers or A/D counts) which includes the radiant energy emitted from the atmosphere, reflected by the surface, and recorded from the imaged surface of the object [107]. Microbolometer temperature is known to vary as a function of sensor, camera, and ambient temperatures [12, 26, 55]. Cooled thermal cameras are significantly more sensitive than uncooled systems and provide more accurate, absolute temperatures [80]. However, current cooled camera technology requires an airborne vector capable of lifting more than 4 kg, which is greater than the capacity of sUAS and smaller tethered-balloon-system payloads [91]. It has been noted in literature that uncooled thermal cameras can be radiometrically calibrated to reduce uncooled camera

error to ±5 K [27, 46].

Oblique imaging systems coupled with sUAS or tethered-balloon systems can significantly increase the recorded land surface area as compared to nadir imaging systems. However, radiometric thermal imaging systems can be impacted from non-nadir setups, where surface temperature error is introduced as a function of observation angle [22]. Viewing angles greater than 30° of nadir over waterbodies have been noted to introduce surface temperature error of approximately 0.5 K [22, 45, 90]. This error is introduced as the surface emissivity of water changes as the viewing angle of the thermal camera becomes more oblique and reflected radiation from the surface increasingly influences the internal sensor of the thermal camera [22]. Horton et al. [35] noted that sea surface temperature emissivity varied between 0.36 and 0.98 for viewing angles between 90° (nadir) and 5° (below horizontal), respectively. James et al. [40] recorded ground-based oblique thermal images of lava flows and quantified a ±3 % difference in radiative power from the lava flows where error increases as more distant objects had high emissivities as compared to closer ones. More distant objects were likely to be influenced the most by increasingly oblique viewing angles. In the study, they accounted for atmospheric transmission effects of the radiation [40]. Hopkinson et al. [33] recorded ground-based oblique thermal images of a glacier at varying diurnal times. It was noted that the maximum temperature difference was ±3 K, where the emissivity was assumed to be 0.98 However, the calculated glacier surface temperatures were not validated. As a result, it is possible that these temperature variations could be influenced by transmitted and reflected radiation.

1.2 Technology Gaps

High spatial and temporal resolution data of the Earth's surface capable of characterising diurnal ST patterns is difficult to obtain from conventional remote sensing sources [58]. Furthermore, the use of open-source direct georeferencing methods and surface temperature calculation for thermal images collected from airborne vectors at oblique angles are not widely reported [95]. The coupling of direct georeferencing thermal images captured from a tethered-balloon-based vector is novel, and the focus of this thesis is the development of an open-source image processing workflow to map surface temperatures with a high spatiotemporal resolution.

1.3 Objectives

In this thesis, the development of an open-source, Python-based thermal image direct georeferencing and ST calculation method is described and evaluated with respect to MODIS satellite imagery. The developed program quantifies ST at a high spatiotemporal resolution as compared to conventional satellite sources. The images were collected during two different field campaigns, both of which will be detailed further, the first within a remote northern Canadian mining facility in May 2018 and the second on campus at the University of Guelph during July and August 2018. The diurnal ST variation will be represented for both field campaigns and trends will be discussed and explained.

1.4 Thesis Structure

The structure of this thesis is as follows:

Chapter 1: An overview of current thermal remote sensing techniques is provided in addition to recent advancements made in UAS remote sensing technology, direct georeferencing, thermal imaging, and oblique imaging. Relevant literature supporting each topic is included and reviewed appropriately. This chapter also highlights the existing technology gaps in the presented research and details the objectives and motivation for this thesis.

Chapter 2: The specifics of the two field campaigns and instrumentation used to collect data are detailed. Section 2.3.1 specifies the direct georeferencing method, Section 2.3.2 specifies the ST calculation, and Section 2.3.2 specifies the calibration procedure applied to the thermal camera used in both field experiments. Section 2.4 briefly describes the workflow used to develop the Principal Component Analysis (PCA).

Chapter 3: The results from the method discussed in **Chapter 2**, are displayed. Diurnal ST variation, ST comparison to MODIS, and PCA figures are presented and discussed in detail with references to recently published, relevant literature.

Chapter: 4: Conclusions from **Chapter 3** are reiterated and potential applications of the imaging workflow are discussed.

Chapter 2

Method Development

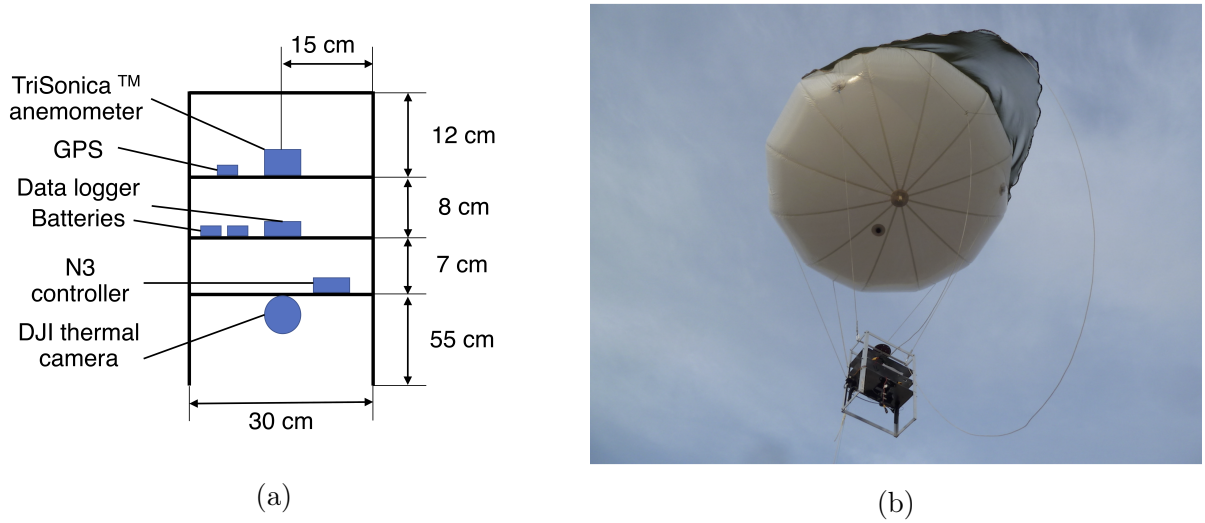
2.1 Experimental Materials

The images processed in the study were obtained from a DJI Zenmuse XT 19-mm lens uncooled thermal camera[1], which was located onboard a customised airborne platform: the Tethered And Navigated Air Blimp 2 (TANAB2). In addition to the thermal camera, a TriSonica^{TM} anemometer[2], measuring wind speed, wind direction, air pressure, and air temperature at 10Hz, a TriSonica^{TM} datalogger, and a DJI N3 flight controller[3] were included onboard the TANAB2. The camera, TriSonica^{TM} system, N3, and related equipment, including batteries, were all located on the TANAB2 payload, referred to as the gondola. The layout of the instruments on the gondola, the dimensions of the gondola, and the TANAB2 in flight are detailed in Figure 2.1.

The TriSonica^{TM} anemometer, has a temperature measurement range of $-40\,°C$ to $80\,°C$ with a resolution of $0.1\,K$ and an accuracy of $\pm 2\,K$. Furthermore, the TriSonica^{TM} anemometer has a pressure measurement range of $50\,kPa$-$115\,kPa$ with a resolution of $0.01\,kPa$ and an accuracy of $\pm 1\,kPa$.

The DJI Zenmuse XT radiometric thermal camera with a 19-mm lens is sensitive to radiation within the $7.5\,\mu m$-$13.5\,\mu m$ band and has a focal plane array resolution of 640×512 (horizontal by vertical pixels). The camera has a radiometric sensitivity of less than $0.05\,K$ and an accuracy of $\pm 5\,K$. The 19-mm lens has a horizontal field of view of 32° and a vertical field of view of 26°. The radiometric camera is also capable of recording pixel data

[1]https://www.dji.com/ca/zenmuse-xt
[2]https://www.anemoment.com/trisonica-mini-product-comparison/
[3]https://www.dji.com/ca/n3/info



(a) (b)

Figure 2.1: (a) Diagram of the gondola on the TANAB2; (b) the TANAB2 deployed during a field environmental monitoring campaign in May 2018.

at 14-bit resolution.

2.2 Field Campaigns

2.2.1 Mining Site Campaign

During May 2018, the TANAB2 was deployed in a northern Canadian remote open-pit mining facility. The specifics of the project, including the name of the mine, location, and client, cannot be disclosed due to the signing of a non-disclosure agreement with the industrial partner of the project. As a result, figures included in this thesis are void of geographical identifying features, such as latitude and longitude data. Instead, ST in relation to the site perimeter, tailings pond, and open-pit mine is quantified.

The TANAB2 and DJI thermal camera have been deployed and surface images have been recorded in a remote mining site in northern Canada during dawn, day, dusk, and night. The TANAB2 was deployed a total of twelve times at three different locations as denoted by Figure 2.2 [65]. Within the boundaries of the remote mining site, the TANAB2 and DJI camera setup were used in conjunction with a Lightbridge2 controller and either an Android- or iOS-powered smartphone. With the TANAB2 deployed, using the Lightbridge2, the thermal camera was tilted parallel to the horizon and was positioned at either the left

or right most Yaw maximum of the camera gimbal. Methodically, the camera was panned horizontally and an image was captured approximately every 5°. When the maximum Yaw limitation of the gimbal was reached, the camera was tilted approximately 5° towards the Earth's surface and the imaging procedure repeated again until the camera was perpendicular to the ground. This imaging procedure occurred approximately every hour during each TANAB2 profile in an effort to record the diurnal variation of surface temperature.

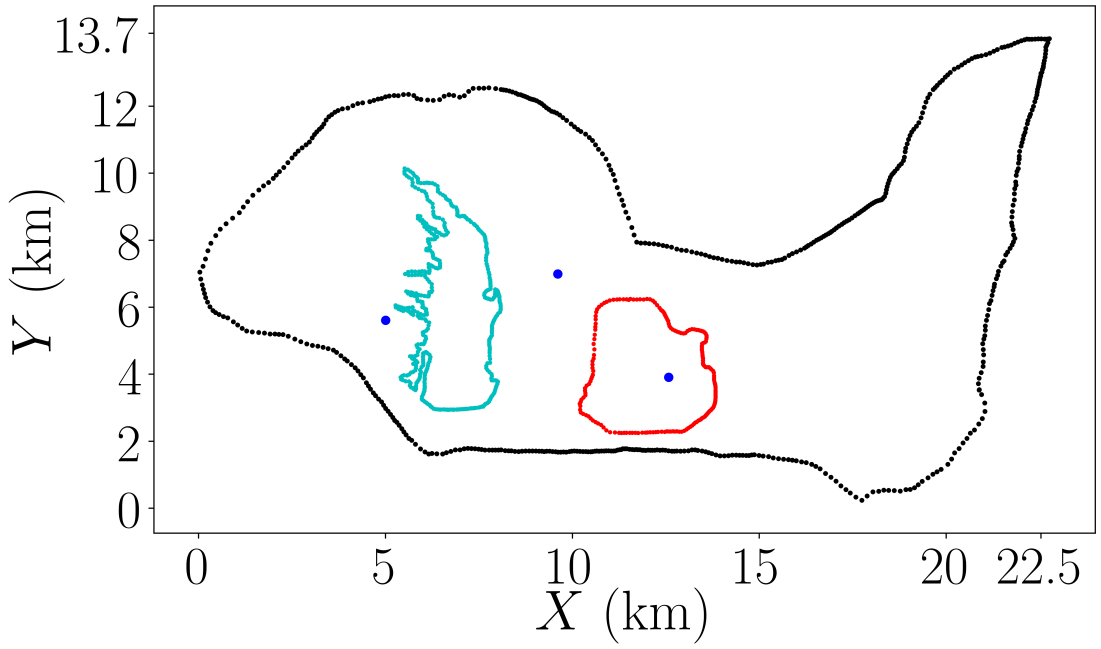


Figure 2.2: Diagram of the mining facility, where the black dots represent the edge of the facility, the red dots represent the outline of the open-pit mine, the teal dots represent the outline of the tailings pond, and the blue dots represent where the TANAB2 was deployed during the field environmental monitoring campaign in May 2018.

A typical TANAB2 deployment included the controlled release of the TANAB2 and gondola from the Earth's surface, up to a maximum altitude of 200 m above ground level. Using a manually controlled reel and rope tether, the TANAB2 was released and retrieved at a constant rate. In general, one profile and retrieval of the TANAB2 lasted a total of one hour. The maximum altitude of each specific profile varied as a function of environmental conditions. During periods of increased wind velocities, up to three mooring lines were attached to the TANAB2 and controlled by personnel on the ground. The use of mooring lines allowed the TANAB2 to be deployed in environments with a maximum wind speed of 10 m s^{-1} (see Figure 2.3). The addition of each rope resulted in a lower launch altitude (and ultimately

less mapped area) due to addition of weight during periods of atmospheric instability such as the afternoons. This trade-off was deemed acceptable as it was imperative to launch the TANAB2 in both stable and unstable atmospheric conditions to successfully map diurnal surface temperatures. The TANAB2 was launched a total of 12 times at the mining facility (Figure 2.2), recording approximately 50 hours of TriSonica^{TM} data. The details of each deployment is noted in Table 2.1 below.

The DJI Zenmuse XT was also deployed either on top of stacked crates or on top of a ladder deployed behind a stationary vehicle. In both instances, the camera had an altitude above ground of approximately 2 m. The details of each surface-based thermal camera deployment are included in Table 2.1 where the number of TANAB2 profiles is not specified.

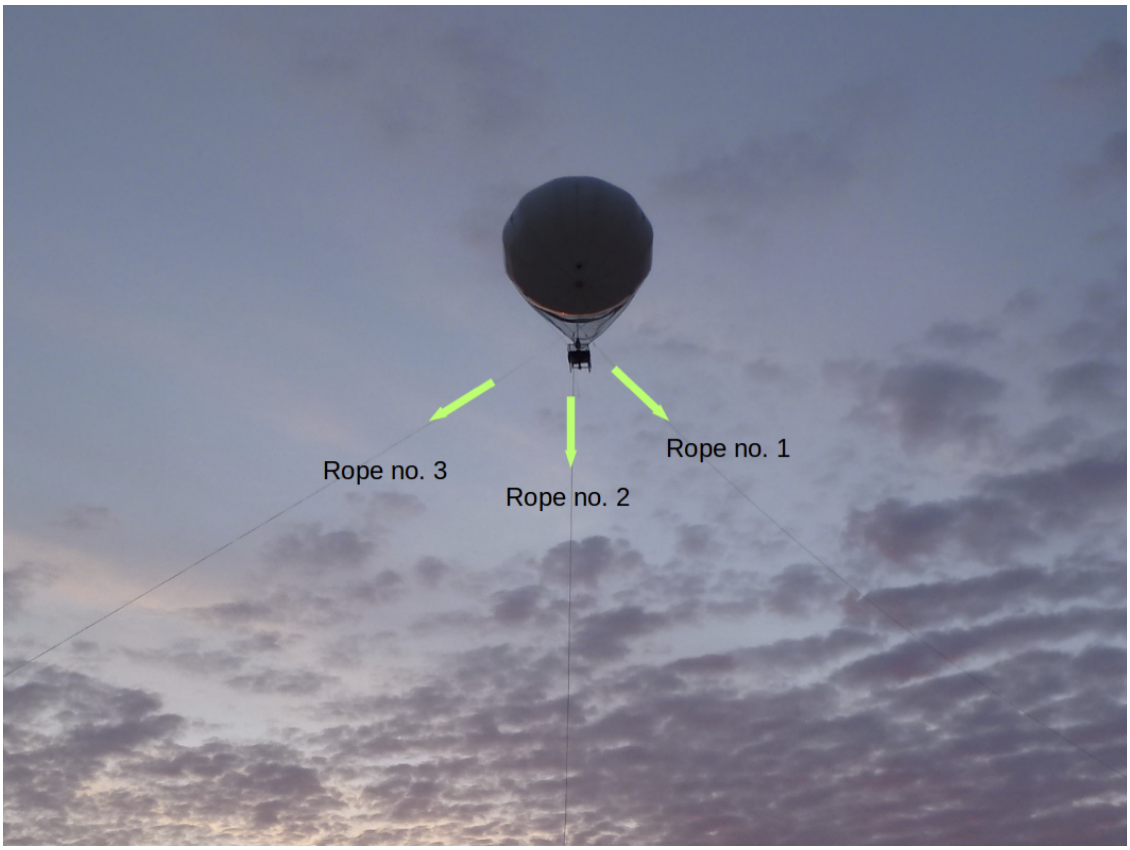


Figure 2.3: Use of three ropes controlled by personnel on the ground during a launch of the TANAB2 at the mining facility in May 2018.

Table 2.1: TANAB2 mining facility launch details. Times are in Local Daylight Time (LDT).

Experiment	Location	Start date	Start time	End time	No. Profiles	Duration
1	Tailings pond	2018/05/05	13:07:00	17:00:00	-	04:07:00
2	Tailings pond	2018/05/06	09:55:00	16:45:00	-	06:50:00
3	Tailings pond	2018/05/07	21:41:00	02:47:00	14	05:06:00
4	Tailings pond	2018/05/09	03:30:00	04:00:00	02	00:30:00
5	Tailings pond	2018/05/10	02:30:00	08:30:00	21	06:00:00
6	Tailings pond	2018/05/11	18:28:00	23:00:00	-	04:32:00
7	Tailings pond	2018/05/12	12:02:00	12:44:00	-	00:42:00
8	Tailings pond	2018/05/13	04:12:00	09:13:00	-	05:01:00
9	Tailings pond	2018/05/15	04:55:00	11:00:00	22	06:05:00
10	Tailings pond	2018/05/17	05:37:00	09:00:00	-	03:23:00
11	Mine	2018/05/18	04:12:00	11:12:00	20	07:00:00
12	Mine	2018/05/19	18:52:00	23:15:00	17	04:23:00
13	Mine	2018/05/21	11:00:00	12:17:00	04	01:17:00
14	Mine	2018/05/23	01:47:00	05:30:00	10	02:43:00
15	Mine	2018/05/24:	11:19:00	14:25:00	12	03:06:00
16	Mine	2018/05/27	14:38:00	17:50:00	18	03:12:00
17	Tailings pond	2018/05/30	10:55:00	18:57:00	24	08:02:00
18	Tailings pond	2018/05/31	11:07:00	14:43:00	08	03:36:00

2.2.2 Guelph Campaign

The TANAB2 was launched at the University of Guelph, Guelph, Ontario, Canada on July 28, 2018 and August 13, 2018. The launches occurred in the Reek Walk, a conventional two-dimensional urban canyon, at 43.5323° N and 80.2253° W [63]. Specific launch site geographical features are illustrated in Figure 2.4, where the yellow dot represents the Reek Walk TANAB2 launch site and other proximal buildings are labelled appropriately. Thermal images were captured between 05:04 and 21:21 Eastern Daylight Time (EDT) on July 28$^{\text{th}}$ and between 05:20 and 20:31 EDT on August 13$^{\text{th}}$. This field campaign seeked to identify the diurnal ST variation of the campus buildings with respect to adjacent green spaces at a high spatiotemporal resolution. Images were captured in the same manner as described in Section 2.2.1. The details of the two TANAB2 deployments are described in Table 2.2.

Table 2.2: TANAB2 launch details for July, 28, 2018 and August 13, 2018 University of Guelph, Guelph, Ontario, Canada deployments. Times are in Eastern Daylight Time (EDT).

Exp.	Location	Start date	Start time	End time	No.Profiles	Experiment time
1	Reek Walk	2018/07/28	02:04:00	21:31:00	20	19:27:00
2	Reek Walk	2018/08/13	04:49:00	20:36:00	12	15:47:00

Figure 2.4: Diagram of TANAB2 launches in relation to other notable buildings at Reek Walk, University of Guelph, Guelph, Ontario, Canada.

2.3 Image Processing Method Development

The Python-based image processing workflow was created using Python 3.5 on Ubuntu 16.04 and associated open-source software, including ExifTool 10.94[4] and ImageMagick 7.0.7.[5] Commands derived from these programs were executed through the Linux terminal window within the Python script. Data recorded by the integrated camera and flight controller system on the TANAB2 were stored within each image file. This data was utilised within the developed mathematical calculations. A process flow diagram for the image processing workflow is displayed in Figure 2.5. Each step is discussed in detail in the following paragraphs.

The image processing workflow includes two general functions, one to directly georeference image pixels and the other to calculate ST from selected image pixels. These two functions will be discussed in detail separately. The process utilised to conduct the Principal Component Analysis (PCA) of surface temperatures is also detailed.

ExifTool was used to extract and assign the gondola's longitude and latitude coordinates, camera gimbal's Roll, Yaw, and Pitch angles, and gondola's Roll, Yaw, and Pitch angles, located in the metadata of each image, to variables in Python. A few images were removed

[4]https://www.sno.phy.queensu.ca/~phil/exiftool/
[5]https://www.imagemagick.org/

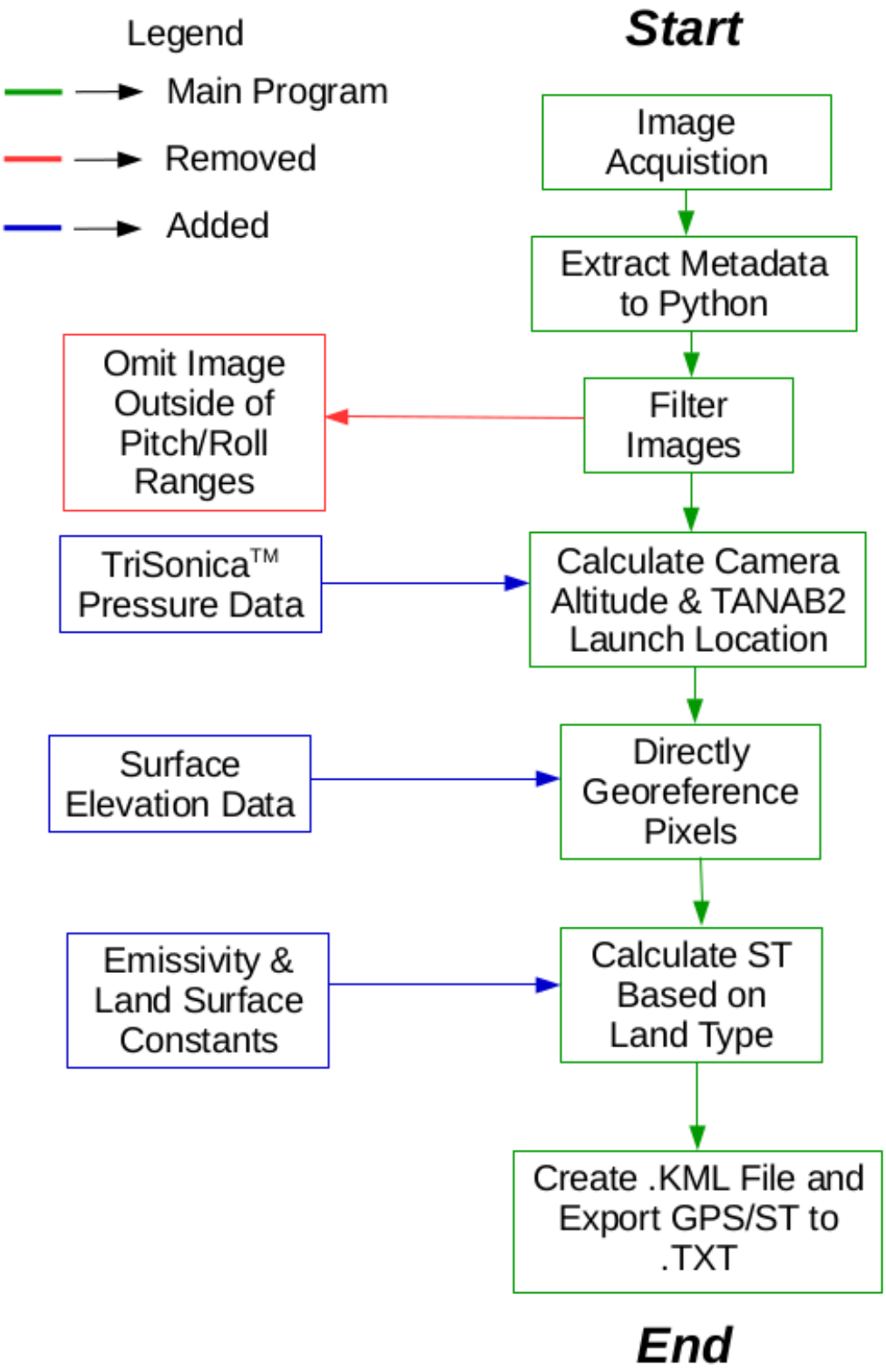


Figure 2.5: Process flow diagram of the image processing workflow.

from the workflow due to excessive angles of the gondola or camera. The camera Roll angle did not significantly impact the method as the Zenmuse XT was self stabilised. However, if the gondola Roll degree was greater than 45° or less than −45°, the camera became destabilised. Images with gondola Roll angles outside of this range were omitted from the workflow. Furthermore, the mechanical Pitch range of the camera was noted to be between 45° and −135°. Gimbal Pitch angles greater than 0° primarily included images of the sky, gimbal Pitch angles equivalent to 0° were images of the horizon and gimbal Pitch angles less than 0° included images primarily of the ground. It was determined that the recorded camera gimbal Pitch angle corresponded to the Pitch angle for the middle of each image. Any images with a gimbal Pitch angle greater than or equal to −2° were omitted from the image processing analysis. Furthermore, very oblique pitch angles, greater than −30° from the horizon, were noted to possibly introduce errors into the ST calculations[6] but were not necessarily eliminated. Based on physical parameters of the camera, including the Vertical and Horizontal Fields Of View ($VFOV$ and $HFOV$) angles, images with a gimbal Pitch angle less than or equivalent to −76° were also removed from the analysis. This filter was chosen because the bottom of the image would have a corresponding Pitch angle of the recorded gimbal Pitch angle, plus one half of the $VFOV$ that would result in an angle close to or less than −89° which may disrupt direct georeferencing calculations. The Pitch angles filtered are related to the compromise between ST spatial distribution and ST accuracy because the TANAB2 only reached a maximum altitude of 200 m above ground level.

2.3.1 Georeferencing

As reported in literature, the Global Positioning System (GPS)-sensor-derived altitude can vary significantly up to 50 m as stated by Eynard et al. [24]. Padró et al. [70] reported a vertical RMSE of 4.21 m for a system that used data collected from the GNSS system of the UAV deployed in their experiment. The TANAB2 system utilised the DJI N3 flight controller which includes a GNSS-Compass unit (a GPS module is included within this system).[7] Since an accurate measurement of TANAB2 gondola altitude was required for direct georeferencing of thermal images with the developed method, the hypsometric equation was used to calculate altitude for images recorded during TANAB2 launches. Images recorded from surface-based structures (on crate or ladder) were assumed to have an altitude of 2 m. The hypsometric equation (Equation 2.1) uses atmospheric pressure and accounts for atmospheric temperature

[6] https://dl.djicdn.com/downloads/zenmuse_xt/en/sUAS_Radiometry_Technical_Note.pdf
[7] http://dl.djicdn.com/downloads/N3/N3+User+manual.pdf

changes within the formula [8, 88]

$$z_2 - z_1 \approx a\overline{T_v} \ln\left(\frac{P_1}{P_2}\right), \tag{2.1}$$

where z_1 and z_2 represent the altitudes (in meters) corresponding to the recorded pressure measurements (in mBar, however the units of pressure do not affect this equation), P_1 and P_2, $\overline{T_v}$ represents the average virtual temperature between the two altitudes (z_1 and z_2), and a is a constant equivalent to 29.3 m K^{-1} [88]. The gondola altitude was calculated in the code in a similar manner as described in Sections A.2.1 and A.3.2 for the mining facility and the Guelph campaigns respectively. For the Guelph campaign, the ascending TANAB2 indices were identified using the code in Section A.3.1. The uncertainty of error for Equation 2.1 was quantified using Equations 2.2, 2.3, and 2.4 [50]. A sample calculation was completed using the theory of error propagation, where P_2 is 100 kPa, P_1 is 101.3 kPa, and $\overline{T_v}$ is 300 K. The atmospheric pressure and temperature measurements were obtained from the TriSonica™ anemometer where the pressure measurement had an uncertainty of 0.01 kPa and the temperature measurement had an uncertainty of 2 K. The uncertainty calculated was 1.2 m using

$$\Delta z_2 = \sqrt{\left(\frac{\partial z_2}{\partial \overline{T_v}}\right)^2 \Delta \overline{T_v}^2 + \left(\frac{\partial z_2}{\partial P_2}\right)^2 \Delta P_2^2}, \tag{2.2}$$

$$\frac{\partial z_2}{\partial \overline{T_v}} = a \ln\left(\frac{P_1}{P_2}\right), \tag{2.3}$$

$$\frac{\partial z_2}{\partial P_2} = a\overline{T_v}\left(\frac{-1}{P_2}\right). \tag{2.4}$$

Note that since differential altitude from the ground is desired, the appropriate uncertainty for the pressure is the resolution of the measurement. With this known uncertainty, this method was deemed acceptable over using the raw GPS altitude data provided by the DJI N3 flight controller unit.

All recorded TriSonica™ data were averaged to the nearest whole second (see Section A.2.1 for detailed code). The averaging procedure was replicated for surface-based thermal camera deployments. For each image, the corresponding day of year in seconds was calculated and the altitude index with the smallest difference between the TriSonica™ and image day of year in seconds was selected. This altitude was referred to as the altitude in meters of the

camera gimbal above ground level.

With the altitude of the camera gimbal known, trigonometric relationships were derived to calculate the geographic coordinates of the four corners, the four midpoints, and centre of each projected image on the surface of the Earth. The gimbal pitch angles for the top and bottom of each image were calculated by adding and subtracting half of the $VFOV$ to the gimbal pitch angle, respectively. If the top pitch angle was greater than or equal to $-1°$, the top pitch angle was adjusted to equal $-1°$ to ensure that all image pixels included the Earth's surface. All angles used in the georeferencing calculations were converted to radians.

In total, the TANAB2 was launched at three locations during the entire field campaign at the mining facility. The TANAB2 was only deployed at a maximum of one location each day. Using Google Earth, the surface elevation above sea level was calculated for each launch location at the mining site. For images collected during the Guelph TANAB2 campaign at the University of Guelph, a Digital Surface Model (DSM) derived from the Panchromatic Remote-sensing Instrument for Stereo Mapping (PRISM) sensor on board the Advanced Land Observing Satellite (ALOS) was used.[8] Version 2.1 (April 2018) of the ALOS PRISM DSM file (30 m spatial resolution) was used when extracting surface elevations in Guelph, Ontario, Canada. Using a variation of the Haversine formula, the distance between the gondola coordinates (lat_2 and lon_2) and each of the three launch locations at the mining facility (lat_1 and lon_1) were calculated, and the minimum distance was chosen, for which a base altitude from Google Earth was assigned. The distance is calculated using

$$d_{\text{launch}} = R[2\text{atan}^2(\sqrt{\sin^2\left(\frac{\text{lat}_2 - \text{lat}_1}{2}\right) + \cos(\text{lat}_1)\cos(\text{lat}_2)\sin^2\left(\frac{\text{lon}_2 - \text{lon}_1}{2}\right)}$$
$$, \sqrt{(1 - (\sin^2\left(\frac{\text{lat}_2 - \text{lat}_1}{2}\right) + \cos(\text{lat}_1)\cos(\text{lat}_2)\sin^2\left(\frac{\text{lon}_2 - \text{lon}_1}{2}\right)})})], \qquad (2.5)$$

where R represents the Equatorial Radius of the Earth in kilometres.[9] The geographic location associated with the smallest value of d_{launch} was determined to be the TANAB2 deployment location. With the smallest distance known, the appropriate base altitude in meters above sea level was assigned for each image.

The direct georeferencing workflow considers surface elevation with respect to geographic distance away from each TANAB2 launch location for the eight cardinal directions (north,

[8]https://www.eorc.jaxa.jp/ALOS/en/aw3d30/index.htm
[9]https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

north-west, west, south-west, south, south-east, east, and north-east) and the line of sight from the camera for a given image pixel. Land surface elevation data in meters above sea level for the eight cardinal directions up to 10 km away from each TANAB2 deployment location at the mining facility, were obtained from the Geocontext-Profiler[10] and saved as individual text files. The camera gimbal Yaw angles were recorded in degrees positive clockwise from north. If the Yaw angles were negative, 360° was added to the gimbal Yaw angle. Based on the Yaw angle of the camera gimbal and the base altitude, the appropriate file, containing data from the Geocontext-Profiler, was loaded into the Python script and a third order polynomial was fitted to the data. Third order models have been used to represent curved Earth surfaces ([81]), such as those encountered in this mining facility.

The line of sight for a given image pixel was constructed by calculating the slope, which is represented by the tangent of the pitch angle. For example, for the top centre pixel, the tangent of the top pitch angle is the slope of the line of sight for the top pixel, equal to the TANAB2 altitude divided by the horizontal distance from the TANAB2 launch location to where the line of sight intercepts the horizontal axis. This slope is negative because the camera's line of sight is always below the horizon.

From the derived third order polynomial for land surface elevation, the horizontal distance from the TANAB2 to where the image pixel is positioned was determined (d_{horiz}). The roots of the intersection of the polynomial curve and the line of sight give the horizontal distance. If the roots were not real, the specific image was omitted from the ST calculation process. If multiple roots were found, the smallest real positive solution was chosen. For the mining facility, if d_{horiz} was greater than 100 km, the pixel was omitted from the analysis. Likewise, for the Guelph campaign, if d_{horiz} was greater than 5 km, for the top centre and centre of an image especially, the pixel was omitted from the analysis. If the bottom centre of an image had a horizontal distance away from the TANAB2 above 5 km, the image was omitted from the analysis. This value was chosen as the TANAB2 was launched in an urban canyon around numerous multi-story buildings where the camera line of sight likely would have intersected an object at least 5 km away. This condition was implemented in the event the ALOS DSM file spatial resolution did not fully consider building heights on the University of Guelph campus. As a result, this filter value could be changed depending on the imaging location.

With the horizontal distance from the TANAB2 to where the image pixel is located known, the geographic coordinate pair for the corresponding horizontal (left to right) and vertical (top to bottom) pixel locations in the image were calculated using a variation of the Haversine

[10]http://www.geocontext.org/publ/2010/04/profiler/pl/

formula

$$\mathrm{Lat}_2 = \mathrm{asin}[\sin(\mathrm{Lat}_1)\cos\left(\frac{d_{\mathrm{horiz}}}{R}\right) + \cos(\mathrm{Lat}_1)\sin\left(\frac{d_{\mathrm{horiz}}}{R}\right)\cos(\mathrm{Yaw})] \tag{2.6}$$

$$\begin{aligned}\mathrm{Lon}_2 = \mathrm{Lon}_1 + \mathrm{atan}^2([\sin(\mathrm{Yaw})\sin\left(\frac{d_{\mathrm{horiz}}}{R}\right)\cos(\mathrm{Lat}_1)] \\ , [\cos\left(\frac{d_{\mathrm{horiz}}}{R}\right) - \sin(\mathrm{Lat}_1)\sin(\mathrm{Lat}_2)]),\end{aligned} \tag{2.7}$$

where Lat_2 and Lon_2 represent the geographic coordinates for the projected image pixel pair, Lat_1 and Lon_1 represent the geographic coordinates of the TANAB2 gondola when the image was recorded, and d_{horiz} represents the horizontal distance the projected image pixel is away from the TANAB2 in kilometres. The geographic coordinates for the top centre, middle, and bottom centre of each image was calculated.

When determining the geographic coordinates for the image corners and edge midpoints, the geographic distance from the TANAB2 and the edge of the image was calculated

$$d_{\mathrm{edge}} = \frac{d_{\mathrm{horiz}}}{\cos(0.5HFOV)}, \tag{2.8}$$

where d_{edge} represents the geographic distance in kilometres from the TANAB2 to the top, middle, and bottom of the projected image edge (both left and right edges), and $HFOV$ represents the camera Horizontal Field Of View. In total, three d_{horiz} values were used, one for pixels at the top of the image, another for pixels in the middle of the image, and one for pixels at the bottom of the image. With d_{edge} known, d_{horiz} is replaced accordingly such that geographic coordinate pairs along edges of each image can be calculated.

With the coordinate pairs of midpoints of the centre, edges, and corners of each image calculated, pixels within the image matrix were georeferenced and the corresponding ST values were calculated accordingly. For instances, where a new pitch angle for the top of the image was assigned (for images whose portion of the top needed to be eliminated), a mathematical relationship was derived to quantify which image pixel rows from the top were to be omitted from the image processing analysis.

Figure 2.6 provides an illustration for the angles used to correlate image pixel position to geographic coordinate location. $P0$, Px, $P256$, and $P512$ represent the top, the new top, the centre, and the bottom pixel rows, respectively, as each image has 640 horizontal pixels

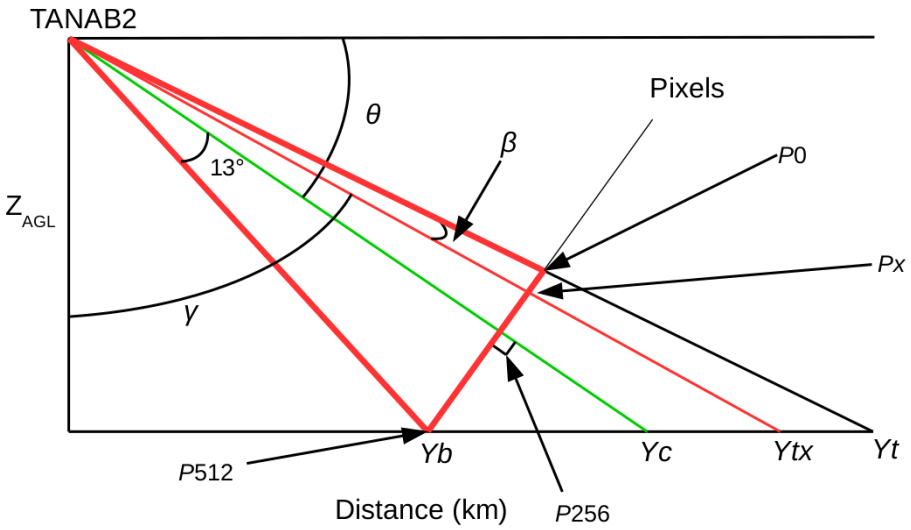


Figure 2.6: Relationship between pixels and horizontal geographic distances.

and 512 vertical pixels.[11] Yt, Ytx, Yc, and Yb represent the horizontal geographic distances away from the TANAB2 for the top, new top, centre, and bottom of each image, respectively. θ represents the camera gimbal pitch angle, 13° is half of the $VFOV$, and γ and β are angles that are used to correlate pixels to distances using

$$\gamma = \text{atan}\left(\frac{d_{\text{horiz}_{\text{top}}}}{z_{\text{AGL}}}\right), \tag{2.9}$$

$$\beta = 90° - \mid \theta \mid + 0.5VFOV - \gamma, \tag{2.10}$$

$$X = \frac{0.5VPR\sin(\beta)}{\sin(0.5VFOV)\sin(180° - \eta)}, \tag{2.11}$$

where VPR is the Vertical Pixel Range (512 based on the camera specifications). Note $d_{\text{horiz}_{\text{top}}}$ is the horizontal distance that the TANAB2 makes with the land location associated with the top of the image, and z_{AGL} is altitude above ground level. The new top pixel, X, was derived from Figure 2.7 by applying the sine law and rearranging the equation. Figure 2.7 displays the red triangle, illustrating the vertical camera field of view from the TANAB2 in greater detail. X represents the number of pixel rows to omit from the top of the image which is a function of the new top angle in the instance that the top pitch angle is greater

[11] https://www.dji.com/ca/zenmuse-xt/specs

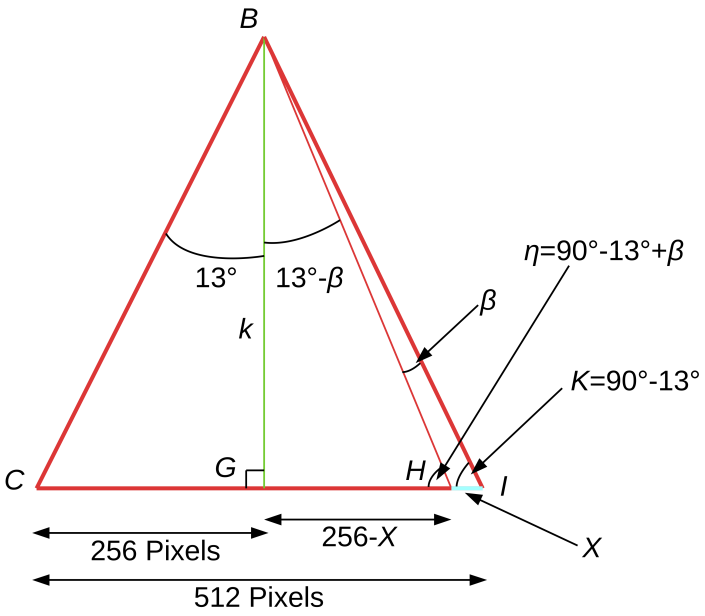


Figure 2.7: Relationship between vertical image pixels and the camera $VFOV$.

than $-1°$.

Considering all image pixels was not a possibility due to the extreme volume of computational operations required. Instead, a geometric step function was implemented to identify which pixel rows to consider for the ST calculation. This was motivated by the fact that pixels near the top of the image correspond to more land surface coverage, so they must be analysed at higher resolution. The geometric step function was achieved using the following equation and coefficients

$$y_{\text{pixel}} = 18(1.41)^n, \tag{2.12}$$

where n is an integer that ranges from 2 to 10. The first two pixel rows to process were chosen to be 0 and 18. Equation 2.12 yields the corresponding pixel rows: 35, 50, 71, 100, 141, 199, 281, and 396. These rows were selected because they are densely packed in the top half of each image. The geographic distance between pixel rows at the top of an image would be greater than the geographic distance of pixel rows at the bottom of an image. To optimise processing efficiency and increase ST spatial distribution, the coefficients in Equation 2.12 were chosen to satisfy the desired image processing criteria. Depending upon application, this pixel row processing workflow can be changed to increase or decrease the number of pixels used in the ST calculations accordingly. The horizontal pixel step was set at a fixed

value of 64. For each image, every 64$^{\text{th}}$ pixel column was used in the ST calculation.

When iterating through the image pixel matrix, β and γ angles were calculated for each pixel coordinate and a corresponding camera line of sight was calculated. Based on the relationship identified in Figures 2.6 and 2.7, the sine law was employed in Equation 2.13 to derive Equation 2.14. The resulting slope of the camera line of sight for each pixel coordinate pair were used to relate the pixel pairs to geographic distances used to georeference pixels within the image as completed with Equations 2.6 and 2.7

$$\frac{\sin(0.5VFOV)}{\sin(\kappa)[0.5VPR]} = \frac{\sin(0.5VFOV - \beta)}{\sin(\eta)[0.5VPR - j]}, \tag{2.13}$$

$$\beta = -\text{atan}\left(\frac{[0.5VPR - j]\sin(0.5VFOV)}{0.5VPR\sin(\kappa)}\right) + 0.5VFOV, \tag{2.14}$$

$$\gamma = 90° - \mid \theta \mid + 0.5VFOV - \beta, \tag{2.15}$$

$$\text{Slope} = \frac{-1}{\tan(\gamma)}. \tag{2.16}$$

Here, j is the location of the pixel coordinate row from top to bottom. With the slope for the camera line of sight for a particular pixel coordinate pair known, the intersections between the third order model representing the land surface and the pixel line of sight were derived. The smallest real positive solution was chosen to be the projected horizontal distance away from the TANAB2 for pixels located in the middle column (pixel 320 from left to right) of the image. Pixels horizontally adjacent to the centre of an image required an adjustment to the heading degree (Yaw). The offset angle (α) to apply to the heading degree was determined to be function a of the $HFOV$ of the camera. Depending on the location of the pixel coordinate column i from left to right, the angular offset formula varied

$$\alpha_{i=0} = \frac{-HFOV}{2}, \tag{2.17a}$$

$$\alpha_{i>0,i<320} = \frac{-\left(\frac{HPR}{2} - i\right)HFOV}{HPR}, \tag{2.17b}$$

$$\alpha_{i=320} = 0, \tag{2.17c}$$

$$\alpha_{i>320,i<640} = \frac{\left(i - \frac{HPR}{2}\right)HFOV}{HPR}, \tag{2.17d}$$

$$\alpha_{i=640} = \frac{HFOV}{2}, \tag{2.17e}$$

where HPR represents the Horizontal Pixel Range (640 pixels based on the physical camera specifications). The Yaw heading of the camera gimbal corresponded to the middle of the image. As a result the heading for any pixels to the left of the centre of the image required the angular offset to be removed from the recorded Yaw. Likewise, any pixels to the right of the image required the angular offset to be added to the recorded Yaw. In cases where the addition of the angular offset to the heading angle resulted in a negative value or a value greater than 2π radians, then 2π radians were either added or subtracted, respectively, to ensure that only positive angles between 0 and 2π radians were passed to Equations 2.6 and 2.7.

The code for the direct georeferencing method is included in Sections A.2.4 and A.3.4 for the mining facility and the Guelph campaigns.

2.3.2 Thermal Camera Calibration

Using ExifTool and ImageMagick, recorded signal values from individual pixels were extracted and saved to a matrix in the Python script. These raw signal values were converted to surface temperatures considering a variation of Planck's Law.

Due to field conditions and physical limitations encountered at the mining facility, errors introduced from reflections and transmission could not be accounted for. However, the thermal camera used in the field campaign was calibrated in a pre-field outdoors experiment on campus at the University of Guelph, Guelph, Ontario, Canada. Three radiometric images were captured roughly thirty seconds apart for every hour between 06:00 and 23:00 EDT over two consecutive days. The thirty second time interval was selected as Olbrycht and Więcek [69] noted that uncooled thermal cameras can experience temperature drift up to 1 K per minute if a radiometric calibration was not recently completed. Four different land surface types were imaged including water, soil, grass, and developed land (urban surfaces). Each image included a certified thermometer which measured the corresponding land surface temperature as recorded by the image.

Surface temperatures from the top of the thermometer were calculated from the thermal images using FLIR Tools. For each hourly image set, the average of the surface temperatures derived in FLIR Tools was calculated and used to calibrate the R, B, O, and F constants accordingly, where $R = \frac{R_1}{R_2}$. The temperatures recorded by the certified thermometer were scaled to adjust for the test location's height above sea level (334 m for Guelph, Ontario,

Canada). For the thermal image temperatures, (U_{Obj}) was calculated from Equation 2.18.

$$U_{\mathrm{Obj}} = \frac{R}{\exp\left(\frac{B}{T_{\mathrm{Obj}}}\right) - F} - O, \tag{2.18}$$

where U_{Obj} represents the radiative energy emitted from the imaged object, T_{Obj} represents the surface temperature of the imaged object derived from FLIR Tools, R represents the uncooled camera response, B is a constant related to Planck's radiation law, F relates to the non-linear response of the thermal imaging system, and O represents an offset [12]. Equation 2.18 can be rearranged to calculate T_{Obj} as per Equation 2.19.

$$T_{\mathrm{Obj}} = \frac{B}{\ln\left(\frac{R}{U_{\mathrm{Obj}}+O} + F\right)}. \tag{2.19}$$

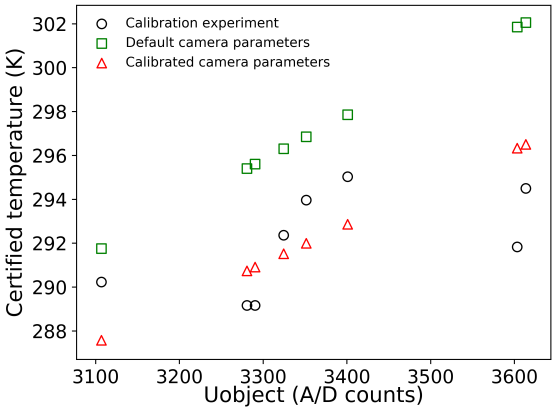
The R, B, O, and F constants used to calculate the U_{Obj} value were the default constants stored in the metadata of each thermal image. The default constants and calibrated constants are displayed in Table 2.3.

Using the empirical line method, described by Smith and Milton [85], the U_{Obj} values and the corresponding certified thermometer temperatures were plotted against each other to calibrate the constants used in Equation 2.19 as a function of land surface type. The figures illustrating the empirical line method are displayed in Figure 2.8. The Non-Linear Least-Squares Minimization and Curve-Fitting (LMFIT) of Python library version 0.9.13[12] was used with Equation 2.19 to fit and optimise the camera constants while minimising residuals for each specific land surface type.

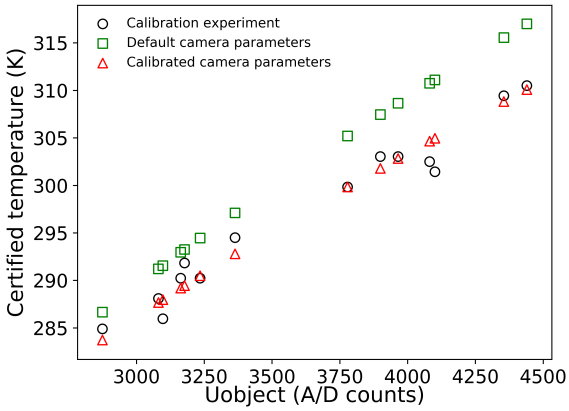
Using the LMFIT library to fit camera constants for each land surface ultimately reduced the bias and RMSE values when compared to the default camera constants shown in Table 2.4. Using the calibrated constants for the calculation of land ST at the mining facility improved accuracy of the measurement. These findings are comparable to Gallardo-Saavedra et al. [27] who reported that the manufacturer stated accuracy of the FLIR Vue Pro R 640, Tau 2 640, and Zenmuse XT 640 was ±5 K. Similarly, Kelly et al. [46] used the empirical line calibration method for a FLIR Vue Pro 640 uncooled thermal camera and quantified the accuracy of the camera to be ±5 K.

The Python code used to calculate the calibrated camera constants and the plots in Figure 2.8, are located in Sections A.1 and A.1.1 respectively.

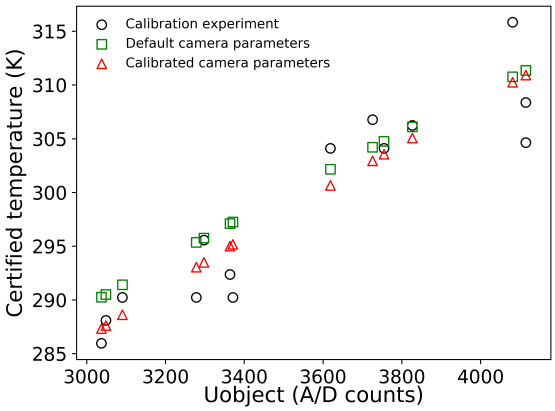
[12] https://lmfit.github.io/lmfit-py/index.html



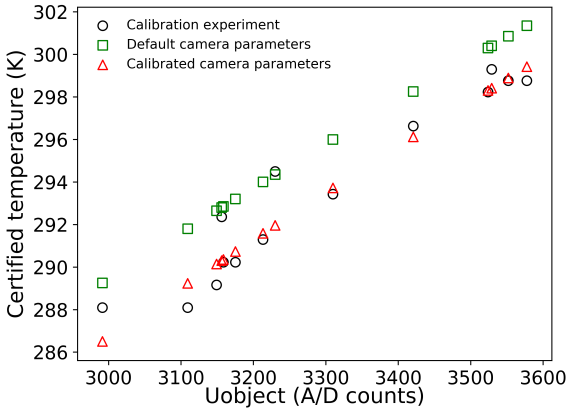
(a) Default and calibrated temperature for water.



(b) Default and calibrated temperature for soil.



(c) Default and calibrated temperature for developed land.



(d) Default and calibrated temperature for grass.

Figure 2.8: Certified temperature compared to radiometric image pixel signal value for water, soil, developed land, and grass.

Table 2.3: Default and calibrated camera parameters.

Camera parameters	R	B	O	F
Default	366545	1428	-342	1
Calibrated water	549789	1507	-171	1.5
Calibrated soil	549800	1510	-171	1.5
Calibrated developed land	247614	1322	-513	1.5
Calibrated grass	314531	1391	-513	1.5

Table 2.4: Default and calibrated camera parameter statistics.

Surface	Water	Soil	Developed land	Grass
Default bias (K)	5.18	4.81	1.83	2.07
Default RMSE (K)	5.83	5.34	3.91	2.34
Calibrated bias (K)	0.27	-0.09	0.13	-0.24
Calibrated RMSE (K)	2.40	1.57	3.31	1.11

Surface Temperature Calculation

Camera constants were applied to the surfaces within the mining facility with geographical coordinates closest to the calibrated land use categories. The effect of emissivity was considered by using the BroadBand Emissivity as described by Wang et al. [100] and calculated in Equation 2.20.

$$BBE = a\epsilon_{29} + b\epsilon_{31} + c\epsilon_{32}, \tag{2.20}$$

where a, b, and c are constants that vary as functions of land surface, and ϵ_{29}, ϵ_{31}, and ϵ_{32} are emissivities derived from the MOD11B3 MODIS data product from bands 29, 31, and 32 respectively. Wang et al. [100] determined that a, b, and c constants are similar for vegetation, soil, and anthropogenic materials. As a result, the a, b, and c coefficients were selected to be 0.2122, 0.3859, and 0.4029, respectively [100].

The code in Section A.2.2 was used to create a point grid with a spatial resolution of 500 m over the mining facility and the code in Section A.3.3 was used to create a point grid over the University of Guelph campus. These point grids were imported into QGIS and overlaid on the MODIS MOD11B3 files. The emissivity values from the three bands in addition to the latitude and longitude coordinates were extracted for each point and saved to a file. These files were used in Sections A.2.4 and A.3.4 to quantify the BBE for the mining facility and the University of Guelph campus respectively. The mining facility had two MOD11B3 MODIS images overlapping the site as a result, the code in Section A.2.3 was used to extract the

emissivity band data from the original satellite images. This problem was not encountered for the Guelph field campaign.

The total signal (U_{Tot}) recorded by the uncooled thermal camera can be separated into three components as in Equation 2.21. The first component represents the radiative energy emitted from the imaged object (U_{Obj}), the second component represents the reflected energy from the imaged object (U_{Refl}), and the third component accounts for the radiative energy transmitted from the atmosphere (U_{Atm}). ϵ represents the emissivity of the surface and is accounted for by Equation 2.20 and τ represents the transmissivity of the atmosphere whose value is generally close to 1.0 [93]. As a result, only the radiative energy reflected and emitted from the imaged object are considered in Equation 2.21, where to retrieve U_{Obj} and subsequently T_{Obj}, U_{Refl} is removed from U_{Tot}. The calibration of camera constants was completed to correct for incoming reflected radiation via

$$U_{\text{Tot}} = \epsilon\tau U_{\text{Obj}} + \tau(1-\epsilon)U_{\text{Refl}} + (1-\tau)U_{\text{Atm}}. \tag{2.21}$$

These calculations are detailed in Sections A.2.4 and A.3.4 corresponding to the mining facility and the Guelph campaign. On average, it takes 17.8 seconds to directly georeference and calculate surface temperature from one image.

2.4 Principal Component Analysis (PCA)

In order to determine the geographical direction for which one has the largest variations in surface temperature, a PCA was performed. PCA is a very well-known approach for analysing data (especially large data) to deduce meaningful conclusions about it. The principle behind PCA lies in the fact that it can mathematically determine the principal components (eigenvectors) showing the directions of the largest deviations in the data; for more information about PCA see for instance the work of Jolliffe [42]. Note that this method gives the main axes along which the variations in the data are the largest. In this analysis, it was of interest to find the direction of the land surface for which the temperature gradient was the largest. Therefore, the axis that had the most variation was picked and the results were analysed accordingly. PCA was completed for six four-hour time intervals.

The Python code created to calculate and plot the PCA is located in Section A.2.8.

Chapter 3

Results and Discussion

3.1 Mining Site Campaign

Three analyses were conducted on the processed image data. The first analysis represents median ST distribution at a spatial resolution of 1 km × 1 km derived from images recorded over the entire length of the field campaign. In total, six four-hour time intervals in Local Daylight Time (LDT) (00:00-04:00 LDT, 04:00-08:00 LDT, 08:00-12:00 LDT, 12:00-16:00 LDT, 16:00-20:00 LDT, and 20:00-24:00 LDT) representing ST for the entire field campaign were produced highlighting diurnal ST variation with respect to the mining facility boundary, the mine, and the tailings pond as displayed in Figure 3.1. Corresponding box plots representing the temperature variation of the mine and tailings pond are also included for each time interval as per Figure 3.2. For each survey, on average 1910 images were used for each four-hour time interval.

The second analysis focuses on comparing the calculated ST derived from the images collected on May 24, 2018 over the 12:00-14:00 LDT time interval with respect to the MODIS MOD11A1 image recorded during the early afternoon on May 24, 2018. Three plots were created (as per Figure 3.3) including the ST spatial distribution map at 1 km × 1 km resolution derived from the workflow, the MOD11A1 dataset for each corresponding ST tile, and the absolute error for each tile is included.

The third analysis focuses on identifying horizontal direction of the highest surface temperature variances. The direction with the highest surface temperature variances for each time interval was calculated from the images collected during the field campaign by completing a PCA on the data derived from each time interval. The results are presented in Figure 3.4.

The code used to separate the data into six four-hour intervals is included in Section A.2.5, the code to correct surface temperatures as a function of land material type is included in Section A.2.6, and the code used to create the surface temperature maps and box plots is included in Section A.2.7.

3.1.1 Diurnal Surface Temperature

Surface temperature maps with a spatial resolution of 1 km × 1 km for the entire mining facility at six four-hour time intervals are displayed in Figure 3.1. These plots were created by calculating the median temperature for all data recorded within each time interval over the entire field campaign within a 1 km × 1 km area (tile). The axes represent distance in kilometres and the colour bar represents surface temperature in Kelvin.

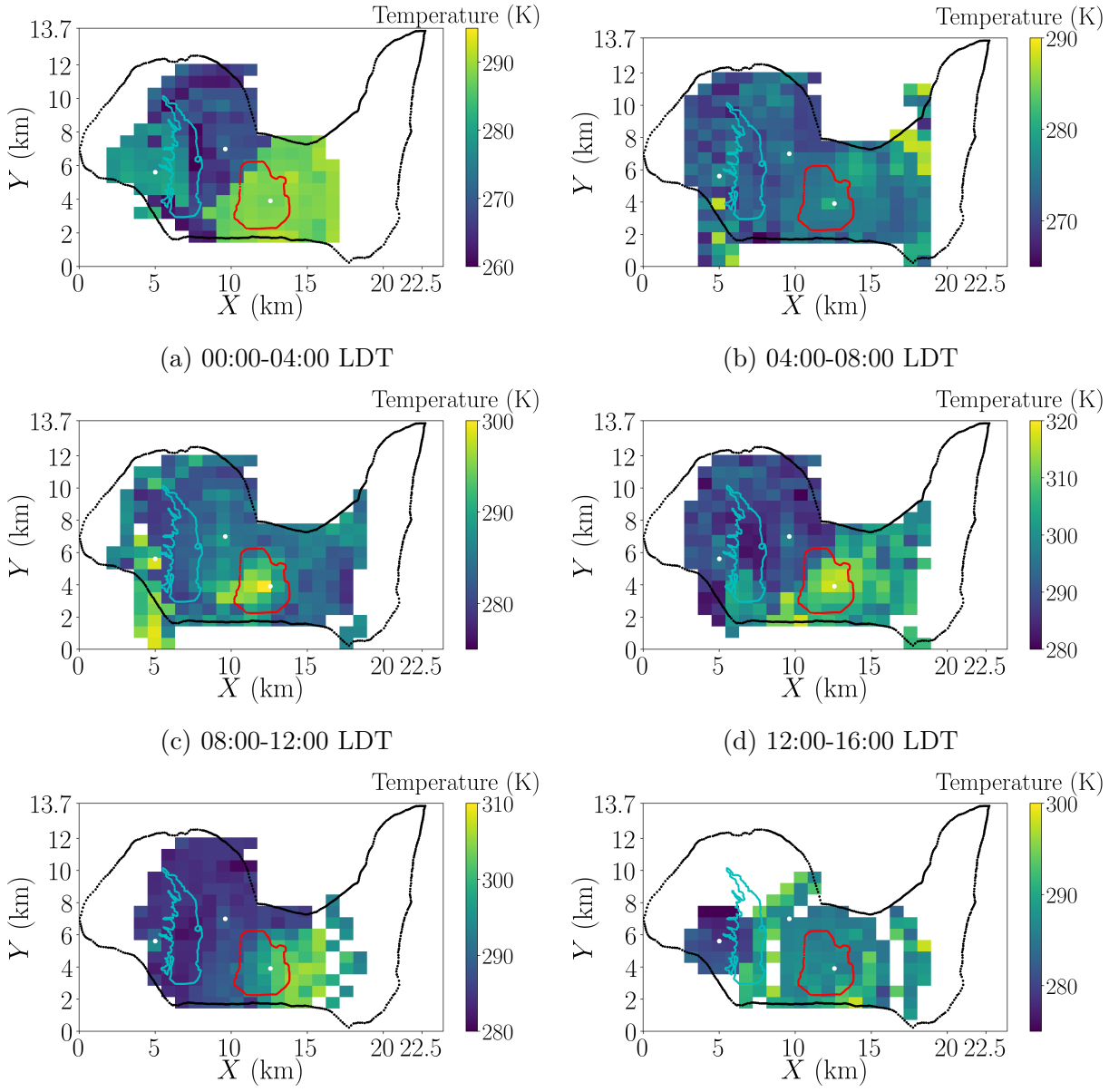
Box plots (Figure 3.2) representing the surface temperature range in Kelvin of the two key geographical features of the mining facility, the mine, and the tailings pond, at the corresponding six four-hour time intervals were created to compare diurnal ST variation. The ST values included in the box plot are located within the red and teal perimeters of the mine and tailings pond, respectively, shown in Figure 2.2. The black circles represent temperature values outside of the 95^{th} and 5^{th} percentiles. The upper black line and lower black line of the box plot correspond to the 95^{th} and 5^{th} percentiles. The middle orange line represents the median surface temperature of each geographical feature.

During the 00:00-04:00 LDT time interval, there was a distinct temperature gradient between the mine, the land west of the pond, and the pond itself. This gradient is further quantified by the corresponding box plot where the median surface temperature gradient between the two surface features was approximately 20 K.

There was a clear surface temperature gradient between the mine and the pond during the 04:00-08:00 LDT time interval. However, the magnitude of the temperature gradient between the mine and the pond was the lowest during this time period. Both the surface temperature map and the box plot display this trend as this time interval includes images captured during and after sunrise.

Over the 08:00-12:00 LDT interval, the surface temperatures of both the mine and the pond increase. Likewise, the temperature gradient between the two land surface features also grows, where the mine's surface temperature is higher than the tailings pond surface temperature.

During the 12:00-16:00 LDT interval, an apparent temperature gradient existed between



(a) 00:00-04:00 LDT

(b) 04:00-08:00 LDT

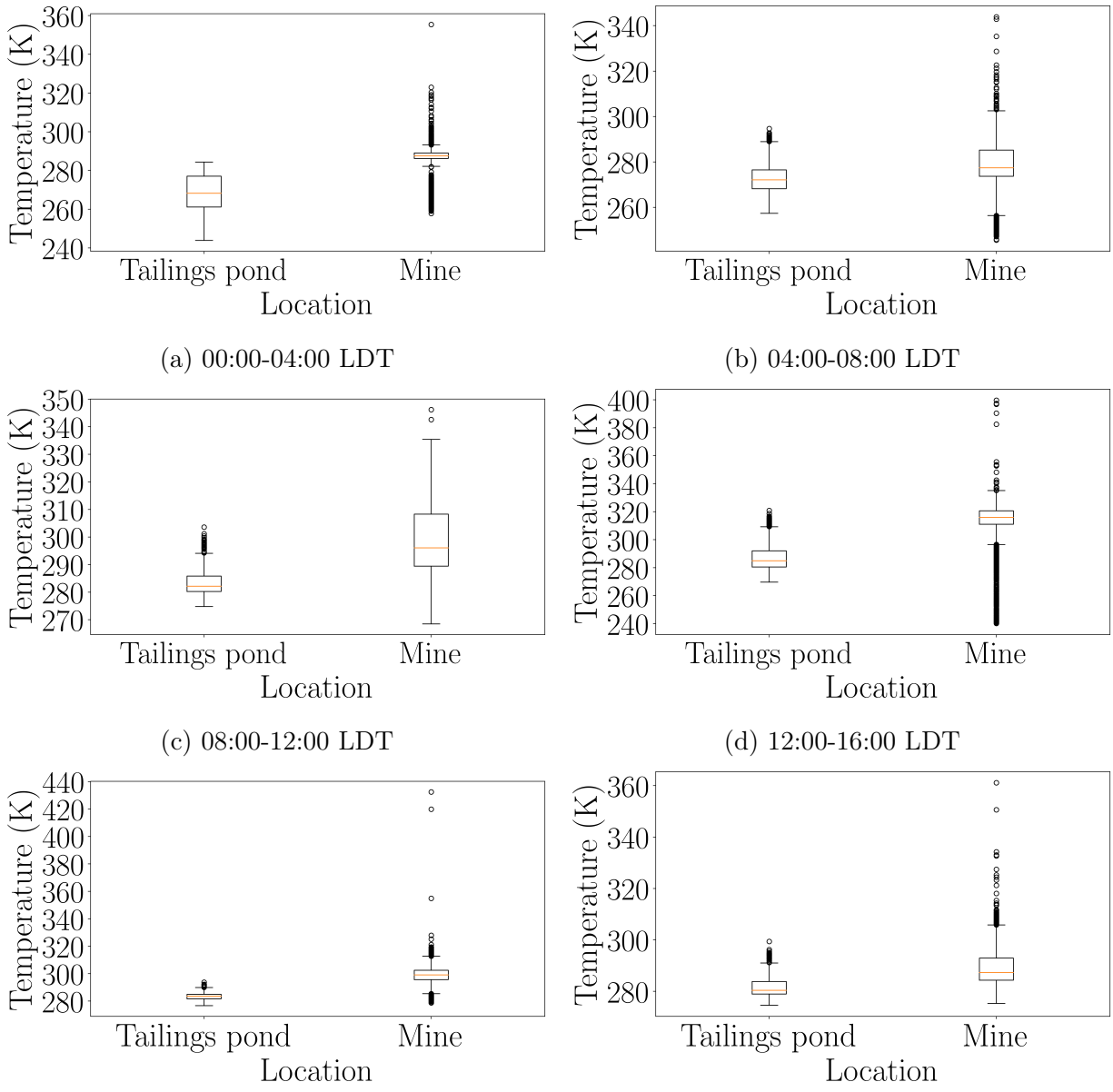
(c) 08:00-12:00 LDT

(d) 12:00-16:00 LDT

(e) 16:00-20:00 LDT

(f) 20:00-24:00 LDT

Figure 3.1: Median temperatures over four-hour time intervals at 1 km × 1 km resolution; times are in Local Daylight Time (LDT).



(a) 00:00-04:00 LDT

(b) 04:00-08:00 LDT

(c) 08:00-12:00 LDT

(d) 12:00-16:00 LDT

(e) 16:00-20:00 LDT

(f) 20:00-24:00 LDT

Figure 3.2: Box plots representing temperature distribution over four-hour time intervals for the tailings pond and mine, where the orange line is the median temperature; times are in Local Daylight Time (LDT).

the tailings pond and the mine. The area to the north-west of the mine had a lower surface temperature as compared to areas south and east of the mine.

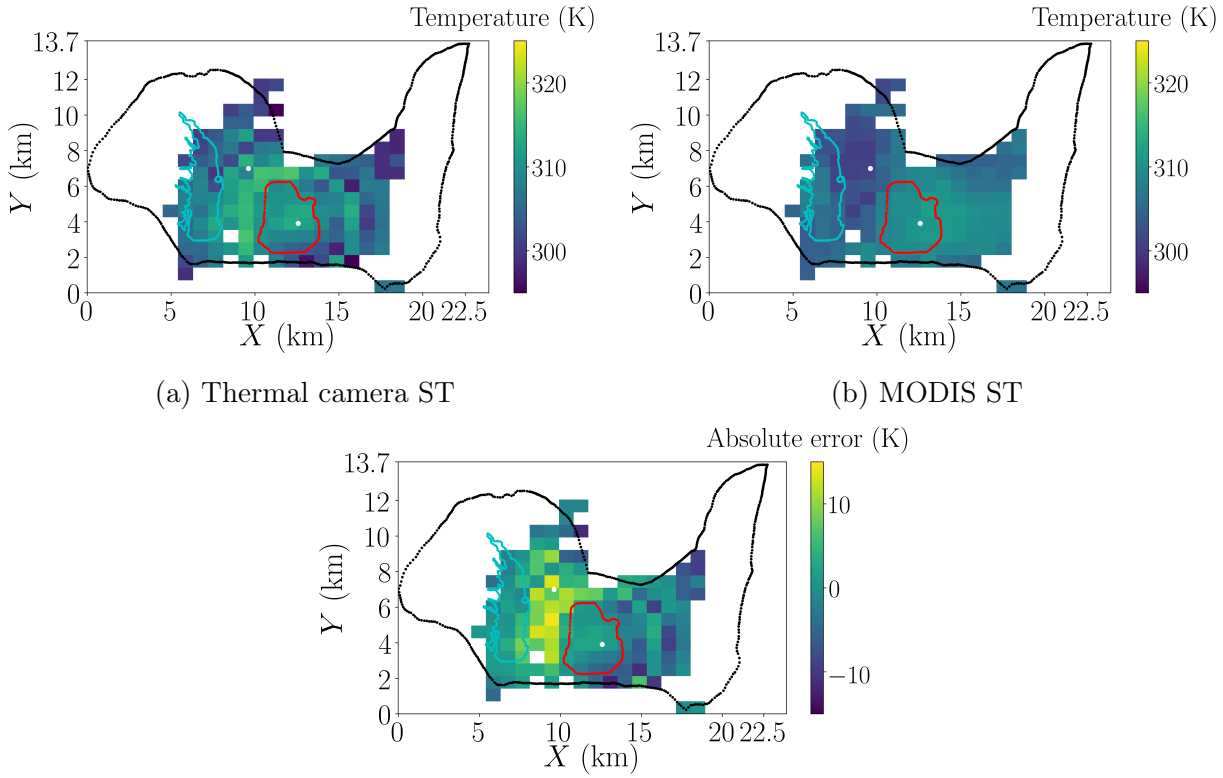
The variability of surface temperatures between the mine and the pond decreased over the 16:00-20:00 LDT interval. Although a clear temperature gradient was present, the box plot displays a narrower temperature range as compared to most other time intervals.

The same temperature gradient as discussed during other time periods occurs within the 20:00-24:00 LDT period. There are a few data gaps for ST north-west of the mine as the TANAB2 was deployed less during these hours compared to other time periods. Nonetheless, the west side of the pond possesses a lower surface temperature as compared to the mine itself. The overall surface temperature magnitude for both land surface features was determined to be decreasing during this interval, after sunset.

3.1.2 Satellite Comparison

On May 24, 2018 MODIS on the Terra satellite imaged the remote mining site during the early afternoon. The TANAB2 was launched within the mine between 12:00 and 14:00 LDT on the same date. Figure 3.3 displays the surface temperatures recorded by the thermal camera from the TANAB2, the surface temperatures recorded by MODIS from the MOD11A1 dataset, and the absolute error between the two datasets.

Absolute error with respect to MODIS temperatures on May 24, 2018 was calculated and the spatial distribution of temperature bias is displayed in Figure 3.3. The maximum, minimum, and median absolute error were calculated to be 14.3 K, -12.2 K, and 0.64 K, respectively. The bias and RMSE were determined to be 0.5 K and 5.45 K, respectively. Furthermore, it was noted that the absolute error increased north-west of the mine, towards the pond. This likely occurred as the TANAB2 was launched within the mine, below grade level (with respect to the mining facility), while the land elevation increases north-west of the mine towards the tailings pond. With this change in elevation, the calculated surface temperatures northwest of the mine are estimated from very oblique angle images, possibly contributing to the increased error. In addition, that region contains very localised hot spots, such as pipelines, that are beyond MODIS data product resolutions to be detected by the satellite but within the resolution of the thermal images in the current method. This can also explain the discrepancy between the methods. On the other hand, the elevation of the land surface decreased south and east of the mine. This decrease is likely attributed to less oblique images and therefore lower absolute error between the two datasets. Nevertheless,



(a) Thermal camera ST

(b) MODIS ST

(c) Absolute error in ST

Figure 3.3: Comparison between the developed thermal imaging method, the MODIS MOD11A1 dataset, and absolute error between the two methods; (a) median ST from May 24, 2018 12:00-14:00 LDT as recorded by the thermal camera at a 1 km × 1 km resolution; (b) daytime temperatures captured by MODIS recorded during the early afternoon on May 24, 2018 and derived from the MOD11A1 dataset at a 1 km × 1 km resolution; (c) absolute error between the two methods at a 1 km × 1 km resolution; times in Local Daylight Time (LDT).

the localised warm regions of surface temperatures within the mine and east of the mine recorded by MODIS were also captured from the thermal images as displayed by the surface temperature plots in Figure 3.3.

The increase in error between the mine and the pond can be accounted for from the rapid change in topography. In this region, the bottom of the mine pit is approximately 100 m into the earth. Conversely, the area directly to the east of the pond (the levee) is the highest location of the entire site. The total change of land surface elevation between the mine and the pond is very significant and may not be fully considered by the Digital Elevation Model (DEM) acquired from Google Earth. Wang et al. [101] evaluated the accuracy of elevation data provided by Google Earth for over 20,000 locations of the conterminous United States. They determined that the mean average error, RMSE, and bias of elevation was 10.72 m, 22.31 m, and 0.13 m, respectively. Based on Wang et al. [101], Google Earth accuracy varies significantly by location. Furthermore, since the landscape of the mining facility is changing rapidly, the use of the Google Earth elevation data likely introduces further error into the method. For more accurate results, sUAS-based Light Detection And Ranging (LiDAR) could be a feasible solution, especially in areas where high time resolution data is required or very high resolution satellite imagery capable of creating elevation models is required [3, 29, 67].

Further improvement of the imaging workflow may also reduce errors. The imaging method only considers elevation profiles for the eight cardinal directions of each TANAB2 launch site. Using a high spatial resolution DEM raster and QGIS, the elevation profile for individual images could be quantified programmatically in Python. The elevation profiles for individual pixels within the image could also be quantified using this method. However, the accuracy of this method is dependent upon the accuracy and resolution of the DEM data source. Nonetheless, the accuracy of the Google Earth elevation data was deemed to be acceptable for this application.

Using oblique and very oblique images in the method may have contributed to surface temperature error even with using the corrected camera parameters R, B, O, and F. Oblique imaging is known to affect observed surface temperatures as a function of camera pitch angle [22]. Increasingly oblique imaging angles can result in a higher proportion of reflected radiation and more varied emissivity values over waterbodies [5, 22, 90]. The proportion of waterbodies within the mining facility is low and even the tailings pond may not truly be representative of a pond due to byproducts introduced from the mine ore extraction process. It is known that imaging angles higher than 30° of nadir can affect surface temperature by

0.5 K [22, 45, 90]. For land surfaces, James et al. [40] recorded lava flows with ±3 % radiative power differences. The areas with the highest temperature error were not waterbodies. Oblique images of land surfaces likely have less impact on emissivity as opposed to images of water bodies. Nonetheless, the presence of this error source is acknowledged in this thesis. Additionally, the processing of oblique and very oblique images may have introduced georeferencing (positioning) error into the quantified pixel latitude and longitude values, especially for the upper half of each image. Pixel rows at the top of each image are increasingly further away from each other as compared to pixel rows at the bottom of an image. As a result, georeferenced pixels near the top of oblique images may not fully consider surface terrain variation, thus leading to increased positioning errors. Since the direct evaluation of the georeferencing method was not evaluated, quantitative impact of these potential errors cannot be fully determined.

Other than surface elevation variation, calculated temperature errors may have been introduced from the camera constant calibration completed in Guelph, Ontario, Canada. The surface materials at the mining site may have been different as opposed to the tested surface temperatures recorded during the calibration experiment. The difference in physical properties may have contributed to the increased minimum and maximum errors of -12.2 K and 14.3 K, respectively. However, the overall median error was calculated to be 0.64 K which is significantly below the manufacturer reported accuracy of $\pm$ 5 K and the calibrated accuracy of a FLIR Vue Pro 640 of $\pm$ 5 K [27, 46]. These elevated maximum and minimum errors may be due to highly oblique images, near horizontal, where reflected radiation can significantly impact the radiometric measurement. To avoid these errors, deploying the TANAB2 at a higher altitude would be necessary to reduce oblique imaging angles. However, this was not possible as the TANAB2 profile height was predetermined from aviation and site specific regulations.

Certain parameters (such as filtering angles or calculated distances away from the TANAB2) included in the direct georeferencing and temperature calculation code could be changed and a sensitivity analysis could be performed. It is possible to reduce errors through the completion of sensitivity analyses on all relevant parameters. The benefits and drawbacks associated with changing these parameters would need to be considered before new values are selected. For example, omitting very oblique images would reduce the area of covered by the surface temperature maps but the errors associated with reflected radiation may be minimised. The developed image processing method is very customisable depending on the desired application.

3.1.3 Principal Component Analysis (PCA)

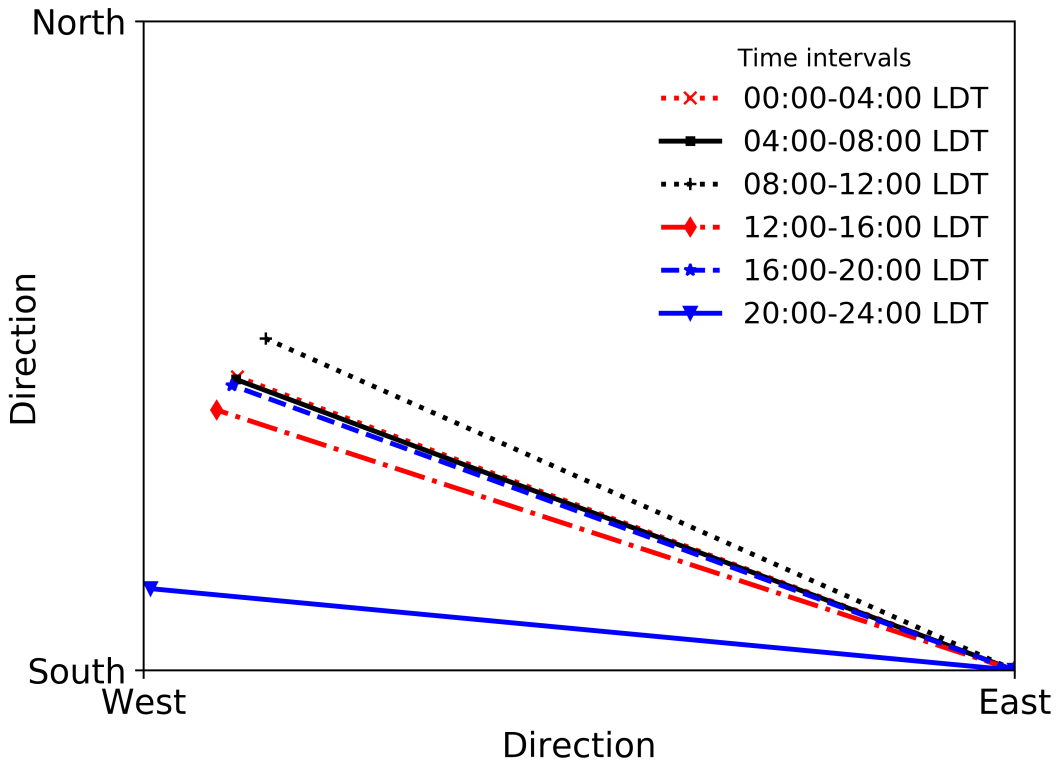


Figure 3.4: Representative horizontal directions encompassing the mining facility which display the largest surface temperature variation for each time interval; times are in Local Daylight Time (LDT).

The result of the PCA analysis is shown in Figure 3.4. As can be seen, the highest surface temperature variation is the north-west-south-east direction. Referring to Figure 2.2, the north-west-south-east direction intersects the mine and processing facilities to the south-east and the pond and forest, beyond the facility, to the north-west. This surface temperature variation was present in each time interval, especially during 00:00-04:00 LDT, 04:00-08:00 LDT, and 16:00-20:00 LDT, where the normalised PCA horizontal directions are close to overlapping each other.

3.2 Guelph Campaign

One analysis was conducted on the images collected during the University of Guelph Summer 2018 field campaign. Diurnal median ST distribution was calculated from the images collected on July 28, 2018 and August 13, 2018 over a 1 km × 1 km study area of the University of Guelph campus. Surface temperature plots were created at two spatial resolutions, 20 m × 20 m and 50 m × 50 m, respectively. In total, five ST plots were created representing Reek Walk and the land surrounding the TANAB2 launch site for 04:00-08:00 Eastern Daylight Time (EDT), 08:00-12:00 EDT, 12:00-16:00 EDT, 16:00-20:00 EDT, and 20:00-24:00 EDT. Additional landmarks were identified in Figures 3.5 and 3.6.

For each plot in Figure 3.5, differing median ST patterns exist in relation to urban surfaces and green spaces on and around the University of Guelph campus. Although Figure 3.6 represents ST at a lower spatial resolution, the median ST distribution follows a similar trend as depicted in Figure 3.5.

During the 04:00-08:00 EDT interval, the area to the bottom left of the TANAB2 launch site was cooler as compared to other regions. This area consists of green space mixed with urban surfaces. The green space could have contributed to lower surface temperatures. The area at the bottom right of the plot has higher median ST values as compared to the bottom left region. The area to the bottom right primarily consists of a built-up residential area which may contribute to the increased temperatures. The upper right region in relation to the TANAB2 launch location is The Arboretum at University of Guelph, which primarily consists of trees and green space. The top right of the plot (The Arboretum) appears to have higher median ST values as compared the green space to the bottom right of the launch. These higher median ST values may be influenced by errors attributed to highly oblique imaging angles where the ST calculation does not account for the increased fraction of reflected or transmitted radiation [33, 40].

During the 08:00-12:00 EDT interval, urban surfaces had higher median ST values as compared to green spaces. Furthermore, this time range had a greater ST range and increasingly non-uniform temperature variations as compared to the 04:00-08:00 EDT interval. These trends may be developed as the sunrise for nearby Hamilton, Ontario, Canada was 06:06 EDT on July 28, 2018 and 06:23 EDT on August 13, 2018.[1] Chudnovsky et al. [16] noted that the minimal surface temperature values occurred immediately before sunrise. As a result, with approximately only two hours of influent solar radiation, the 04:00-08:00 EDT

[1] https://www.nrc-cnrc.gc.ca/eng/services/sunrise/index.html

interval should be cooler than the 08:00-12:00 EDT plot with four hours of increasingly intense solar radiation.

Over the 12:00-16:00 EDT interval, the trend of increasing surface temperatures continue with respect to the 08:00-12:00 EDT interval. The plot represents median ST for midday and afternoon, which is commonly one of the intervals with the highest land surface temperatures. Highly urbanized surfaces, such as parking lots, especially in close proximity to the TANAB2 launch site, had the highest median surface temperatures. Furthermore, campus buildings generally had higher ST values as compared to their surroundings. This trend is especially true for the Edmund C. Bovey Building, the Albert A. Thornbrough Building, the Crop Science Building, the Fieldhouse, and the Athletic Centre. Johnston Green and the residential area at the bottom right of the launch site, had cooler temperatures as compared to the rest of the ST plot. Outside of these specific landmarks, the median surface temperature distribution is rather uniform within the 290 K to 300 K range.

The urban surfaces of the University of Guelph campus had increased surface temperatures as compared to the other areas in the 12:00-16:00 EDT interval. Specifically, areas to the right of the TANAB2 launch site had the highest ST, including the Athletic Centre, the Gryphon Centre Arena, the Crop Science Building, the Landscape Architecture Building, and Rozanski Hall. The region to the far right and bottom right of Reek Walk also had higher surface temperatures within the residential area. Urbanized areas with mixed green space had generally uniform surface temperatures within the 290 K to 300 K range. Furthermore, the far top left region of the plot had cooler surface temperatures as compared to other areas. The Cutten Fields golf course and the University of Guelph North Residences, intermixed with green spaces, are situated in the region.

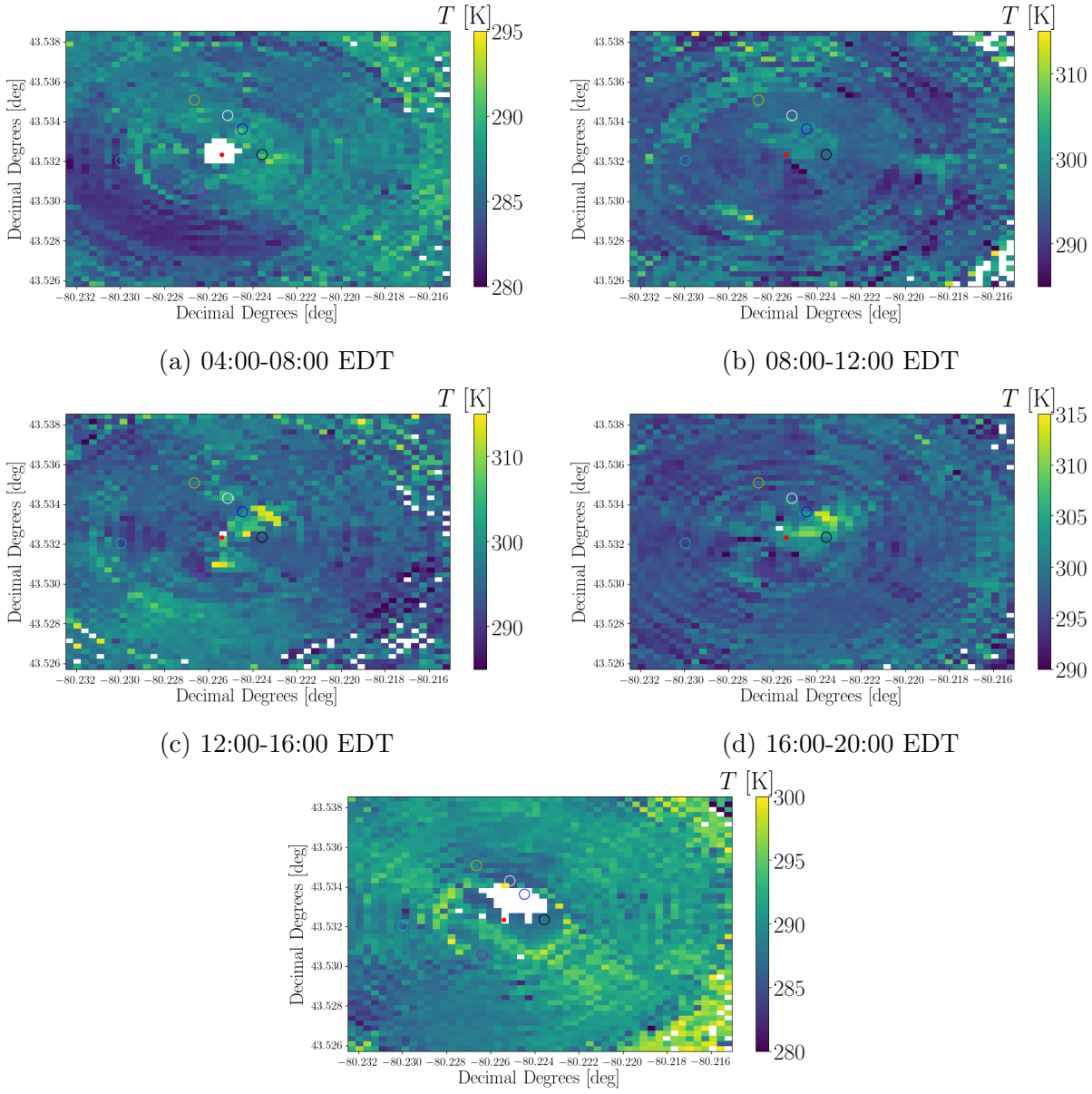
The spatial pattern of ST where green spaces have lower ST compared to urbanized areas has been reported in literature [108]. In nearby Hamilton, Ontario, Canada on July 28, 2018, the sunset occurred at 20:46 EDT and 20:26 EDT on July 28, 2018 and August 13, 2018, respectively. The 16:00-20:00 EDT interval was the last time range with continual incoming solar radiation. The time period with the highest ST values varies by location. Urban ST is known to be highly influenced by the height and width of buildings, street orientation, building and street orientation, and the sky view factor of the location [2]. The time interval with the highest surface temperatures was calculated to be the 16:00-20:00 EDT period. However Ahmed et al. [2] conducted a study which determined that building roof temperatures achieved a maximum at noon and building wall and sidewalk surface temperatures reached a maximum later in the afternoon. As a result, urban surface

temperature trends cannot be generalised.

The surface temperature values decrease between the 16:00-20:00 EDT and the 20:00-24:00 EDT intervals. Without incoming solar radiation, object surfaces have a net radiative loss and ultimately possess lower surface temperatures values. The highest surface temperatures occur at the bottom right and top right regions of the plot, in the residential area and The Arboretum respectively. Additionally, the parking lots to the right of the TANAB2 launch, Johnston Hall, and McLaughlin library had higher temperatures as opposed to other buildings and regions. The lowest ST values were located at the bottom left of the plot, adjacent to the University of Guelph Equine Sport & Reproduction Center. However, surface temperatures near the edges of this plot may be influenced by highly oblique images where the fraction of reflected and transmitted radiation is higher than the values used in the ST calculation.

Surface temperature plots were created at two spatial resolutions, 20 m $\times$ 20 m, and 50 m $\times$ 50 m. Although the 20 m spatial resolution figures provide more detail, the 50 m spatial resolution plots may be more useful for analysis. Due to the nature of the image processing method, pixels in the same vertical rows are selected for each image. In the event where more images are recorded during a particular TANAB2 profile, surface temperatures from the same region would be calculated. This process would result in concentric-like circles of temperature gradients. In Figure 3.5, concentric-like surface temperature patterns appear for all time periods. Conversely, the 50 m spatial resolution temperature plots do not have any noticeable concentric temperature patterns. With a lower spatial resolution, the quantified median temperature for each square is large enough to omit spatial temperature distribution effects imposed from the image processing workflow.

The code used to separate the data into the five four-hour time intervals is included in Section A.3.5 and the code used to calculate surface temperatures as a function of land material type and to plot the surface temperature maps is included in Section A.3.6.



(a) 04:00-08:00 EDT

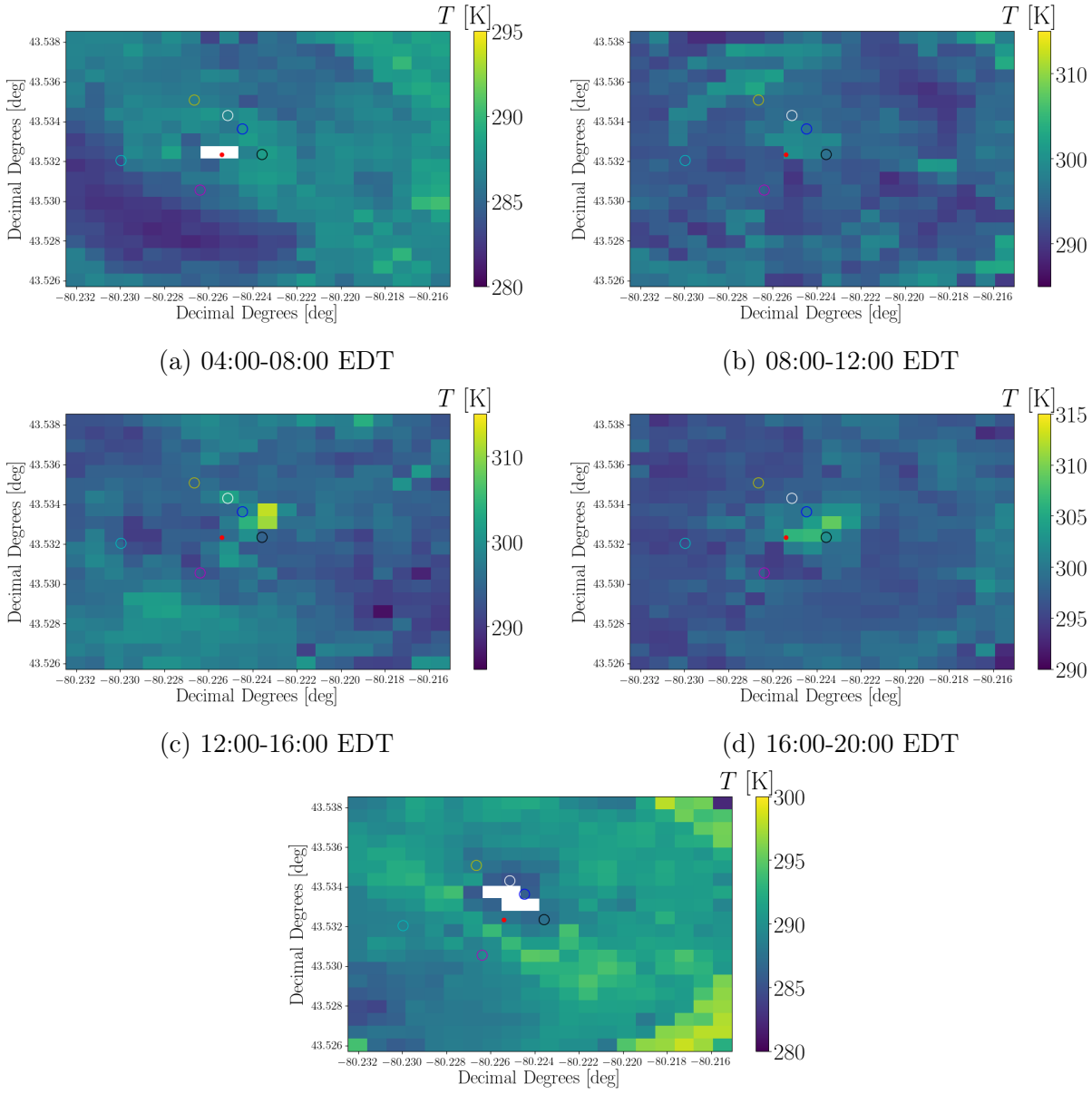
(b) 08:00-12:00 EDT

(c) 12:00-16:00 EDT

(d) 16:00-20:00 EDT

(e) 20:00-24:00 EDT

Figure 3.5: Median surface temperatures over four-hour time intervals at 20 m × 20 m spatial resolution, where the red dot represents the TANAB2 launch location (Reek Walk), the black circle represents the Gryphon Centre Arena, the magenta circle represents the University Centre, the blue circle represents the Athletic Centre, the yellow circle represents Varsity Field, the cyan circle represents Johnston Green, and the white circle represents the Fieldhouse.



(a) 04:00-08:00 EDT

(b) 08:00-12:00 EDT

(c) 12:00-16:00 EDT

(d) 16:00-20:00 EDT

(e) 20:00-24:00 EDT

Figure 3.6: Median surface temperatures over four-hour time intervals at 50 m × 50 m spatial resolution, where the red dot represents the TANAB2 launch location (Reek Walk), the black circle represents the Gryphon Centre Arena, the magenta circle represents the University Centre, the blue circle represents the Athletic Centre, the yellow circle represents Varsity Field, the cyan circle represents Johnston Green, and the white circle represents the Fieldhouse.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

A novel, small Unmanned Aerial System (sUAS)-based and open-source thermal image processing approach was developed to directly georeference and calculate Earth surface temperatures with a high spatiotemporal resolution. An uncooled thermal camera was launched on a tethered balloon during May 2018 at a remote northern Canadian mining facility and at the University of Guelph, Guelph, Ontario, Canada on July 28, 2018 and August 13, 2018. Based on the topography of the surrounding land, the camera's Global Positioning System (GPS) location, the balloon altitude, and the camera's Pitch, Yaw, and Roll angles, individual pixels within each image were directly georeferenced by assigning a calculated longitude and latitude to each respective pixel. The derived imaging workflow was developed for images recorded with oblique angles relative to the land surface.

4.1.1 Georeferencing

The accuracy of the direct georeferencing method was indirectly evaluated through the comparison with an image from the MODerate resolution Imaging Spectroradiometer (MODIS). However a similar direct georeferencing system using a Global Navigation Satellite System (GNSS) module reported horizontal positioning errors of 1.06 m [70]. A direct comparison of planimetric geographical pixel positioning could be completed with the images obtained from the Guelph field campaign in relation to GPS coordinates of recognizable building footprints from satellite images in a Geographic Information System (GIS) application. The Tethered And Navigated Air Blimp 2 (TANAB2) vertical altitude accuracy however was quantified

to be 1.2 m as calculated with the uncertainty of error equation. Aside from potential positioning errors, the direct georeferencing workflow is economically and conceptually efficient. When recording images, Ground Control Points (GCPs) are not required as the geographical positioning of the camera is used in conjunction with camera field of view, Pitch, Roll, and Yaw parameters to calculate planimetric pixel position. The omission of GCPs from field campaigns allows airborne imaging platforms to be deployed in environments where GCPs cannot be placed such as industrial facilities, large waterbodies, and urban centres where it is generally assumed to be unsafe, logistically challenging, or legally restrictive to deploy GCPs.

4.1.2 Thermal Imaging

A radiometric calibration was completed for the DJI Zenmuse XT 19-mm camera for grass, water, soil, and developed land surfaces in an outdoors field experiment at the University of Guelph, Guelph, Ontario, Canada. Using a non-linear fitting library in Python, the camera constants (B, R, O, and F) used in the surface temperature calculation of individual thermal image pixels were optimised as a function of land surface material. These constants were used to quantify surface temperatures from images collected from both the mining facility and the University of Guelph field campaigns.

The calculated land surface temperatures from images recorded during the mining campaign accurately represented the diurnal variation of surface temperature with a high degree of spatiotemporal accuracy as compared to conventional remote sensing techniques including satellites. A comparison between a MODIS satellite image and the results from the imaging workflow yielded a bias of 0.5 K, a Root Mean Square Error (RMSE) of 5.45 K, and a median absolute error of 0.64 K of surface temperatures surrounding the mining facility. A Principal Component Analysis (PCA) was conducted for each four-hour time interval and the direction with the highest surface temperature variation was determined to be north-west-south-east. The PCA agrees well with the diurnal surface temperature maps and the MODIS image.

The quantified land surface temperatures from images recorded during the Guelph campaign displayed temperature variations diurnally. Two sets of surface temperature plots at 20 m $\times$ 20 m and 50 m $\times$ 50 m spatial resolution were created for five four-hour time periods. Based on the image processing workflow, the 50 m spatial resolution figures may be more appropriate than the 20 m resolution plots as circles of concentric surface temperature variations are easily identifiable in plots with the higher spatial resolution. This issue could be

corrected by changing the image processing workflow or by recording more images for each four-hour interval at a variety of altitudes, camera Pitch, and Yaw angles respectively.

The developed direct georeferencing thermal imaging method is able to quantify surface temperatures at a high spatiotemporal resolution as compared to satellite-based remote sensing alternatives. The use of an airborne platform enables operators to measure Earth surface temperature over any time interval. Furthermore, the low level flight of the airborne vector significantly increases the spatial resolution of the imaged Earth surface. Physical capabilities of the thermal imaging camera limit the accuracy of the measured Earth surface temperature such that calculated absolute temperature measurements may not be suitable for applications where a high degree of accuracy is required. However, the calculated Earth surface temperatures are suitable for representing relative surface temperatures. Nonetheless, the derivation of the direct georeferencing equations are applicable to a wide range of remote sensing cameras and will continue to be relevant as thermal imaging technology advances.

4.2 Future Work

Proper, direct validation of the direct georeferencing method should be quantified, especially if very high spatial resolution temperature data (below 10 m) is desired. Quantitative positional accuracy is also required if the workflow were to be used to process images and calculate temperatures comparable to a legal standard. Similarly, methods to reduce the RMSE of the surface temperature measurement should be studied and implemented to increase surface temperature accuracy. Furthermore, the use of a very high resolution (less than 10 m) Digital Surface Model (DSM) or Digital Elevation Model (DEM) to quantify terrain elevation above sea level for any camera Yaw degree would reduce georeferencing errors in environments with highly variable surfaces (for example urban areas or mountainous regions). The use of Unmanned Aerial System (UAS)-based Light Detection And Ranging (LiDAR) could be equipped to the TANAB2 to develop very high resolution DSM files for thermally imaged locations [38, 98, 99]. To increase versatility of the imaging method, it is recommended that images are acquired through the use of an airborne vehicle, such as a drone. Deployment of the TANAB2 requires multiple personnel to control mooring ropes that stabilise the balloon in strong wind conditions.

The developed imaging workflow could be applied to other Earth surfaces such as waterbodies. Evaluation and mapping of thermal plume distribution in waterbodies, such as lakes and rivers, especially in urban areas has been published in literature [13, 20, 53]. The

direct georeferencing method could be applied to other imaging systems such as hyperspectral, multispectral, and Red Green Blue (RGB)-colour cameras. Hyperspectral cameras have been used in remote sensing for a wide variety of applications including but not limited to precision agriculture and for geological mapping of minerals [10, 39, 87]. Multispectral cameras have been used heavily for precision agriculture and forestry applications [7, 14, 71]. RGB cameras have been used, along with multispectral cameras, to identify algal blooms in waterbodies [103, 105]. The developed direct georeferencing method functions independently of camera type. As a result, the detailed mathematical equations in this thesis are suitable for numerous applications.

Bibliography

[1] Aber, J. S. Lighter-than-air platforms for small-format aerial photography. *Transactions of the Kansas Academy of Science 107*, 1 (2004), 39–44.

[2] Ahmed, A. Q., Ossen, D. R., Jamei, E., Manaf, N. A., Said, I., and Ahmad, M. H. Urban surface temperature behaviour and heat island effect in a tropical planned city. *Theor. Appl. Climatol. 119*, 3-4 (2015), 493–514.

[3] Akturk, E., and Altunel, A. O. Accuracy assessment of a low-cost UAV derived digital elevation model (DEM) in a highly broken and vegetated terrain. *Measurement 136* (2019), 382–386.

[4] Bah, M. K., Gunshor, M. M., and Schmit, T. J. Generation of GOES-16 true color imagery without a green band. *Earth Space Sci. 5*, 9 (2018), 549–558.

[5] Baker, E. A., Lautz, L. K., McKenzie, J. M., and Aubry-Wake, C. Improving the accuracy of time-lapse thermal infrared imaging for hydrologic applications. *J. Hydrol. 571* (2019), 60–70.

[6] Bakuła, K., Salach, A., Wziątek, D. Z., Ostrowski, W., Górski, K., and Kurczyński, Z. Evaluation of the accuracy of lidar data acquired using a UAS for levee monitoring: preliminary results. *Int. J. Remote Sens. 38*, 8-10 (2017), 2921–2937.

[7] Berni, J. A., Zarco-Tejada, P. J., Suárez, L., and Fereres, E. Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle. *IEEE T. Geosci. Remote 47*, 3 (2009), 722–738.

[8] Bolanakis, D. E., Kotsis, K. T., and Laopoulos, T. Temperature influence on differential barometric altitude measurements. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)* (Warsaw, Poland, Sep. 2015), vol. 1, pp. 120–124.

[9] Boon, M. A., Drijfhout, A. P., and Tesfamichael, S. Comparison of a fixed-wing and multi-rotor UAV for environmental mapping applications: a case study. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-2/W6* (2017), 47–54.

[10] Boubanga-Tombet, S., Huot, A., Vitins, I., Heuberger, S., Veuve, C., Eisele, A., Hewson, R., Guyot, E., Marcotte, F., and Chamberland, M. Thermal infrared hyperspectral imaging for mineralogy mapping of a mine face. *Remote Sens-Basel 10*, 10 (2018), 1518.

[11] Brenner, C., Zeeman, M., Bernhardt, M., and Schulz, K. Estimation of evapotranspiration of temperate grassland based on high-resolution thermal and visible range imagery from unmanned aerial systems. *Int. J. Remote Sens. 39*, 15-16 (2018), 5141–5174.

[12] Budzier, H., and Gerlach, G. Calibration of uncooled thermal infrared cameras. *J. Sens. Sens. Syst. 4*, 1 (2015), 187–197.

[13] Caldwell, S. H., Kelleher, C., Baker, E. A., and Lautz, L. K. Relative information from thermal infrared imagery via unoccupied aerial vehicle informs simulations and spatially-distributed assessments of stream temperature. *Sci. Total Environ. 661* (2019), 364–374.

[14] Candiago, S., Remondino, F., De Giglio, M., Dubbini, M., and Gattelli, M. Evaluating multispectral images and vegetation indices for precision farming applications from UAV images. *Remote Sens-Basel 7*, 4 (2015), 4026–4047.

[15] Chastain, R., Housman, I., Goldstein, J., Finco, M., and Tenneson, K. Empirical cross sensor comparison of Sentinel-2A and 2B MSI, Landsat-8 OLI, and Landsat-7 ETM+ top of atmosphere spectral characteristics over the conterminous United States. *Remote Sens. Environ. 221* (2019), 274–285.

[16] Chudnovsky, A., Ben-Dor, E., and Saaroni, H. Diurnal thermal behavior of selected urban objects using remote sensing measurements. *Energ. Buildings 36*, 11 (2004), 1063–1074.

[17] Cintineo, R. M., Otkin, J. A., Jones, T. A., Koch, S., and Stensrud, D. J. Assimilation of synthetic GOES-R ABI infrared brightness temperatures and WSR-88D radar observations in a high-resolution OSSE. *Mon. Weather Rev. 144*, 9 (2016), 3159–3180.

[18] Colomina, I., and Molina, P. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS J. Photogramm. 92* (2014), 79–97.

[19] Crosson, W. L., Al-Hamdan, M. Z., Hemmings, S. N. J., and Wade, G. M. A daily merged MODIS Aqua-Terra land surface temperature data set for the conterminous United States. *Remote Sens. Environ. 119*, 8 (2012), 315–324.

[20] DeMario, A., Lopez, P., Plewka, E., Wix, R., Xia, H., Zamora, E., Gessler, D., and Yalin, A. P. Water plume temperature measurements by an Unmanned Aerial System (UAS). *Sensors-Basel 17*, 2 (2017), 306.

[21] Duffy, J. P., and Anderson, K. A 21st-century renaissance of kites as platforms for proximal sensing. *Prog. Phys. Geog. 40*, 2 (2016), 352–361.

[22] Dugdale, S. J. A practitioner's guide to thermal infrared remote sensing of rivers and streams: recent advances, precautions and considerations. *WIREs Water 3* (2016), 251–268.

[23] Estoque, R. C., and Murayama, Y. Classification and change detection of built-up lands from landsat-7 etm+ and landsat-8 oli/tirs imageries: a comparative assessment of various spectral indices. *Ecol. Indic. 56* (2015), 205–217.

[24] Eynard, D., Vasseur, P., Demonceaux, C., and Frémont, V. Real time UAV altitude, attitude and motion estimation from hybrid stereovison. *Auton. Robot. 33*, 1-2 (2012), 157–172.

[25] Fang, L., Zhan, X., Hain, C. R., Yin, J., Liu, J., and Schull, M. A. An assessment of the impact of land thermal infrared observation on regional weather forecasts using two different data assimilation approaches. *Remote Sens-Basel 10*, 4 (2018), 625.

[26] FLIR-Systems. *The ultimate infrared handbook for R&D professionals*. FLIR Systems, 2012.

[27] Gallardo-Saavedra, S., Hernández-Callejo, L., and Duque-Perez, O. Technological review of the instrumentation used in aerial thermographic inspection of photovoltaic plants. *Renew. Sust. Energ. Rev. 93* (2018), 566–579.

[28] Gémes, O., Tobak, Z., and van Leeuwen, B. Satellite based analysis of surface urban heat island intensity. *J. Environ. Geogr. 9*, 1-2 (2016), 23–30.

[29] Gray, P. C., Ridge, J. T., Poulin, S. K., Seymour, A. C., Schwantes, A. M., Swenson, J. J., and Johnston, D. W. Integrating drone imagery into high resolution satellite remote sensing assessments of estuarine environments. *Remote Sens-Basel 10*, 8 (2018), 1257.

[30] Hais, M., and Kučera, T. The influence of topography on the forest surface temperature retrieved from landsat TM, ETM+ and ASTER thermal channels. *ISPRS J. Photogramm. 64*, 6 (2009), 585–591.

[31] Hardin, P. J., Lulla, V., Jensen, R. R., and Jensen, J. R. Small unmanned aerial systems (sUAS) for environmental remote sensing: challenges and opportunities revisited. *Gisci. Remote Sens. 56*, 2 (2019), 309–322.

[32] Hirano, A., Welch, R., and Lang, H. Mapping from ASTER stereo image data: DEM validation and accuracy assessment. *ISPRS J. Photogramm. 57*, 5-6 (2003), 356–370.

[33] Hopskinson, C., Barlow, J., Demuth, M., and Pomeroy, J. Mapping changing temperature patterns over a glacial moraine using oblique thermal imagery and lidar. *Can. J. Remote Sensing. 36*, Suppl. 2 (2010), S257–S265.

[34] Horrocks, L. A., Candy, B., Nightingale, T. J., Saunders, R. W., O'Carroll, A., and Harris, A. R. Parameterizations of the ocean skin effect and implications for satellite-based measurement of sea-surface temperature. *J. Geophys. Res-Oceans 108*, C3 (2003), 3096.

[35] Horton, T. W., Oline, A., Hauser, N., Khan, T. M., Laute, A., Stoller, A., Tison, K., and Zawar-Reza, P. Thermal imaging and biometrical thermography of humpback whales. *Front. Mar. Sci. 4* (2017), 424.

[36] Inamdar, A. K., French, A., Hook, S., Vaughan, G., and Luckett, W. Land surface temperature retrieval at high spatial and temporal resolutions over the southwestern United States. *J. Geophys. Res-Atmos. 113*, D7 (2008), 1–18.

[37] Irons, J. R., Dwyer, J. L., and Barsi, J. A. The next landsat satellite: the landsat data continuity mission. *Remote Sens. Environ. 122* (2012), 11–21.

[38] Jaakkola, A., Hyyppä, J., Kukko, A., Yu, X., Kaartinen, H., Lehtomäki, M., and Lin, Y. A low-cost multi-sensoral mobile mapping system and its feasibility for tree measurements. *ISPRS. J. Photogramm. 65*, 6 (2010), 514–522.

[39] Jakob, S., Zimmermann, R., and Gloaguen, R. The need for accurate geometric and radiometric corrections of drone-borne hyperspectral data for mineral exploration: MEPHySTo-a toolbox for pre-processing drone-borne hyperspectral data. *Remote Sens-Basel 9*, 1 (2017), 88.

[40] James, M. R., Robson, S., Pinkerton, H., and Ball, M. Oblique photogrammetry with visible and thermal images of active lava flows. *B. Volcanol. 69*, 1 (2006), 105–108.

[41] Ji, L., and Brown, J. F. Effect of NOAA satellite orbital drift on AVHRR-derived phenological metrics. *Int. J. Appl. Earth Obs. 62* (2017), 215–223.

[42] Jolliffe, I. *Principal component analysis, second edition.* Springer-Verlag, New York, 2002.

[43] Kalegaev, V. V., and Vlasova, N. A. Some peculiarities of longitudinal distribution of proton fluxes at high latitudes. *Adv. Space Res. 48*, 12 (2011), 2028–2035.

[44] Kawai, Y., and Wada, A. Diurnal sea surface temperature variation and its impact on the atmosphere and ocean: a review. *J. Oceanogr. 63*, 5 (2007), 721–744.

[45] Kay, J. E., Kampf, S. K., Handcock, R. N., Cherkauer, K. A., Gillespie, A. R., and Burges, S. J. Accuracy of lake and stream temperatures estimated from thermal infrared images. *J. Am. Water Resour. As. 41*, 5 (2005), 1161–1175.

[46] Kelly, J., Kljun, N., Olsson, P.-O., Mihai, L., Liljeblad, B., Weslien, P., Klemedtsson, L., and Eklundh, L. Challenges and best practices for deriving temperature data from an uncalibrated UAV thermal infrared camera. *Remote Sens-Basel 11*, 5 (2019), 567.

[47] Klemas, V. Remote sensing of coastal plumes and ocean fronts: overview and case study. *J. Coastal Res. 28*, 1 (2012), 1–7.

[48] Klemas, V. V. Coastal and environmental remote sensing from unmanned aerial vehicles: An overview. *J. Coastal Res. 31*, 5 (2015), 1260–1267.

[49] Kolodochka, A. A. Schemes of profile models of heat and mass transfer in large lakes. *Water Resour. 30*, 1 (2003), 34–41.

[50] Ku, H. H. Notes on the use of propagation of error formulas. *J. Res. Nat. Bur. Stand. Sec. C: Eng. Inst. 70*, 4 (1966), 263.

[51] Kumar, A. Long term (2003-2012) spatio-temporal MODIS (Terra/Aqua level 3) derived climatic variations of aerosol optical depth and cloud properties over a semi arid urban tropical region of Northern India. *Atmos. Environ. 83* (2014), 291–300.

[52] Lázaro, J. R. G., Ruiz, J. A. M., and Arbeló, M. Effect of spatial resolution on the accuracy of satellite-based fire scar detection in the northwest of the Iberian Peninsula. *Int. J. Remote Sens. 34* (2013), 4736–4753.

[53] Lee, E., Yoon, H., Hyun, S. P., Burnett, W. C., Koh, D.-C., Ha, K., Kim, D.-j., Kim, Y., and Kang, K.-m. Unmanned aerial vehicles (UAVs)-based thermal infrared (TIR) mapping, a novel approach to assess groundwater discharge into the coastal zone. *Limnol. Oceanogr.: Methods 14*, 11 (2016), 725–735.

[54] Li, Z.-L., Tang, B.-H., Wu, H., Ren, H., Yan, G., Wan, Z., Trigo, I. F., and Sobrino, J. A. Satellite-derived land surface temperature: Current status and perspectives. *Remote Sens. Environ. 131* (2013), 14–37.

[55] Lin, D., Maas, H.-G., Westfeld, P., Budzier, H., and Gerlach, G. An advanced radiometric calibration approach for uncooled thermal cameras. *Photogramm. Rec. 33*, 161 (2018), 30–48.

[56] Liu, L., Li, C., Lei, Y., Yin, J., and Zhao, J. Volcanic ash cloud detection from MODIS image based on CPIWS method. *Acta Geophys. 65*, 1 (2017), 151–163.

[57] Malamiri, H. R. G., Rousta, I., Olafsson, H., Zare, H., and Zhang, H. Gap-filling of MODIS time series Land Surface Temperature (LST) products using Singular Spectrum Analysis (SSA). *Atmosphere-Basel 9*, 9 (2018), 334.

[58] Malbéteau, Y., Parkes, S., Aragon, B., Rosas, J., and McCabe, M. F. Capturing the diurnal cycle of land surface temperature using an unmanned aerial vehicle. *Remote Sens-Basel 10*, 9 (2018), 1407.

[59] Mathew, A., Khandelwal, S., and Kaul, N. Analysis of diurnal surface temperature variations for the assessment of surface urban heat island effect over indian cities. *Energ. Buildings 159* (2018), 271–295.

[60] Mildrexler, D. J., Zhao, M., and Running, S. W. Satellite finds highest land skin temperatures on Earth. *B. Am. Meteorol. Soc. 92*, 7 (2011), 855–860.

[61] Moen, R., Pastor, J., and Cohen, Y. Accuracy of GPS telemetry collar locations with differential correction. *J. Wildlife Manage. 61*, 2 (1997), 530–539.

[62] Mohamed, A. A., Odindi, J., and Mutanga, O. Land surface temperature and emissivity estimation for urban heat island assessment using medium- and low-resolution space-borne sensors: a review. *Geocarto Int. 32*, 4 (2017), 455–470.

[63] Moradi, M., Dyer, B., Nazem, A., Nambiar, M. K., Nahian, M. R., Bueno, B., Mackey, C., Vasanthakumar, S., Nazarian, N., Krayenhoff, E. S., Norford, L. K., and Aliabadi, A. A. The Vertical City Weather Generator (VCWG v1.0.0). *Geosci. Model Dev. Discuss.* (2019).

[64] Moukomla, S., and Blanken, P. D. Remote sensing of the North American Laurentian Great Lakes' surface temperature. *Remote Sens-Basel 8*, 4 (2016), 286.

[65] Nambiar, M. K., Byerlay, R., Nazem, A., Nahian, M. R., Moradi, M., and Aliabadi, A. A. A Tethered and Navigated Air Blimp (TANAB) for observing the microclimate over a complex terrain. *Geosci. Instrum. Method. Data Syst. Discuss.* (2019).

[66] Nefesliogl u, H. A., San, B. T., Gokceoglu, C., and Duman, T. Y. An assessment on the use of Terra ASTER L3A data in landslide susceptibility mapping. *Int. J. Appl. Earth Obs. 14*, 1 (2012), 40–60.

[67] Nemmaoui, A., Aguilar, F. J., Aguilar, M. A., and Qin, R. DSM and DTM generation from VHR satellite stereo imagery over plastic covered greenhouse areas. *Comput. Electron. Agr. 164* (2019), 104903.

[68] Neteler, M. Estimating daily land surface temperatures in mountainous environments by reconstructed MODIS LST data. *Remote Sens-Basel 2*, 1 (2010), 333–351.

[69] Olbrycht, R., and Więcek, B. New approach to thermal drift correction in microbolometer thermal cameras. *Quant. Infr. Therm. J. 12*, 2 (2015), 184–195.

[70] Padró, J.-C., Muñoz, F.-J., Planas, J., and Pons, X. Comparison of four UAV georeferencing methods for environmental monitoring purposes focusing on the combined use with airborne and satellite remote sensing platforms. *Int. J. Appl. Earth Obs. 75* (2019), 130–140.

[71] Pajares, G. Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs). *Photogramm. Eng. Rem. S. 81*, 4 (2015), 281–330.

[72] Patel, N. R. Investigating relations between satellite derived land surface parameters and meteorological variables. *Geocarto Int. 21*, 3 (2006), 47–53.

[73] Puliti, S., Ørka, H. O., Gobakken, T., and Næsset, E. Inventory of small forest areas using an unmanned aerial system. *Remote Sens-Basel 7*, 8 (2015), 9632–9654.

[74] Rahaghi, A. I., Lemmin, U., Sage, D., and Barry, D. A. Achieving high-resolution thermal imagery in low-contrast lake surface waters by aerial remote sensing and image registration. *Remote Sens. Environ. 221* (2019), 773–783.

[75] Rahaman, K. R., Hassan, Q. K., and Chowdhury, E. H. Quantification of local warming trend: a remote sensing-based approach. *PLOS ONE 13*, 5 (2018), e0196882.

[76] Rankin, A. M., and Wolff, E. W. Aerosol profiling using a tethered balloon in coastal Antarctica. *J. Atmos. Ocean. Tech. 19*, 12 (2002), 1978–1985.

[77] Reichle, R. H., Kumar, S. V., Mahanama, S. P. P., Koster, R. D., and Liu, Q. Assimilation of satellite-derived skin temperature observations into land surface models. *J. Hydrometeorol. 11* (2010), 1103–1122.

[78] Reintsma, K. M., McGowan, P. C., Callahan, C., Collier, T., Gray, D., Sullivan, J. D., and Prosser, D. J. Preliminary evaluation of behavioral response of nesting waterbirds to small unmanned aircraft flight. *Waterbirds 41*, 3 (2018), 326–331.

[79] Ren, L., Castillo-Effen, M., Yu, H., Johnson, E., Yoon, Y., Takuma, N., and Ippolito, C. A. Small unmanned aircraft system (sUAS) categorization framework for low altitude traffic services. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)* (St. Petersburg, FL, USA, September 2017).

[80] Ribeiro-Gomes, K., Hernández-López, D., Ortega, J. F., Ballesteros, R., Poblete, T., and Moreno, M. A. Uncooled thermal camera calibration and optimization of the photogrammetry process for UAV applications in agriculture. *Sensors-Basel 17*, 10 (2017), 2173.

[81] Schmidt, J., Evans, I. S., and Brinkmann, J. Comparison of polynomial models for land surface curvature calculation. *Int. J. Geogr. Inf. Sci. 17*, 8 (2003), 797–814.

[82] Schmit, T. J., Griffith, P., Gunshor, M. M., Daniels, J. M., Goodman, S. J., and Lebair, W. J. A closer look at the ABI on the GOES-R series. *B. Am. Meteorol. Soc. 98*, 4 (2017), 681–698.

[83] Schmit, T. J., Gunshor, M. M., Menzel, W. P., Gurka, J. J., Li, J., and Bachmeier, A. S. Introducing the next-generation advanced baseline imager on GOES-R. *B. Am. Meteorol. Soc. 86*, 8 (2005), 1079–1096.

[84] Sheng, H., Chao, H., Coopmans, C., Han, J., McKee, M., and Chen, Y. Low-cost UAV-based thermal infrared remote sensing: platform, calibration and applications. In *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications* (Qingdao, China, 2010), pp. 38–43.

[85] Smith, G. M., and Milton, E. J. The use of the empirical line method to calibrate remotely sensed data to reflectance. *Int. J. Remote Sens. 20*, 13 (1999), 2653–2662.

[86] Stöcker, C., Nex, F., Koeva, M., and Gerke, M. Quality assessment of combined IMU/GNSS data for direct georeferencing in the context of UAV-based mapping. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-2/W6* (2017), 355–361.

[87] Stuart, M. B., McGonigle, A. J. S., and Willmott, J. R. Hyperspectral imaging in environmental monitoring: a review of recent developments and technological advances in compact field deployable systems. *Sensors-Basel 19*, 14 (2019), 3071.

[88] Stull, R. B. *Practical Meteorology: An Algebra-based Survey of Atmospheric Science*. Univ. of British Columbia, 2015.

[89] Tomlinson, C. J., Chapman, L., Thornes, J. E., and Baker, C. Remote sensing land surface temperature for meteorology and climatology: a review. *Meteorol. Appl. 18*, 3 (2011), 296–306.

[90] Torgersen, C. E., Faux, R. N., McIntosh, B. A., Poage, N. J., and Norton, D. J. Airborne thermal remote sensing for water temperature assessment in rivers and streams. *Remote Sens. Environ. 76*, 3 (2001), 386–398.

[91] Torres-Rua, A. Vicarious calibration of sUAS microbolometer temperature imagery for estimation of radiometric land surface temperature. *Sensors-Basel 17*, 7 (2017), 1499.

[92] Turner, D., Lucieer, A., and Wallace, L. Direct georeferencing of ultrahigh-resolution UAV imagery. *IEEE T. Geosci. Remote 52*, 5 (2014), 2738–2745.

[93] Usamentiaga, R., Venegas, P., Guerediaga, J., Vega, L., Molleda, J., and Bulnes, F. G. Infrared thermography for temperature measurement and non-destructive testing. *Sensors-Basel 14*, 7 (2014), 12305–12348.

[94] van der Meer, F. Near-infrared laboratory spectroscopy of mineral chemistry: a review. *Int. J. Appl. Earth Obs. 65* (2018), 71–78.

[95] Verykokou, S., and Ioannidis, C. Oblique aerial images: a review focusing on georeferencing procedures. *Int. J. Remote Sens. 39*, 11 (2018), 3452–3496.

[96] Vierling, L. A., Fersdahl, M., Chen, X., Li, Z., and Zimmerman, P. The Short Wave Aerostat-Mounted Imager (SWAMI): A novel platform for acquiring remotely sensed data from a tethered balloon. *Remote Sens. Environ. 103*, 3 (2006), 255–264.

[97] von Bueren, S. K., Burkart, A., Hueni, A., Rascher, U., Tuohy, M. P., and Yule, I. J. Deploying four optical UAV-based sensors over grassland: challenges and limitations. *Biogeosciences 12*, 1 (2015), 163–175.

[98] Wallace, L., Lucieer, A., Watson, C., and Turner, D. Development of a UAV-LiDAR system with application to forest inventory. *Remote Sens-Basel 6*, 4 (2012), 1519–1543.

[99] Wallace, L., Lucieer, A., and Watson, C. S. Evaluating tree detection and segmentation routines on very high resolution UAV LiDAR data. *IEEE T. Geosci. Remote. 52*, 12 (2014), 7619–7628.

[100] Wang, K., Wan, Z., Wang, P., Sparrow, M., Liu, J., Zhou, X., and Haginoya, S. Estimation of surface long wave radiation and broadband emissivity using Moderate Resolution Imaging Spectroradiometer (MODIS) land surface temperature/emissivity products. *J. Geophys. Res-Atmos. 110*, 11 (2005).

[101] Wang, Y., Zou, Y., Henrickson, K., Wang, Y., Tang, J., and Park, B.-J. Google Earth elevation data extraction and accuracy assessment for transportation applications. *PLoS ONE 12*, 4 (2017), 1–17.

[102] Whitehead, K., Hugenholtz, C. H., Myshak, S., Brown, O., LeClair, A., and Tamminga, A. Remote sensing of the environment with small unmanned aircraft systems (UASs), part 2: scientific and commercial applications. *J. Unmanned Veh. Syst. 2*, 3 (2014), 86–102.

[103] Wu, D., Li, R., Zhang, F., and Liu, J. A review on drone-based harmful algae blooms monitoring. *Enviorn. Monit. Assess.* (2019), 191–211.

[104] Xiong, X., Cao, C., and Chander, G. An overview of sensor calibration intercomparison and applications. *Front. Earth Sci. Chin. 4*, 2 (2010), 237–252.

[105] Xu, F., Gao, Z., Jiang, X., Shang, W., Ning, J., Song, D., and Ai, J. A UAV and S2A data-based estimation of the initial biomass of green algae in the South Yellow Sea. *Mar. Pollut. Bull. 128* (2018), 408–414.

[106] Zakšek, K., and Oštir, K. Downscaling land surface temperature for urban heat island diurnal cycle analysis. *Remote Sens. Environ. 117* (2012), 114–124.

[107] Zeise, B., Kleinschmidt, S. P., and Wagner, B. Improving the interpretation of thermal images with the aid of emissivity's angular dependency. In *2015 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)* (2015), pp. 1–8.

[108] Zhang, X., Zhong, T., Feng, X., and Wang, K. Estimation of the relationship between vegetation patches and urban land surface temperature with remote sensing. *Int. J. Remote Sens. 30*, 8 (2009), 2105–2118.

[109] Zhang, X. H., Guo, F., and Li, X. X. A novel Stop&Go GPS precise point positioning (PPP) method and its application in geophysical exploration and prospecting. *Surv. Rev. 44*, 327 (2012), 251–255.

Appendix A

Source Code

A.1 Thermal Camera Calibration

```
# Accurate as of October 11, 2019
# Code to optimize R, B, O, and F used in the ST calculation as a function of surface
    material

import pandas as pd
from pylab import *
import numpy as np
from lmfit import Minimizer, Parameters, report_fit

# Reads data from CSV file
# This is the signal value of the pixel, back-calculated by substituting in the manufacturer
    calculated
# R, B, O, and F values as well as the temperature value retreived from FLIR Tools
Upixel_data  = '/export/home/users/username/Documents/DG_Temp/Calibration/' \
               'Experiment_Data/Upixel.csv'

# This is the certified thermometer data
cert_therm_data  = '/export/home/users/username/Documents/DG_Temp/Calibration/' \
                   'Exoeriment_Data/CertifiedTemperature.csv'

# Import CSV data into Pandas dataframe
Upixel  = pd.read_csv(Upixel_data)

# Separate signal value columns for each surface material type
Upixel_grass  = (Upixel.values.astype(float64)[:,0])
Upixel_soil  = (Upixel.values.astype(float64)[:,1])
Upixel_concrete  = (Upixel.values.astype(float64)[:,2])
Upixel_water  = (Upixel.values.astype(float64)[:,3])

# Slice Upixel water array to remove N an values. Nan values cannot be present when using
    the lmfit library.
```

```
# Water temperature data was not collected for each time interval, Nan values cannot be
#  present when using the lmfit library
Upixel_water = Upixel_water[0:7]

# Call in the certified thermometer temperature data
cert_therm  = pd.read_csv(cert_therm_data)

# Separate certified temperature columns for each surface material type
cert_therm_grass  = (cert_therm.values.astype(float64)[:,0])
cert_therm_soil  = (cert_therm.values.astype(float64)[:,1])
cert_therm_concrete  = (cert_therm.values.astype(float64)[:,2])
cert_therm_water  = (cert_therm.values.astype(float64)[:,3])

# Slice thermometer water array. Nan values cannot be present when using the lmfit library.
# Water temperature data was not collected for each time interval
cert_therm_water = cert_therm_water[0:7]

##################################################################################################
# Create arrays for the signal value and certified temperature for each land surface
    material.
# Can onlt calibrate one surface material at a time.
# For Grass
Upixel_counts = np.array(Upixel_grass, dtype=float64)
cert_temp = np.array(cert_therm_grass, dtype=float64)

# # For Soil
# Upixel_counts = np.array(Upixel_soil, dtype=float64)
# cert_temp = np.array(cert_therm_soil, dtype=float64)

# # For Concrete
# Upixel_counts = np.array(Upixel_concrete, dtype=float64)
# cert_temp = np.array(cert_therm_concrete, dtype=float64)

# # For Water
# Upixel_counts = np.array(Upixel_water, dtype=float64)
# cert_temp = np.array(cert_therm_water, dtype=float64)
##################################################################################################
# Define objective function, return the array to be minimised
def fcn2min(params, Upixel_counts, cert_temp):
    B = params['B']
    R = params['R']
    O = params['O']
    F = params['F']

    model = B / np.log(R / (Upixel_counts + O) + F)

    return model - cert_temp


# Create a set of Parameters (R, B, O, and F) with the manufacturer value and maximum/
    minimum values
params = Parameters()
```

```
# Declare manufacturer constant values and set maximum/minimum possible constant values
params.add('B', value=1428, min=714, max=2142)
params.add('R', value=366545, min=183272, max=549817)
params.add('O', value=-342, min=-513, max=-171)
params.add('F', value=1, min=0.5, max=1.5)

# Use the least squares model and fit the curve to the imported data
minner = Minimizer(fcn2min, params, fcn_args=(Upixel_counts, cert_temp))
result = minner.minimize()

# Calculate the final result
final = cert_temp + result.residual

# Print Fit Statistics, Variables (R, B, O, and F) and Correlations
report_fit(result)

# Try to plot the non linear fit
try:
    import matplotlib.pyplot as plt

    plt.plot(Upixel_counts, cert_temp, 'k+')
    plt.plot(Upixel_counts, final, 'ro')
    plt.show()
except ImportError:
    pass
```

A.1.1 Thermal Camera Calibration Plots

```
import numpy
from matplotlib import pyplot as plt

# File with certified temperature
certified_temperature_file = '/export/home/users/username/Documents/DG_Temp/' \
                             'Calibration/Experiment_Data/CertifiedTemperature.csv'

# File with corresponding Upixel (raw signal) values
Upixel_values_file = '/export/home/users/username/Documents/' \
                     'DG_Temp/Calibration/Experiment_Data/Upixel.csv'

# Import certified temperature data
certified_temperature_data = numpy.genfromtxt(certified_temperature_file,delimiter=',')

# Separate Columns
cert_grass_temp = certified_temperature_data[:,0]
cert_soil_temp = certified_temperature_data[:,1]
cert_concrete_temp = certified_temperature_data[:,2]
cert_water_temp = certified_temperature_data[:,3]

#
    ##########################################################################################################################
```

```
# Import image Upixel (raw signal) data
Upixel_values_data = numpy.genfromtxt(Upixel_values_file, delimiter=',')

# Separate Columns
Upixel_values_grass = Upixel_values_data[:,0]
Upixel_values_soil = Upixel_values_data[:,1]
Upixel_values_concrete = Upixel_values_data[:,2]
Upixel_values_water = Upixel_values_data[:,3]

# Remove Nan data
Upixel_values_water = Upixel_values_water[0:8]

# State original FLIR factory Planck constants, from image metadata
R1_flir = 17096.453
R2_flir = 0.046642166
B_flir = 1428
O_flir =-342
F_flir =1

R_flir = R1_flir/R2_flir

# Back-calculate Temperature From Upixel values
# Initialize Arrays
backcalc_grass_temp = numpy.zeros((len(Upixel_values_grass)))
backcalc_soil_temp = numpy.zeros((len(Upixel_values_soil)))
backcalc_concrete_temp = numpy.zeros((len(Upixel_values_concrete)))
backcalc_water_temp = numpy.zeros((len(Upixel_values_water)))

# Back-calculate image pixel temperature based on Upixel data and Factory set FLIR Constants
for i in range(0, len(Upixel_values_grass)):
    backcalc_grass_temp[i] = B_flir/(numpy.log(R1_flir/(R2_flir*(Upixel_values_grass[i]+
        O_flir))+F_flir))
    backcalc_soil_temp[i] = B_flir/(numpy.log(R1_flir/(R2_flir*(Upixel_values_soil[i]+O_flir
        ))+F_flir))
    backcalc_concrete_temp[i] = B_flir/(numpy.log(R1_flir/(R2_flir*(Upixel_values_concrete[i
        ]+O_flir))+F_flir))

for i in range(0, len(backcalc_water_temp)):
    backcalc_water_temp[i] = B_flir / (numpy.log(R1_flir / (R2_flir * (Upixel_values_water[i
        ] + O_flir)) + F_flir))

# Calculate Error, Square Error, Percentage Error, Bias, RMSE, and average percentage error
#  between certified temperature and back-calculated temperature
# Initialize arrays
# Back-calculate Grass
backcalc_grass_error = numpy.zeros((len(Upixel_values_grass)))
backcalc_grass_sq_error = numpy.zeros((len(Upixel_values_grass)))
backcalc_grass_PE = numpy.zeros((len(Upixel_values_grass)))

# Back-calculate Soil
backcalc_soil_error = numpy.zeros((len(Upixel_values_soil)))
```

```
backcalc_soil_sq_error = numpy.zeros((len(Upixel_values_soil)))
backcalc_soil_PE = numpy.zeros((len(Upixel_values_soil)))

# Back-calculate Concrete
backcalc_concrete_error = numpy.zeros((len(Upixel_values_concrete)))
backcalc_concrete_sq_error = numpy.zeros((len(Upixel_values_concrete)))
backcalc_concrete_PE = numpy.zeros((len(Upixel_values_concrete)))

# Back-calculate Water
backcalc_water_error = numpy.zeros((len(Upixel_values_water)))
backcalc_water_sq_error = numpy.zeros((len(Upixel_values_water)))
backcalc_water_PE = numpy.zeros((len(Upixel_values_water)))

# Calculate error in back-calculated temperatures
for i in range(0, len(Upixel_values_grass)):
    backcalc_grass_error[i] = backcalc_grass_temp[i]-cert_grass_temp[i]
    backcalc_soil_error[i] = backcalc_soil_temp[i]-cert_soil_temp[i]
    backcalc_concrete_error[i] = backcalc_concrete_temp[i]-cert_concrete_temp[i]

for i in range(0, len(backcalc_water_temp)):
    backcalc_water_error[i] = backcalc_water_temp[i]-cert_water_temp[i]

# Calculate Temperature Bias for each material
backcalc_grass_bias = numpy.average(backcalc_grass_error)
backcalc_soil_bias = numpy.average(backcalc_soil_error)
backcalc_concrete_bias = numpy.average(backcalc_concrete_error)
backcalc_water_bias = numpy.average(backcalc_water_error)

# Calculate Square Error
for i in range(0, len(Upixel_values_grass)):
    backcalc_grass_sq_error[i] = backcalc_grass_error[i]**2
    backcalc_soil_sq_error[i] = backcalc_soil_error[i]**2
    backcalc_concrete_sq_error[i] = backcalc_concrete_error[i]**2

for i in range(0, len(backcalc_water_temp)):
    backcalc_water_sq_error[i] = backcalc_water_error[i]**2

# Calculate RMSE
backcalc_grass_RMSE = numpy.sqrt(numpy.average(backcalc_grass_sq_error))
backcalc_soil_RMSE = numpy.sqrt(numpy.average(backcalc_soil_sq_error))
backcalc_concrete_RMSE = numpy.sqrt(numpy.average(backcalc_concrete_sq_error))
backcalc_water_RMSE = numpy.sqrt(numpy.average(backcalc_water_sq_error))

# Calculate Percentage Error (PE)
for i in range(0, len(Upixel_values_grass)):
    backcalc_grass_PE[i] = numpy.abs((cert_grass_temp[i]-backcalc_grass_temp[i])/
        cert_grass_temp[i])*100
    backcalc_soil_PE[i] = numpy.abs((cert_soil_temp[i]-backcalc_soil_temp[i])/cert_soil_temp
        [i])*100
    backcalc_concrete_PE[i] = numpy.abs((cert_concrete_temp[i]-backcalc_concrete_temp[i])/
        cert_concrete_temp[i])*100
```

```
for i in range(0, len(backcalc_water_temp)):
    backcalc_water_PE[i] = numpy.abs((cert_water_temp[i]-backcalc_water_temp[i])/
        cert_water_temp[i])*100

# Calculate Average Percentage Error
backcalc_grass_average_PE = numpy.average(backcalc_grass_PE)
backcalc_soil_average_PE = numpy.average(backcalc_soil_PE)
backcalc_concrete_average_PE = numpy.average(backcalc_concrete_PE)
backcalc_water_average_PE = numpy.average(backcalc_water_PE)

#
    ##########################################################################################################################

# LMFIT (using the lmfit library) Planck Constants and Temperature back-calculate
# Grass
R_lmfit_grass = 314531
B_lmfit_grass = 1391
O_lmfit_grass = -513
F_lmfit_grass = 1.5

# Soil
R_lmfit_soil = 549800
B_lmfit_soil = 1510
O_lmfit_soil = -171
F_lmfit_soil = 1.5

# Concrete
R_lmfit_concrete = 247614
B_lmfit_concrete = 1322
O_lmfit_concrete = -513
F_lmfit_concrete = 1.5

# Water
R_lmfit_water = 549789
B_lmfit_water = 1507
O_lmfit_water = -171
F_lmfit_water = 1.5

# Back-calculate temperature given new constants and Upixel values
# Initialize arrays
# Grass
lmfit_grass_temp = numpy.zeros((len(Upixel_values_grass)))
lmfit_grass_error = numpy.zeros((len(Upixel_values_grass)))
lmfit_grass_sq_error = numpy.zeros((len(Upixel_values_grass)))

# Soil
lmfit_soil_temp = numpy.zeros((len(Upixel_values_soil)))
lmfit_soil_error = numpy.zeros((len(Upixel_values_soil)))
lmfit_soil_sq_error = numpy.zeros((len(Upixel_values_soil)))

# Concrete
lmfit_concrete_temp = numpy.zeros((len(Upixel_values_concrete)))
```

```
lmfit_concrete_error = numpy.zeros((len(Upixel_values_concrete)))
lmfit_concrete_sq_error = numpy.zeros((len(Upixel_values_concrete)))

# Water
lmfit_water_temp = numpy.zeros((len(Upixel_values_water)))
lmfit_water_error = numpy.zeros((len(Upixel_values_water)))
lmfit_water_sq_error = numpy.zeros((len(Upixel_values_water)))

# Back-calculate temperature accordingly
for i in range(0, len(Upixel_values_grass)):
    lmfit_grass_temp[i] = B_lmfit_grass / (numpy.log(R_lmfit_grass / (Upixel_values_grass[i]
        + O_lmfit_grass) + F_lmfit_grass))
    lmfit_soil_temp[i] = B_lmfit_soil / (numpy.log(R_lmfit_soil / (Upixel_values_soil[i] +
        O_lmfit_soil) + F_lmfit_soil))
    lmfit_concrete_temp[i] = B_lmfit_concrete / (numpy.log(R_lmfit_concrete / (
        Upixel_values_concrete[i] + O_lmfit_concrete) + F_lmfit_concrete))

for i in range(0, len(backcalc_water_temp)):
    lmfit_water_temp[i] = B_lmfit_water / (numpy.log(R_lmfit_water / (Upixel_values_water[i]
        + O_lmfit_water) + F_lmfit_water))

# Calculate error, Square Error, bias and RMSE for back-calculated LMFIT temperature
# Initialize arrays
# LMFIT error
for i in range(0, len(Upixel_values_grass)):
    lmfit_grass_error[i] = lmfit_grass_temp[i] - cert_grass_temp[i]
    lmfit_soil_error[i] = lmfit_soil_temp[i] - cert_soil_temp[i]
    lmfit_concrete_error[i] = lmfit_concrete_temp[i] - cert_concrete_temp[i]

for i in range(0, len(backcalc_water_temp)):
    lmfit_water_error[i] = lmfit_water_temp[i] - cert_water_temp[i]

# Bias calculation
lmfit_grass_bias = numpy.average(lmfit_grass_error)
lmfit_soil_bias = numpy.average(lmfit_soil_error)
lmfit_concrete_bias = numpy.average(lmfit_concrete_error)
lmfit_water_bias = numpy.average(lmfit_water_error)

# LMFIT square error
for i in range(0, len(Upixel_values_grass)):
    lmfit_grass_sq_error[i] = lmfit_grass_error[i]**2
    lmfit_soil_sq_error[i] = lmfit_soil_error[i]**2
    lmfit_concrete_sq_error[i] = lmfit_concrete_error[i]**2

for i in range(0, len(backcalc_water_temp)):
    lmfit_water_sq_error[i] = lmfit_water_error[i]**2

# Calculate RMSE
lmfit_grass_rmse = numpy.sqrt(numpy.average(lmfit_grass_sq_error))
lmfit_soil_rmse = numpy.sqrt(numpy.average(lmfit_soil_sq_error))
lmfit_concrete_rmse = numpy.sqrt(numpy.average(lmfit_concrete_sq_error))
lmfit_water_rmse = numpy.sqrt(numpy.average(lmfit_water_sq_error))
```

```
#
    ##########################################################################################################################

# Plot Upixel vs certified temperature for Grass, soil, concrete, and water considering the
    calibration
#  experiment, the calibrated camera parameters, and the default camera parameters
fig_direct = '/export/home/users/username/Documents/DG_Temp/' \
             'Calibration/Calibrated_Figures/'
# Grass
plt.figure()
# Calibration experiment
plt.scatter(Upixel_values_grass, cert_grass_temp, c='k', marker='+')
# Default Camera Parameters
plt.scatter(Upixel_values_grass, backcalc_grass_temp, c='g', marker='d')
# Caibrated camera parameters
plt.scatter(Upixel_values_grass, lmfit_grass_temp, c='r', marker='*')
plt.xlabel('Uobject [A/D Counts]')
plt.ylabel('Certified Temperature [K]')
plt.legend(['Calibration Experiment', 'Default Camera Parameters', 'Calibrated Camera
    Parameters'])
plt.savefig(fig_direct+'GrassCalibration.png')
plt.show()

# Soil
plt.figure()
# Calibration experiment
plt.scatter(Upixel_values_soil, cert_soil_temp, c='k', marker='+')
# Default Camera Parameters
plt.scatter(Upixel_values_soil, backcalc_soil_temp, c='g', marker='d')
# Caibrated camera parameters
plt.scatter(Upixel_values_soil, lmfit_soil_temp, c='r', marker='*')
plt.xlabel('Uobject [A/D Counts]')
plt.ylabel('Certified Temperature [K]')
plt.legend(['Calibration Experiment', 'Default Camera Parameters', 'Calibrated Camera
    Parameters'])
plt.savefig(fig_direct+'SoilCalibration.png')
plt.show()

# Concrete/Developed land
plt.figure()
# Calibration experiment
plt.scatter(Upixel_values_concrete, cert_concrete_temp, c='k', marker='+')
# Default Camera Parameters
plt.scatter(Upixel_values_concrete, backcalc_concrete_temp, c='g', marker='d')
# Caibrated camera parameters
plt.scatter(Upixel_values_concrete, lmfit_concrete_temp, c='r', marker='*')
plt.xlabel('Uobject [A/D Counts]')
plt.ylabel('Certified Temperature [K]')
plt.legend(['Calibration Experiment', 'Default Camera Parameters', 'Calibrated Camera
    Parameters'])
plt.savefig(fig_direct+'DevelopedLandCalibration.png')
```

```
plt.show()

# Remove Nan water data
cert_water_temp = cert_water_temp[0:8]

# Water
plt.figure()
# Calibration experiment
plt.scatter(Upixel_values_water, cert_water_temp, c='k', marker='+')
# Default Camera Parameters
plt.scatter(Upixel_values_water, backcalc_water_temp, c='g', marker='d')
# Caibrated camera parameters
plt.scatter(Upixel_values_water, lmfit_water_temp, c='r', marker='*')
plt.xlabel('Uobject [A/D Counts]')
plt.ylabel('Certified Temperature [K]')
plt.legend(['Calibration Experiment', 'Default Camera Parameters', 'Calibrated Camera
    Parameters'])
plt.savefig(fig_direct+'WaterCalibration.png')
plt.show()
```

A.2 Mining Site Campaign

A.2.1 TriSonica Atmospheric Pressure to Altitude

```
# Code to load in Trisonica data, concatenate data for entire May 2018 campaign, and
    calculate 1 second averaged data
# Current as of October 16, 2019
import sys
import numpy
import datetime
import matplotlib.dates as dates
from datetime import date

# Call in Trisonica data for each day after completing preliminary
#  processing (using excel/libre office Calc to remove colon/character
#  delimiters and remove Nan data etc. such that data can easily
#  be post processed in Python)

# The TriSonica data has been pre-processed before being loaded into this script. Indices of
    ascending and
#  descending TANAB2 launches have been identifed from another script and were used to
    create a file used in
#  this script identifying the ground level Pressure (Altitude) at the start of each TANAB2
    profile

# May 7/2018
# Call in all Pressure Data including year, month, hour, minute, seconds
fileName_MFT = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/TriSonica
    /Unaveraged/' \
               'Cleaned_Files/Individual_Files/2018-05-07-Data.txt'
```

```
data_MFT = numpy.genfromtxt(fileName_MFT, skip_header=5, invalid_raise=False,
                            usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                filling_values=numpy.nan)

# Separate columns
year_MFT = data_MFT[:,0]
month_MFT = data_MFT[:,1]
day_MFT = data_MFT[:,2]
hour_MFT = data_MFT[:,3]
minute_MFT = data_MFT[:,4]
seconds_MFT = data_MFT[:,5]
temp_MFT = data_MFT[:,6]
P_MFT = data_MFT[:,7]

# May 9/2018
fileName_09_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
              'Cleaned_Files/Individual_Files/2018-05-09-Data.txt'

data_09_05_2018 = numpy.genfromtxt(fileName_09_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_09_05_2018 = data_09_05_2018[:,0]
month_09_05_2018 = data_09_05_2018[:,1]
day_09_05_2018 = data_09_05_2018[:,2]
hour_09_05_2018 = data_09_05_2018[:,3]
minute_09_05_2018 = data_09_05_2018[:,4]
seconds_09_05_2018 = data_09_05_2018[:,5]
temp_09_05_2018 = data_09_05_2018[:,6]
P_09_05_2018 = data_09_05_2018[:,7]

# May 10/2018
fileName_10_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
              'Cleaned_Files/Individual_Files/2018-05-10-Data.txt'

data_10_05_2018 = numpy.genfromtxt(fileName_10_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_10_05_2018 = data_10_05_2018[:,0]
month_10_05_2018 = data_10_05_2018[:,1]
day_10_05_2018 = data_10_05_2018[:,2]
hour_10_05_2018 = data_10_05_2018[:,3]
minute_10_05_2018 = data_10_05_2018[:,4]
seconds_10_05_2018 = data_10_05_2018[:,5]
temp_10_05_2018 = data_10_05_2018[:,6]
P_10_05_2018 = data_10_05_2018[:,7]
```

```
# May 15/2018
fileName_15_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-15-Data.txt'

data_15_05_2018 = numpy.genfromtxt(fileName_15_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_15_05_2018 = data_15_05_2018[:,0]
month_15_05_2018 = data_15_05_2018[:,1]
day_15_05_2018 = data_15_05_2018[:,2]
hour_15_05_2018 = data_15_05_2018[:,3]
minute_15_05_2018 = data_15_05_2018[:,4]
seconds_15_05_2018 = data_15_05_2018[:,5]
temp_15_05_2018 = data_15_05_2018[:,6]
P_15_05_2018 = data_15_05_2018[:,7]

# May 18/2018
fileName_18_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-18-Data.txt'

data_18_05_2018 = numpy.genfromtxt(fileName_18_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_18_05_2018 = data_18_05_2018[:,0]
month_18_05_2018 = data_18_05_2018[:,1]
day_18_05_2018 = data_18_05_2018[:,2]
hour_18_05_2018 = data_18_05_2018[:,3]
minute_18_05_2018 = data_18_05_2018[:,4]
seconds_18_05_2018 = data_18_05_2018[:,5]
temp_18_05_2018 = data_18_05_2018[:,6]
P_18_05_2018 = data_18_05_2018[:,7]

# May 19/2018
fileName_19_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-19-Data.txt'

data_19_05_2018 = numpy.genfromtxt(fileName_19_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_19_05_2018 = data_19_05_2018[:,0]
month_19_05_2018 = data_19_05_2018[:,1]
day_19_05_2018 = data_19_05_2018[:,2]
```

```
hour_19_05_2018 = data_19_05_2018[:,3]
minute_19_05_2018 = data_19_05_2018[:,4]
seconds_19_05_2018 = data_19_05_2018[:,5]
temp_19_05_2018 = data_19_05_2018[:,6]
P_19_05_2018 = data_19_05_2018[:,7]

# May 21/2018
fileName_21_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/TriSonica/2018-05-21-Data.txt'

data_21_05_2018 = numpy.genfromtxt(fileName_21_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_21_05_2018 = data_21_05_2018[:,0]
month_21_05_2018 = data_21_05_2018[:,1]
day_21_05_2018 = data_21_05_2018[:,2]
hour_21_05_2018 = data_21_05_2018[:,3]
minute_21_05_2018 = data_21_05_2018[:,4]
seconds_21_05_2018 = data_21_05_2018[:,5]
temp_21_05_2018 = data_21_05_2018[:,6]
P_21_05_2018 = data_21_05_2018[:,7]

# May 23/2018
fileName_23_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-23-Data.txt'

data_23_05_2018 = numpy.genfromtxt(fileName_23_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_23_05_2018 = data_23_05_2018[:,0]
month_23_05_2018 = data_23_05_2018[:,1]
day_23_05_2018 = data_23_05_2018[:,2]
hour_23_05_2018 = data_23_05_2018[:,3]
minute_23_05_2018 = data_23_05_2018[:,4]
seconds_23_05_2018 = data_23_05_2018[:,5]
temp_23_05_2018 = data_23_05_2018[:,6]
P_23_05_2018 = data_23_05_2018[:,7]

# May 24/2018
fileName_24_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-24-Data.txt'

data_24_05_2018 = numpy.genfromtxt(fileName_24_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)
```

```
# Separate columns
year_24_05_2018 = data_24_05_2018[:,0]
month_24_05_2018 = data_24_05_2018[:,1]
day_24_05_2018 = data_24_05_2018[:,2]
hour_24_05_2018 = data_24_05_2018[:,3]
minute_24_05_2018 = data_24_05_2018[:,4]
seconds_24_05_2018 = data_24_05_2018[:,5]
temp_24_05_2018 = data_24_05_2018[:,6]
P_24_05_2018 = data_24_05_2018[:,7]

# May 27/2018
fileName_27_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/TriSonica/2018-05-27-Data.txt'

data_27_05_2018 = numpy.genfromtxt(fileName_27_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_27_05_2018 = data_27_05_2018[:,0]
month_27_05_2018 = data_27_05_2018[:,1]
day_27_05_2018 = data_27_05_2018[:,2]
hour_27_05_2018 = data_27_05_2018[:,3]
minute_27_05_2018 = data_27_05_2018[:,4]
seconds_27_05_2018 = data_27_05_2018[:,5]
temp_27_05_2018 = data_27_05_2018[:,6]
P_27_05_2018 = data_27_05_2018[:,7]

# May 30/2018
fileName_30_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                'Cleaned_Files/Individual_Files/2018-05-30-Data.txt'

data_30_05_2018 = numpy.genfromtxt(fileName_30_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_30_05_2018 = data_30_05_2018[:,0]
month_30_05_2018 = data_30_05_2018[:,1]
day_30_05_2018 = data_30_05_2018[:,2]
hour_30_05_2018 = data_30_05_2018[:,3]
minute_30_05_2018 = data_30_05_2018[:,4]
seconds_30_05_2018 = data_30_05_2018[:,5]
temp_30_05_2018 = data_30_05_2018[:,6]
P_30_05_2018 = data_30_05_2018[:,7]

# May 31/2018
fileName_31_05_2018 = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
```

```
                'Cleaned_Files/Individual_Files/2018-05-31-Data.txt'

data_31_05_2018 = numpy.genfromtxt(fileName_31_05_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                       filling_values=numpy.nan)

# Separate columns
year_31_05_2018 = data_31_05_2018[:,0]
month_31_05_2018 = data_31_05_2018[:,1]
day_31_05_2018 = data_31_05_2018[:,2]
hour_31_05_2018 = data_31_05_2018[:,3]
minute_31_05_2018 = data_31_05_2018[:,4]
seconds_31_05_2018 = data_31_05_2018[:,5]
temp_31_05_2018 = data_31_05_2018[:,6]
P_31_05_2018 = data_31_05_2018[:,7]

# Concatenate columns together
year = numpy.concatenate([year_MFT, year_09_05_2018, year_10_05_2018, year_15_05_2018,
    year_18_05_2018,
                          year_19_05_2018, year_21_05_2018, year_23_05_2018, year_24_05_2018
                              ,
                          year_27_05_2018, year_30_05_2018, year_31_05_2018])
month = numpy.concatenate([month_MFT, month_09_05_2018, month_10_05_2018, month_15_05_2018,
    month_18_05_2018,
                           month_19_05_2018, month_21_05_2018, month_23_05_2018,
                               month_24_05_2018,
                           month_27_05_2018, month_30_05_2018, month_31_05_2018])
day = numpy.concatenate([day_MFT, day_09_05_2018, day_10_05_2018, day_15_05_2018,
    day_18_05_2018,
                         day_19_05_2018, day_21_05_2018, day_23_05_2018, day_24_05_2018,
                             day_27_05_2018,
                         day_30_05_2018, day_31_05_2018])
hour = numpy.concatenate([hour_MFT, hour_09_05_2018, hour_10_05_2018, hour_15_05_2018,
    hour_18_05_2018,
                          hour_19_05_2018, hour_21_05_2018, hour_23_05_2018, hour_24_05_2018
                              , hour_27_05_2018,
                          hour_30_05_2018, hour_31_05_2018])
minute = numpy.concatenate([minute_MFT, minute_09_05_2018, minute_10_05_2018,
    minute_15_05_2018,
                            minute_18_05_2018, minute_19_05_2018, minute_21_05_2018,
                                minute_23_05_2018,
                            minute_24_05_2018, minute_27_05_2018, minute_30_05_2018,
                                minute_31_05_2018])
seconds = numpy.concatenate([seconds_MFT, seconds_09_05_2018, seconds_10_05_2018,
    seconds_15_05_2018,
                             seconds_18_05_2018, seconds_19_05_2018, seconds_21_05_2018,
                                 seconds_23_05_2018,
                             seconds_24_05_2018, seconds_27_05_2018, seconds_30_05_2018,
                                 seconds_31_05_2018])
temp = numpy.concatenate(([temp_MFT, temp_09_05_2018, temp_10_05_2018, temp_15_05_2018,
    temp_18_05_2018,
                           temp_19_05_2018, temp_21_05_2018, temp_23_05_2018,
```

```
                                    temp_24_05_2018, temp_27_05_2018,
                                temp_30_05_2018, temp_31_05_2018]))
P = numpy.concatenate([P_MFT, P_09_05_2018, P_10_05_2018, P_15_05_2018, P_18_05_2018,
    P_19_05_2018, P_21_05_2018,
                           P_23_05_2018, P_24_05_2018, P_27_05_2018, P_30_05_2018, P_31_05_2018
                               ])

# Calculate Altitudes from pressure
# From another analysis (via first order fit of hypsometric equation) define 1st
#  order polyfit coefficients to convert pressure to altitude alt=a*p+b
a = -8.51514286e+00
b = 8.62680905e+03

# Initialize altitude array
altitude = numpy.zeros(len(year))
print(P)

# Calculate altitude for each index
for i in range(0, len(altitude)):
    altitude[i] =(a*P[i])+b

# Calculate the number of minutes passed since the beginning of 2018 (based on day of year)
doy_min = numpy.zeros(len(year))
for i in range(0, len(year)):
    if day[i] == 7:
        doy_min[i] = (127*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 8:
        doy_min[i] = (128*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 9:
        doy_min[i] = (129*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 10:
        doy_min[i] = (130*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 15:
        doy_min[i] = (135*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 18:
        doy_min[i] = (138*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 19:
        doy_min[i] = (139*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 21:
        doy_min[i] = (141*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 23:
        doy_min[i] = (143*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 24:
```

```
        doy_min[i] = (144*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 27:
        doy_min[i] = (147*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 30:
        doy_min[i] = (150*24*60)+(hour[i]*60)+minute[i]

    elif day[i] == 31:
        doy_min[i] = (151*24*60)+(hour[i]*60)+minute[i]

    else:
        print('More Dates need to be included above')


# Save Unaveraged concatenated variables and altitudes to file
today_date = datetime.date.today().strftime("%B_%d_%Y")

outputFileName = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                 'Cleaned_Files/TriSonica_May_2018_Unaveraged_Airborne_Altitudes.txt'
outputFile = open(outputFileName, 'w')
outputFile.write("# Data collected by TriSonica for May 2018 field campaign. Includes
    Altitude calculation \n")
outputFile.write("#By: Ryan Byerlay \n")
outputFile.write("#Created on "+today_date+" \n")
outputFile.write("#Recorded Time is Local Time (MDT) \n")
outputFile.write("#0:Year \t #1:Month \t #2:Day \t #3:Hour \t #4:Minute \t #5:Seconds \t"
                 " #6:DOY In Minutes \t  #7:Pressure \t #8:Altitude \n")
for i in range(0, len(year)):
    outputFile.write("%i \t %i \t %i \t %i \t %i \t %f"
                     " \t %f \t %f \t %f \t %f \n" % (year[i], month[i], day[i], hour[i],
                         minute[i],
                                                     seconds[i], doy_min[i], P[i], altitude
                                                         [i], temp[i]))
outputFile.close()

#
    ##########################################################################################################################

# Complete 1 second averaging
# Second Averaging (10 Hz Frequency)
AverageSample = 10

# Total number of samples
Ntotal = numpy.size(altitude)
NSample = int(Ntotal/AverageSample)


# Calculate 1 second averaged altitudes, year, month, day, hour, minute, second,
#  pressure, day of year in minutes, and temperature averages for each sample
yearavg = numpy.zeros((NSample, 1))
```

```
monthavg = numpy.zeros((NSample, 1))
houravg = numpy.zeros((NSample, 1))
dayavg = numpy.zeros((NSample, 1))
minuteavg = numpy.zeros((NSample, 1))
secondavg = numpy.zeros((NSample, 1))
pressureavg = numpy.zeros((NSample, 1))
altitudeavg = numpy.zeros((NSample, 1))
doy_min_avg = numpy.zeros((NSample, 1))
temp_avg = numpy.zeros((NSample, 1))

# Calculate Averages
for i in range(0, NSample):
    yearavg[i] = numpy.mean(year[i*AverageSample:(i+1)*AverageSample])
    monthavg[i] = numpy.mean(month[i*AverageSample:(i+1)*AverageSample])
    houravg[i] = numpy.mean(hour[i*AverageSample:(i+1)*AverageSample])
    dayavg[i] = numpy.mean(day[i*AverageSample:(i+1)*AverageSample])
    minuteavg[i] = numpy.mean(minute[i*AverageSample:(i+1)*AverageSample])
    secondavg[i] = numpy.mean(seconds[i*AverageSample:(i+1)*AverageSample])
    pressureavg[i] = numpy.mean(P[i*AverageSample:(i+1)*AverageSample])
    altitudeavg[i] = numpy.mean(altitude[i*AverageSample:(i+1)*AverageSample])
    doy_min_avg[i] = numpy.mean(doy_min[i*AverageSample:(i+1)*AverageSample])
    temp_avg[i] = numpy.mean(temp[i*AverageSample:(i+1)*AverageSample])

# The following is to the fix the issue where in the averaged second column,
#  the 59th averaged second does not equal 59, instead it is a lower value. For example:
#  58, 59, 0 are being averaged together so the value will be < 59
# Include a condition to omit Nan values too
# When saving, put in a check to see if the values are Nan, if index has Nan values, then
    skip index
for i in range(0, len(secondavg)):
    print(secondavg[i])
    if numpy.isnan(secondavg[i]) == True:
        continue

    else:
        print(type(secondavg[i]))
        print(int(secondavg[i]))
        if int(secondavg[i]) == 58 and int(secondavg[i+1]) != 59:
            yearavg[i+1] = numpy.nan
            monthavg[i+1] = numpy.nan
            houravg[i+1] = numpy.nan
            dayavg[i+1] = numpy.nan
            minuteavg[i+1] = numpy.nan
            secondavg[i+1] = numpy.nan
            pressureavg[i+1] = numpy.nan
            altitudeavg[i+1] = numpy.nan
            doy_min_avg[i+1] = numpy.nan
            temp_avg[i+1] = numpy.nan

# Call in TriSonica Base Altitude files: Previously created from the indices files for
#  each TANAB2 launch. The indices file identifies the time and location of each TANAB2
    launch.
```

```
fileName_BaseAlt = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/Unaveraged/' \
                   'Cleaned_Files/Ascending_Descending_TANAB2/BaseAltitudesTriSonica_V3.txt'

data_BaseAlt = numpy.genfromtxt(fileName_BaseAlt, skip_header=5)

day_BaseAlt = data_BaseAlt[:,2]
hour_BaseAlt = data_BaseAlt[:,3]
minute_BaseAlt = data_BaseAlt[:,4]
doy_BaseAlt = data_BaseAlt[:,6]
BasePressure_trisonica = data_BaseAlt[:,7]
BaseAlt_trisonica = data_BaseAlt[:,8]
Location_BaseAlt = data_BaseAlt[:,9]

# Initialize variable that stores the difference between day of year in minutes
#  for the TriSonica base altitude and for the second averaged file
delta_doy = numpy.zeros((len(data_BaseAlt),1))

# Initialize BaseAltitude used to calculate change in altitude and index_Base to
#  indicate the index with the smallest day of year in minutes difference
BaseAltitude = numpy.zeros((NSample,1))
BasePressure = numpy.zeros((NSample,1))
Location = numpy.empty(NSample, dtype=str)
index_Base = numpy.zeros((NSample,1))

# Match day of year in minutes indices from Base Altitude file & second averaged file
# loop through averaged day of year in minutes indices
for i in range(0, len(yearavg)):
    # Skip Nan values
    if numpy.isnan(yearavg[i]) == True:
        continue
    else:
        # Loop through Base Altitude indices
        for j in range(0, len(day_BaseAlt)):
            # Calculate difference between day of year in minutes times
            delta_doy[j] = abs(doy_BaseAlt[j]-doy_min_avg[i])

            # When at the last value of the Base Altitude file, find the index with the
                minimum value
            # Also record location and index of the Base Altitude file
            if j == (len(BaseAlt_trisonica)-1):
                BaseAltitude[i] = BaseAlt_trisonica[numpy.argmin(delta_doy)]
                BasePressure[i] = BasePressure_trisonica[numpy.argmin(delta_doy)]
                index_Base[i] = numpy.argmin(delta_doy[j])

# Initialize variables for change in atmospheric pressure and change in altitude relative to
    the ground
deltaPressure = numpy.zeros((NSample,1))
deltaAltitude  = numpy.zeros((NSample,1))

# Calculate the change in pressure and the change in altitude and assign geographic
    locations to each index
```

```
for i in range(0, NSample):
    # Skip Nan values
    if numpy.isnan(pressureavg[i]) == True:
        continue
    else:
        deltaPressure[i] = abs(BasePressure[i]-pressureavg[i])
        deltaAltitude[i] = abs(BaseAltitude[i]-altitudeavg[i])

    if dayavg[i] == 7 or dayavg[i] == 8:
        Location[i] = 'T'
    elif dayavg[i] == 9 or dayavg[i] == 10 or dayavg[i] == 15 or dayavg[i] == 30 or dayavg[i
        ] == 31:
        Location[i] = 'B'
    elif dayavg[i] == 18 or dayavg[i] == 19 or dayavg[i] == 21 or dayavg[i] == 23 or dayavg[
        i] == 24 or dayavg[i] == 27:
        Location[i] = 'M'
    else:
        # Note: Need to manually remove rows with 'N' in the resulting text file
        Location[i] = 'N'
        print('Day falls outside of the ones listed above')

print(BaseAltitude)

# Save Averaged data to file
outputFileName_avg = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    TriSonica/' \
                     'TriSonica_May2018_Altitudes_averaged.txt'

outputFile_avg = open(outputFileName_avg, 'w')
outputFile_avg.write("# Second Avgeraged Data collected by TriSonica for May"
                     " 2018 field campiagn. Includes Altitude calculation \n")
outputFile_avg.write("#By: Ryan Byerlay \n")
outputFile_avg.write("#Created on "+today_date+" \n")
outputFile_avg.write("#Recorded Time is Local Time (MDT) \n")
outputFile_avg.write("#0:AvgYear \t #1:AvgMonth \t #2:AvgDay \t"
                     " #3:AvgHour \t #4:AvgMinute \t #5:AvgSeconds \t #6:Location (T=MFT/B=
                         Berm/M=Mine/N=NoMatch)"
                     " \t #7:Day of Year in minutes \t #8:AvgPressure \t #9:BasePressure \t"
                     " #10:DeltaPressure \t #11:AvgAltitude \t #12:BaseAltitude \t #13:
                         DeltaAltitude \t"
                     " #14:Temperature(degC) \t #15:Index in BaseAlt File \n")
for i in range(0, NSample):
    # If index has Nan value, skip and do not write
    if numpy.isnan(yearavg[i]) == True:
        continue
    else:
        outputFile_avg.write("%i \t %i \t %i \t %i \t %i"
                             " \t %i \t %s \t %f \t %f \t %f"
                             " \t %f \t %f \t %f \t %f \t %f"
                             " \t %i \n" % (yearavg[i], monthavg[i], dayavg[i], houravg[i],
                                 minuteavg[i],
                                            secondavg[i], Location[i], doy_min_avg[i],
```

```
                                        pressureavg[i], BasePressure[i],
                                    deltaPressure[i], altitudeavg[i], BaseAltitude[i
                                        ], deltaAltitude[i],
                                    temp_avg[i], index_Base[i]))
outputFile_avg.close()
```

A.2.2 Spatial Coordinate Grid Overlaid on Mine Site

```
import numpy
import math
from math import radians
from math import degrees
from math import asin
from math import tan
from math import atan2
from math import sin
from math import cos
from math import acos
from math import sqrt
from math import atan

# Current as of October 16, 2019
# Given a desired spatial resolution, coordinates about an approximately
# 40km by 30km rectangle around mine are calculated and saved to a file

# The created file can be imported into QGIS and the coordinates can be laid on top of
    satellite images/raster
#  files (MODIS emissivity, surface temperature and surface elevation etc.) and data from
#  the images can be extracted for each point

# Calculate latitude/longitude coordinates based on the defined spatial resolution
def SiteCoordinatesCalc(res_x, y_iterator, TLeft_lat_rads, R, TLeft_lon_rads, GPS_matrix):
    for a in range(0, res_x):
        for b in range(0, res_y):
            if a == 0 and b == 0:
                continue
            elif a == 0 and b != 0:
                d_km = y_iterator[b-1]
                Yaw = 180
                Yaw_rads = math.radians(Yaw)

                lat2 = math.asin(math.sin(TLeft_lat_rads) * math.cos(d_km / R) + math.cos(
                    TLeft_lat_rads)
                                 * math.sin(d_km / R) * math.cos(Yaw_rads))
                lon2 = TLeft_lon_rads + math.atan2(math.sin(Yaw_rads) * math.sin(d_km / R) *
                     math.cos(TLeft_lat_rads),
                                                    math.cos(d_km / R) - math.sin(
                                                        TLeft_lat_rads) * math.sin(lat2))

                # Convert back to decimal degrees
                lat2 = math.degrees(lat2)
```

```
            lon2 = math.degrees(lon2)

            # Save to GPS Matrix
            GPS_matrix[a][b][0] = lat2
            GPS_matrix[a][b][1] = lon2
            continue

        elif b == 0 and a != 0:
            d_km = x_iterator[a-1]
            Yaw = 90
            Yaw_rads = math.radians(Yaw)

            lat2 = math.asin(math.sin(TLeft_lat_rads) * math.cos(d_km / R) + math.cos(
                TLeft_lat_rads)
                             * math.sin(d_km / R) * math.cos(Yaw_rads))
            lon2 = TLeft_lon_rads + math.atan2(math.sin(Yaw_rads) * math.sin(d_km / R) *
                 math.cos(TLeft_lat_rads),
                                               math.cos(d_km / R) - math.sin(
                                                   TLeft_lat_rads) * math.sin(lat2))

            # Convert back to decimal degrees
            lat2 = math.degrees(lat2)
            lon2 = math.degrees(lon2)

            # Save to GPS Matrix
            GPS_matrix[a][b][0] = lat2
            GPS_matrix[a][b][1] = lon2
            continue

        else:
            d_km = y_iterator[b]
            Yaw = 180
            Yaw_rads = math.radians(Yaw)

            lat2 = math.asin(math.sin(math.radians(GPS_matrix[a][0][0]))*math.cos(d_km /
                 R)
                             + math.cos(math.radians(GPS_matrix[a][0][0])) * math.sin(
                                 d_km/R)*math.cos(Yaw_rads))

            lon2 = math.radians(GPS_matrix[a][0][1]) + \
                   math.atan2(math.sin(Yaw_rads)*math.sin(d_km/R) * math.cos(math.
                        radians(GPS_matrix[a][0][0])),
                                 math.cos(d_km/R)-math.sin(math.radians(GPS_matrix[a
                                     ][0][0]))*math.sin(lat2))

            # Convert back to decimal degrees
            lat2 = math.degrees(lat2)
            lon2 = math.degrees(lon2)

            # Save to GPS Matrix
            GPS_matrix[a][b][0] = lat2
            GPS_matrix[a][b][1] = lon2
```

```
    return GPS_matrix

# Get GPS coordinates for an approximately 40km by 30km rectangle around mine
# Identify the Top Left latitude/longitude
# For the horizontal loop
# Use the top left as a reference point
TLeft_lat = XX.XXXXXX
TLeft_lon = -XXX.XXXXXX

# Convert degrees to radians
TLeft_lat_rads = radians(TLeft_lat)
TLeft_lon_rads = radians(TLeft_lon)

# Ending lat/lon
TRight_lat = XX.XXXXXX
TRight_lon = -XXX.XXXXX

# For the vertical loop
# Starting latitude/longitude
BRight_lat = XX.XXXXXX
BRight_lon = -XXX.XXXXXX

# Ending latitude/longitude
BLeft_lat = XX.XXXXXX
BLeft_lon = -XXX.XXXXXX

# Equatorial radius of earth as per: https://nssdc.gsfc.nasa.gov/planetary/factsheet/
    earthfact.html
R = 6378.1

#
    ##########################################################################################################################

# Number of horizontal and vertical squares per spatial resolution
# 500m by 500m resolution
res500_x = 113
res500_y = 114

# 100m by 100m resolution
res100_x = 570
res100_y = 570

# 1km by 1km resolution
res1000_x = 57
res1000_y = 57

#
    ##########################################################################################################################

# For code below use the following resolution
res_x = res100_x
```

```
res_y = res100_y

# Create array for declaring the iterator to be used for the horizontal and vertical
    directions
x_iterator = numpy.zeros((res_x,1))
y_iterator = numpy.zeros((res_y,1))

# Initialize longitude/latitude matrix
GPS_matrix = numpy.zeros((res_x, res_y, 2))

# Assign Known GPS latitude/longitude to matrix
GPS_matrix[0][0][0] = TLeft_lat
GPS_matrix[0][0][1] = TLeft_lon

# The number of GPS coordinates to save to file
len_save = res_x*res_y
save_GPS_matrix = numpy.zeros((len_save,2))
save_GPS_matrix[:] = numpy.nan

# Set up x_iterator vector for left to right GPS coordinates
if res_x == res100_x:
    d_iterator = 0.1
elif res_x == res500_x:
    d_iterator = 0.5
elif res_x == res1000_x:
    d_iterator = 1

for i in range(0, res_x):
    if i == 0:
        x_iterator[i] = d_iterator
    else:
        x_iterator[i] = d_iterator*(i+1)

# Set up y_iterator vector for top to bottom GPS coordinates
for i in range(0, res_y):
    if i == 0:
        y_iterator[i] = d_iterator
    else:
        y_iterator[i] = d_iterator*(i+1)


# Calculate new latitude and longitude coordinates 500m apart from each other and save to
    text file
# Create Site Grid
SiteCoordinatesCalc(res_x, y_iterator, TLeft_lat_rads, R, TLeft_lon_rads, GPS_matrix)

# Save GPS coordinates in matrix
for i in range(0,res_x):
    for j in range(0,res_y):
        for k in range(0, len_save):
            if numpy.isnan(save_GPS_matrix[k][0]) == True:
                save_GPS_matrix[k][0] = GPS_matrix[i][j][0]
```

```
                save_GPS_matrix[k][1] = GPS_matrix[i][j][1]
                break

# Save Data to file
if res_x == res100_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
        MODIS/QGIS/' \
                     '100m_Resolution_Mine_Site_Grid.csv'
elif res_x == res500_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
        MODIS/QGIS/' \
                     '500m_Resolution_Mine_Site_Grid.csv'
elif res_x == res1000_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
        MODIS/QGIS/' \
                     '1000m_Resolution_Mine_Site_Grid.csv'

outputFile = open(outputFileName, 'w')

numpy.savetxt(outputFileName, save_GPS_matrix, delimiter=',', fmt='%f', header='#0:Lat,#1:
    Lon')
```

A.2.3 Emissivity Data Retrieval

```
import numpy

# Current as of October 16, 2019

# Import outputted emissivity data extracted from QGIS for the two adjacent MODIS images
# If one MODIS latitude/longitude is zero/Nan, the other image should have the data
# Identify latitude/longitude with emissivity of 0, Nan etc. and output
#  CSV with latitude/longitude and a single emissivity value for each band

# Load emissivity data (3 bands) from the two adjacent MODIS images
emis_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/MODIS/
    Emissivity/QGIS/' \
                'Emis_29_31_32_May_2018_MODIS11B3_C6_V2.txt'

emis_data = numpy.genfromtxt(emis_filename, delimiter=',', skip_header=1)

# Assign emissivity_data to variables
Latitude = emis_data[:,0]
Longitude = emis_data[:,1]

N_emis_29 = emis_data[:,2]
N_emis_31 = emis_data[:,3]
N_emis_32 = emis_data[:,4]

S_emis_29 = emis_data[:,5]
S_emis_31 = emis_data[:,6]
S_emis_32 = emis_data[:,7]
```

```
# Create new vectors for latitudes and longitudes with zero emissivity
Lat_zero = numpy.empty((len(Latitude),1))
Lat_zero[:] = numpy.nan
Lon_zero = numpy.empty((len(Longitude),1))
Lon_zero[:] = numpy.nan

# Create new vectors for emissivity in bands 29, 31, and 32
#  corresponding to each respective latitude/longitude coordinate
Lat_emis = numpy.empty((len(Latitude),1))
Lat_emis[:] = numpy.nan
Lon_emis = numpy.empty((len(Latitude),1))
Lon_emis[:] = numpy.nan

emis_29 = numpy.empty((len(Latitude),1))
emis_29[:] = numpy.nan

emis_31 = numpy.empty((len(Latitude),1))
emis_31[:] = numpy.nan

emis_32 = numpy.empty((len(Latitude),1))
emis_32[:] = numpy.nan

# If any emissivity value is 0, write the latitude/longitude location to CSV
for i in range(0, len(Latitude)):
    # Check if zero
    if N_emis_29[i] == 0 and N_emis_31[i] == 0 and N_emis_32[i] == 0\
            and S_emis_29[i] == 0 and S_emis_31[i] == 0 and S_emis_32[i] == 0:
        for j in range(0, len(Lat_zero)):
            if numpy.isnan(Lat_zero[j]) == True:
                Lat_zero[j] = Latitude[i]
                Lon_zero[j] = Longitude[i]
                break

    # Check if North image is zero, if so, write South image data to appropriate arrays
    elif (N_emis_29[i] == 0 and N_emis_31[i] == 0 and N_emis_32[i] == 0)\
            and (S_emis_29[i] != 0 and S_emis_31[i] != 0 and S_emis_32[i] != 0):
        for j in range(len(Lat_emis)):
            if numpy.isnan(Lat_emis[j]) == True:
                Lat_emis[j] = Latitude[i]
                Lon_emis[j] = Longitude[i]

                emis_29[j] = S_emis_29[i]
                emis_31[j] = S_emis_31[i]
                emis_32[j] = S_emis_32[i]
                break

    # Check if South image is zero, if so write North image data to appropriate arrays
    elif (N_emis_29[i] != 0 and N_emis_31[i] != 0 and N_emis_32[i] != 0)\
            and (S_emis_29[i] == 0 and S_emis_31[i] == 0 and S_emis_32[i] == 0):
        for j in range(len(Lat_emis)):
            if numpy.isnan(Lat_emis[j]) == True:
```

```
                Lat_emis[j] = Latitude[i]
                Lon_emis[j] = Longitude[i]

                emis_29[j] = N_emis_29[i]
                emis_31[j] = N_emis_31[i]
                emis_32[j] = N_emis_32[i]
                break

    # Check to see if the emissivity values from the north image = the emissivity values
        from the south image
    elif (N_emis_29[i] == S_emis_29[i] and N_emis_31[i] == S_emis_31[i] and N_emis_32[i] ==
        S_emis_32[i]):
        for j in range(len(Lat_emis)):
            if numpy.isnan(Lat_emis[j]) == True:
                Lat_emis[j] = Latitude[i]
                Lon_emis[j] = Longitude[i]

                # If both the south and north image data are the same,
                #  it doesnt matter which data is saved to the final array
                emis_29[j] = N_emis_29[i]
                emis_31[j] = N_emis_31[i]
                emis_32[j] = N_emis_32[i]
                break

    # If emissivity values from both images are present for each band and
    #  they are different values, average the band emissivities
    elif (N_emis_29[i] > 0 and S_emis_29[i] > 0 and N_emis_29[i] != S_emis_29[i]) or\
            (N_emis_31[i] > 0 and S_emis_31[i] > 0 and N_emis_31[i] != S_emis_31[i]) or\
            (N_emis_32[i] > 0 and S_emis_32[i] > 0 and N_emis_32[i] != S_emis_32[i]):
        for j in range(len(Lat_emis)):
            if numpy.isnan(Lat_emis[j]) == True:
                Lat_emis[j] = Latitude[i]
                Lon_emis[j] = Longitude[i]

                # If both the south and north image data are the same,
                #  it doesnt matter which data is saved to the final array
                emis_29[j] = float((N_emis_29[i]+S_emis_29[i])/2)
                emis_31[j] = float((N_emis_31[i]+S_emis_31[i])/2)
                emis_32[j] = float((N_emis_32[i]+S_emis_32[i])/2)
                break


# Column stack latitude zero/longitude zero and emissivity bands with the corresponding
    latitudes and longitudes
GPS_zero = numpy.column_stack((Lat_zero,Lon_zero))
GPS_emis = numpy.column_stack(((Lat_emis,Lon_emis, emis_29,emis_31,emis_32)))

# Save combined emissivity/latitude/longitude file
outputFileName_emis = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    MODIS/Emissivity/' \
                      'May_2018_MODIS_Lat_Lon_Emissivity.csv'
outputFile_emis = open(outputFileName_emis, 'w')
```

```
# Save zero emissivity/latitude/longitude file
numpy.savetxt(outputFileName_emis, GPS_emis, delimiter=',',
              header='#0:Lat,#1:Lon,#2:Band_29_Emissivity,#3:Band_31_Emissivity,#4:
                  Band_32_Emissivity')
```

A.2.4 Direct Georeferencing and Temperature Calculation

```
# Calculate temperatures (degK and degC) and GPS position from individual pixels within each
    image
# Created By: Ryan Byerlay On: April 26, 2018, Current as of: October 15, 2019

# Successfully works on ImageMagick (IM) 7.0.7 and ExifTool 10.94 on a linux OS
#  (both Ubuntu 16.04 and Ubuntu 18.04) with Python 3.5
# NOTE: Syntax for IM before 7 is different
# NOTE: The Raw Thermal Image Type must be TIFF, to check put image in same folder as IM7,
#  ExifTool and this script and type the following into the command line: "exiftool filename
    .jpg"
# NOTE: May need to install ubuntu/linux developer tools for TIFF, PNG, JPEG etc as IM 7 may
#  not be able to process images
# NOTE: Updated versions of ExifTool may have more functionality for FLIR Images, may result
#  in improved quantitative image analysis
# Check here for the latest on ExifTool: https://www.sno.phy.queensu.ca/~phil/exiftool/
#
    ##########################################################################################################################


import os
import subprocess
import numpy
import time
import math
from math import tan
from math import sin
from math import cos
from math import asin
from math import sqrt
from math import atan2
from math import radians
from math import atan
import datetime
import simplekml
from numba import jit
import pytemperature

# Data derived from the Geocontext-Profiler
# (http://www.geocontext.org/publ/2010/04/profiler/en/) and saved to text documents
def LandSlopeEquations (BaseAltitude, heading):
    # Convert the heading from float to int
    heading_int = int(heading)
```

```
# Degree of poly fit. Use degree of 3 for Earth surface elevation as
# per: https://doi.org/10.1080/13658810310001596058
poly_deg = 3

# For the Berm (TANAB2 launch location)
if BaseAltitude == 337:
    # For the North direction
    # Between N (0 deg) and NNE (22.5 deg) or NNW (337.5 deg) and N (360 deg)
    if heading_int >= 0 and heading_int < 22.5 or heading_int > 337.5 and heading_int
        <=360:
        # Elevation data for 10 km due North of the TANAB2 launch location
        Berm_N_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'Berm/Berm_N.txt'

        # Load distance and elevation data from file
        Berm_N_data = numpy.genfromtxt(Berm_N_filename, delimiter=',')

        # Distance away from the TANAB2 launch location in meters
        distance_Berm_N = Berm_N_data[:, 0]

        # Elevation above sea level in meters for each data point away from the TANAB2
            launch location
        elevation_Berm_N = Berm_N_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        # the TANAB2 launch location and the corresponding elevation above sea level in
            meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_N, elevation_Berm_N, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the North direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_N)

        # Detrend the resulting land surface elevations calculated from the derived
        # polynomial with respect to the elevation data derived from Geocontext-Profiler
        ground_elev_AGL = elevation_Berm_N - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the North-East direction
    # Between NNE (22.5 deg) and ENE (67.5 deg)
    elif heading_int > 22.5 and heading_int < 67.5:
        # Elevation data for 10 km due North East of the TANAB2 launch location
        Berm_NE_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'Berm/Berm_NE.txt'

        # Load data from file
        Berm_NE_data = numpy.genfromtxt(Berm_NE_filename, delimiter=',')

        # Distance away from the TANAB2 launch location in meters
```

```
        distance_Berm_NE = Berm_NE_data[:, 0]

        # Elevation above sea level in meters for each data point away from the TANAB2
            launch location in meters
        elevation_Berm_NE = Berm_NE_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        # the TANAB2 launch location and the corresponding surface elevation above sea
            level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_NE, elevation_Berm_NE, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the North East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_NE)

        # Detrend the resulting land surface elevations from the derived
        #  polynomial with respect to the elevation data derived from the Geocontext-
            Profiler
        ground_elev_AGL = elevation_Berm_NE - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the East direction
    # Between ENE (67.5 deg) and ESE (112.5 deg)
    elif heading_int > 67.5 and heading_int < 112.5:
        # Elevation data for 10 km due East of the TANAB2 launch location
        Berm_E_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                          'Berm/Berm_E.txt'

        # Load data from file
        Berm_E_data = numpy.genfromtxt(Berm_E_filename, delimiter=',')

        # Distance away from the TANAB2 launch location in meters
        distance_Berm_E = Berm_E_data[:, 0]

        # Elevation above sea level in meters for each data point away from the TANAB2
            launch location
        elevation_Berm_E = Berm_E_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_E, elevation_Berm_E, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_E)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
```

```
        ground_elev_AGL = elevation_Berm_E - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South-East direction
    # Between ESE (112.5 deg) and SSE (157.5 deg)
    elif heading_int > 112.5 and heading_int < 157.5:
        # Elevation data for 10 km due South East of the TANAB2 launch location
        Berm_SE_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'Berm/Berm_SE.txt'

        # Load data from file
        Berm_SE_data = numpy.genfromtxt(Berm_SE_filename, delimiter=',')

        # Distance away from TANAB2 launch location in meters
        distance_Berm_SE = Berm_SE_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Berm_SE = Berm_SE_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_SE, elevation_Berm_SE, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_SE)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Berm_SE - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South direction
    # Between SSE (157.5 deg) and SSW (202.5 deg)
    elif heading_int > 157.5 and heading_int < 202.5:
        # Elevation data for 10 km due South of the TANAB2 launch location
        Berm_S_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'Berm/Berm_S.txt'

        # Load data from file
        Berm_S_data = numpy.genfromtxt(Berm_S_filename, delimiter=',')

        # Distance away from TANAB2 launch location in meters
        distance_Berm_S = Berm_S_data[:, 0]
```

```
        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Berm_S = Berm_S_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_S, elevation_Berm_S, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_S)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Berm_S - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South-West direction
    # Between SSW (202.5 deg) and WSW (247.5 deg)
    elif heading_int > 202.5 and heading_int < 247.5:
        # Elevation data for 10 km due South West of the TANAB2 launch location
        Berm_SW_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                           'Berm/Berm_SW.txt'

        # Load data from file
        Berm_SW_data = numpy.genfromtxt(Berm_SW_filename, delimiter=',')

        # Distance away from TANAB2 launch location in meters
        distance_Berm_SW = Berm_SW_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Berm_SW = Berm_SW_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_SW, elevation_Berm_SW, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_SW)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Berm_SW - ground_elev_ASL_fitted
```

```
        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the West direction
    # Between WSW (247.5 deg) and WNW (292.5 deg)
    elif heading_int > 247.5 and heading_int < 292.5:
        # Elevation data for 10 km due West of the TANAB2 launch location
        Berm_W_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                          'Berm/Berm_W.txt'

        # Load data from file
        Berm_W_data = numpy.genfromtxt(Berm_W_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Berm_W = Berm_W_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Berm_W = Berm_W_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Berm_W, elevation_Berm_W, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_W)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Berm_W - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the North-West direction
    # Between WNW (292.5 deg) and NNW (337.5 deg)
    elif heading_int > 292.5 and heading_int < 337.5:
        # Elevation data for 10 km due North West of the TANAB2 launch location
        Berm_NW_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                          'Berm/Berm_NW.txt'

        # Load data from file
        Berm_NW_data = numpy.genfromtxt(Berm_NW_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Berm_NW = Berm_NW_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
```

```
            elevation_Berm_NW = Berm_NW_data[:, 1]

            # Returns coefficients for the polyfit equation between the distance away from
            #  the TANAB2 launch location and the corresponding elevation above sea level in
                 meters
            Land_Poly_Coeff = numpy.polyfit(distance_Berm_NW, elevation_Berm_NW, poly_deg)

            # Evaluate the polynomial at specific values as given by the distance away from
            #  the TANAB2 launch location in the North West direction
            ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Berm_NW)

            # Detrend the resulting land surface elevations calculated from the derived
            #  polynomial with respect to the elevation data derived from Geocontext-
                Profiler
            ground_elev_AGL = elevation_Berm_NW - ground_elev_ASL_fitted

            return ground_elev_ASL_fitted, ground_elev_AGL

    # For the Mine (TANAB2 launch location)
    elif BaseAltitude == 317:
        # For the North direction
        # Between N (0 deg/360 deg), NNE (22.5 deg) and NNW (337.5 deg)
        if heading_int >= 0 and heading_int < 22.5 or heading_int > 337.5 and heading_int
            <=360:
            # Elevation data for 10 km due North of the TANAB2 launch location
            Mine_N_filename = '/export/home/users/username/Documents/DG_Temp/
                Mining_Facility_2018/Elevation_Data/' \
                              'Mine/Mine_N.txt'

            # Load data from file
            Mine_N_data = numpy.genfromtxt(Mine_N_filename, delimiter=',')

            # Distance from TANAB2 launch location in meters
            distance_Mine_N = Mine_N_data[:, 0]

            # Elevation above sea level in meters for each data point away from TANAB2
                launch location
            elevation_Mine_N = Mine_N_data[:, 1]

            # Returns coefficients for the polyfit equation between the distance away from
            #  the TANAB2 launch location and the corresponding elevation above sea level in
                 meters
            Land_Poly_Coeff = numpy.polyfit(distance_Mine_N, elevation_Mine_N, poly_deg)

            # Evaluate the polynomial at specific values as given by the distance away from
            #  the TANAB2 launch location in the North direction
            ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_N)

            # Detrend the resulting land surface elevations calculated from the derived
            #  polynomial with respect to the elevation data derived from Geocontext-
                Profiler
            ground_elev_AGL = elevation_Mine_N - ground_elev_ASL_fitted
```

```
    return ground_elev_ASL_fitted, ground_elev_AGL

# For the North-East direction
# Between NNE (22.5 deg) and ENE (67.5 deg)y
elif heading_int > 22.5 and heading_int < 67.5:
    # Elevation data for 10 km due North East of the TANAB2 launch location
    Mine_NE_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                    'Mine/Mine_NE.txt'

    # Load data from file
    Mine_NE_data = numpy.genfromtxt(Mine_NE_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_Mine_NE = Mine_NE_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_Mine_NE = Mine_NE_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
         meters
    Land_Poly_Coeff = numpy.polyfit(distance_Mine_NE, elevation_Mine_NE, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from
    #  the TANAB2 launch location in the North East direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_NE)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_Mine_NE - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the East direction
# Between ENE (67.5 deg) and ESE (112.5 deg)
elif heading_int > 67.5 and heading_int < 112.5:
    # Elevation data for 10 km due East of the TANAB2 launch location
    Mine_E_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                    'Mine/Mine_E.txt'

    # Load data from file
    Mine_E_data = numpy.genfromtxt(Mine_E_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_Mine_E = Mine_E_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
```

```
        launch location
    elevation_Mine_E = Mine_E_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
        meters
    Land_Poly_Coeff = numpy.polyfit(distance_Mine_E, elevation_Mine_E, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from
    #  the TANAB2 launch location in the East direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_E)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_Mine_E - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the South-East direction
# Between ESE (112.5 deg) and SSE (157.5 deg)
elif heading_int > 112.5 and heading_int < 157.5:
    # Elevation data for 10 km due South East of the TANAB2 launch location
    Mine_SE_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                       'Mine/Mine_SE.txt'

    # Load data from file
    Mine_SE_data = numpy.genfromtxt(Mine_SE_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_Mine_SE = Mine_SE_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_Mine_SE = Mine_SE_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
        meters
    Land_Poly_Coeff = numpy.polyfit(distance_Mine_SE, elevation_Mine_SE, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from
    #  the TANAB2 launch location in the South East direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_SE)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_Mine_SE - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL
```

```
# For the South direction
# Between SSE (157.5 deg) and SSW (202.5 deg)
elif heading_int > 157.5 and heading_int < 202.5:
    # Elevation data for 10 km due South of the TANAB2 launch location
    Mine_S_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                      'Mine/Mine_S.txt'

    # Load data from file
    Mine_S_data = numpy.genfromtxt(Mine_S_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_Mine_S = Mine_S_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_Mine_S = Mine_S_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
         meters
    Land_Poly_Coeff = numpy.polyfit(distance_Mine_S, elevation_Mine_S, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from
    #  the TANAB2 launch location in the South direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_S)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_Mine_S - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the South-West direction
# Between SSW (202.5 deg) and WSW (247.5 deg)
elif heading_int > 202.5 and heading_int < 247.5:
    # Elevation data for 10 km due South West of the TANAB2 launch location
    Mine_SW_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                      'Mine/Mine_SW.txt'

    # Load data from file
    Mine_SW_data = numpy.genfromtxt(Mine_SW_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_Mine_SW = Mine_SW_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_Mine_SW = Mine_SW_data[:, 1]
```

```
        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
            meters
        Land_Poly_Coeff = numpy.polyfit(distance_Mine_SW, elevation_Mine_SW, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_SW)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Mine_SW - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the West direction
    # Between WSW (247.5 deg) and WNW (292.5 deg)
    elif heading_int > 247.5 and heading_int < 292.5:
        # Elevation data for 10 km due West of the TANAB2 launch location
        Mine_W_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                          'Mine/Mine_W.txt'

        # Load data from file
        Mine_W_data = numpy.genfromtxt(Mine_W_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Mine_W = Mine_W_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Mine_W = Mine_W_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
            meters
        Land_Poly_Coeff = numpy.polyfit(distance_Mine_W, elevation_Mine_W, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_W)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Mine_W - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the North-West direction
```

```
    # Between WNW (292.5 deg) and NNW (337.5 deg)
    elif heading_int > 292.5 and heading_int < 337.5:
        # Elevation data for 10 km due North West of the TANAB2 launch location
        Mine_NW_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'Mine/Mine_NW.txt'

        # Load data from file
        Mine_NW_data = numpy.genfromtxt(Mine_NW_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Mine_NW = Mine_NW_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_Mine_NW = Mine_NW_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_Mine_NW, elevation_Mine_NW, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the North West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Mine_NW)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_Mine_NW - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

# For MFT (TANAB2 launch location)
elif BaseAltitude == 398:
    # For the North direction
    # Between N (0 deg/360 deg), NNE (22.5 deg) and NNW (337.5 deg)
    if heading_int >= 0 and heading_int < 22.5 or heading_int > 337.5 and heading_int
        <=360:
        # Elevation data for 10 km due North of the TANAB2 launch location
        MFT_N_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                        'MFT/MFT_N.txt'

        # Load data from file
        MFT_N_data = numpy.genfromtxt(MFT_N_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_MFT_N = MFT_N_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
```

```
        elevation_MFT_N = MFT_N_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_MFT_N, elevation_MFT_N, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the North direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_N)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
             Profiler
        ground_elev_AGL = elevation_MFT_N - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the North-East direction
    # Between NNE (22.5 deg) and ENE (67.5 deg)
    elif heading_int > 22.5 and heading_int < 67.5:
        # Elevation data for 10 km due North East of the TANAB2 launch location
        MFT_NE_filename = '/export/home/users/username/Documents/DG_Temp/
             Mining_Facility_2018/Elevation_Data/' \
                          'MFT/MFT_NE.txt'

        # Load data from file
        MFT_NE_data = numpy.genfromtxt(MFT_NE_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_MFT_NE = MFT_NE_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
             launch location
        elevation_MFT_NE = MFT_NE_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
             meters
        Land_Poly_Coeff = numpy.polyfit(distance_MFT_NE, elevation_MFT_NE, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the North East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_NE)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
             Profiler
        ground_elev_AGL = elevation_MFT_NE - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL
```

```
# For the East direction
# Between ENE (67.5 deg) and ESE (112.5 deg)
elif heading_int > 67.5 and heading_int < 112.5:
    # Elevation data for 10 km due East of the TANAB2 launch location
    MFT_E_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                     'MFT/MFT_E.txt'

    # Load data from file
    MFT_E_data = numpy.genfromtxt(MFT_E_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_MFT_E = MFT_E_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_MFT_E = MFT_E_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
         meters
    Land_Poly_Coeff = numpy.polyfit(distance_MFT_E, elevation_MFT_E, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away
    # from the TANAB2 launch location in the East direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_E)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_MFT_E - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the South-East direction
# Between ESE (112.5 deg) and SSE (157.5 deg)
elif heading_int > 112.5 and heading_int < 157.5:
    # Elevation data for 10 km due South East of the TANAB2 launch location
    MFT_SE_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                     'MFT/MFT_SE.txt'

    # Load data from file
    MFT_SE_data = numpy.genfromtxt(MFT_SE_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_MFT_SE = MFT_SE_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_MFT_SE = MFT_SE_data[:, 1]
```

```
        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
            meters
        Land_Poly_Coeff = numpy.polyfit(distance_MFT_SE, elevation_MFT_SE, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_SE)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_MFT_SE - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South direction
    # Between SSE (157.5 deg) and SSW (202.5 deg)
    elif heading_int > 157.5 and heading_int < 202.5:
        # Elevation data for 10 km due South of the TANAB2 launch location
        MFT_S_filename = '/export/home/users/username/Documents/DG_Temp/
            Mining_Facility_2018/Elevation_Data/' \
                         'MFT/MFT_S.txt'

        # Load data from file
        MFT_S_data = numpy.genfromtxt(MFT_S_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_MFT_S = MFT_S_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2
            launch location
        elevation_MFT_S = MFT_S_data[:, 1]

        # Returns coefficients for the polyfit equation between the distance away from
        #  the TANAB2 launch location and the corresponding elevation above sea level in
            meters
        Land_Poly_Coeff = numpy.polyfit(distance_MFT_S, elevation_MFT_S, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from
        #  the TANAB2 launch location in the South direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_S)

        # Detrend the resulting land surface elevations calculated from the derived
        #  polynomial with respect to the elevation data derived from Geocontext-
            Profiler
        ground_elev_AGL = elevation_MFT_S - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South-West direction
    # Between SSW (202.5 deg) and WSW (247.5 deg)
```

```
elif heading_int > 202.5 and heading_int < 247.5:
    # Elevation data for 10 km due South West of the TANAB2 launch location
    MFT_SW_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                      'MFT/MFT_SW.txt'

    # Load data from file
    MFT_SW_data = numpy.genfromtxt(MFT_SW_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_MFT_SW = MFT_SW_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_MFT_SW = MFT_SW_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
         meters
    Land_Poly_Coeff = numpy.polyfit(distance_MFT_SW, elevation_MFT_SW, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from
    #  the TANAB2 launch location in the South West direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_SW)

    # Detrend the resulting land surface elevations calculated from the derived
    #  polynomial with respect to the elevation data derived from Geocontext-
        Profiler
    ground_elev_AGL = elevation_MFT_SW - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the West direction
# Between WSW (247.5 deg) and WNW (292.5 deg)
elif heading_int > 247.5 and heading_int < 292.5:
    # Elevation data for 10 km due West of the TANAB2 launch location
    MFT_W_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/Elevation_Data/' \
                     'MFT/MFT_W.txt'

    # Load data from file
    MFT_W_data = numpy.genfromtxt(MFT_W_filename, delimiter=',')

    # Distance from TANAB2 launch location in meters
    distance_MFT_W = MFT_W_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2
        launch location
    elevation_MFT_W = MFT_W_data[:, 1]

    # Returns coefficients for the polyfit equation between the distance away from
    #  the TANAB2 launch location and the corresponding elevation above sea level in
```

```
                meters
            Land_Poly_Coeff = numpy.polyfit(distance_MFT_W, elevation_MFT_W, poly_deg)

            # Evaluate the polynomial at specific values as given by the distance away from
            #  the TANAB2 launch location in the West direction
            ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_W)

            # Detrend the resulting land surface elevations calculated from the derived
            #  polynomial with respect to the elevation data derived from Geocontext-
                Profiler
            ground_elev_AGL = elevation_MFT_W - ground_elev_ASL_fitted

            return ground_elev_ASL_fitted, ground_elev_AGL

        # For the North-West direction
        # Between WNW (292.5 deg) and NNW (337.5 deg)
        elif heading_int > 292.5 and heading_int < 337.5:
            # Elevation data for 10 km due North West of the TANAB2 launch location
            MFT_NW_filename = '/export/home/users/username/Documents/DG_Temp/
                Mining_Facility_2018/Elevation_Data/' \
                              'MFT/MFT_NW.txt'

            # Load data from file
            MFT_NW_data = numpy.genfromtxt(MFT_NW_filename, delimiter=',')

            # Distance from TANAB2 launch location in metersqualifying transaction
            distance_MFT_NW = MFT_NW_data[:, 0]

            # Elevation above sea level in meters for equalifying transactionach data point
                away from TANAB2 launch location
            elevation_MFT_NW = MFT_NW_data[:, 1]

            # Returns coefficients for the polyfit equation between the distance away from
            #  the TANAB2 launch location and the corresponding elevation above sea level in
                meters
            Land_Poly_Coeff = numpy.polyfit(distance_MFT_NW, elevation_MFT_NW, poly_deg)

            # Evaluate the polynomial at specific values as given by the distance away from
            #  the TANAB2 launch location in the North West direction
            ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_MFT_NW)

            # Detrend the resulting land surface elevations calculated from the derived
            #  polynomial with respect to the elevation data derived from Geocontext-
                Profiler
            ground_elev_AGL = elevation_MFT_NW - ground_elev_ASL_fitted

            return ground_elev_ASL_fitted, ground_elev_AGL

#
    ##########################################################################################################################

# NOTE: Using the Numba library and the @jit (Just In Time compiler),
```

```
# these functions are sped up with parallel processing as this function
#  is executed in another compiler in the computer after the code is transformed to machine
    code

# This library supports CUDA GPU processing within Python

# This library is continually being updated and future versions should have increased
#  functionality with respect to parallel processing, GPU/CUDA processing from a Python
    script

#
    ##########################################################################################################################

# Calculate pixel distance to assign emissivity value. For selected pixels
#  Use the JIT compiler to translate Python/numpy code into machine code
#  that is executed in parallel with the Python code

# This compiler reduced the run time of the code by 90%
# The following formulas are based off of:
# https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-
    latitude-longitude
@jit(nopython=True, parallel=True)
def HaversinePixelCalc(emis_lat, lat2_pixel, emis_lon, lon2_pixel, Radius_Earth, haversine_d
    ):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_pixel)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_pixel)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_pixel))
             * \
                    math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d

# Calculate pixel distance to assign emissivity value for the top left pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_left(emis_lat, lat2_top_left, emis_lon, lon2_top_left,
                                Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
```

```
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_top_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_top_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_top_left)) * \
                    math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d

# Calculate pixel distance to assign emissivity value for the top center pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_center(emis_lat, lat2_top, emis_lon, lon2_top, Radius_Earth,
    haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_top)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_top)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_top)) *
            \
                    math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d

# Calculate pixel distance to assign emissivity value for the top right pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_right(emis_lat, lat2_top_right, emis_lon, lon2_top_right,
```

```
                                    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_top_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_top_right)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(radians(lat2_top_right))
            * \
                    math.cos(radians(emis_lat[k])) * math.sin(haversine_dlon / 2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the center left pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center_left(emis_lat, lat2_center_left, emis_lon, lon2_center_left,
                                   Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_center_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_center_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_center_left)) * \
                    math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the center pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center(emis_lat, lat2_center, emis_lon, lon2_center, Radius_Earth,
    haversine_d):
```

```
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_center)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_center)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_center)
            ) * \
                      math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                          2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the center right pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center_right(emis_lat, lat2_center_right, emis_lon, lon2_center_right
    ,
                                    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_center_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_center_right)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_center_right)) * \
                      math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                          2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the bottom left pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom_left(emis_lat, lat2_bottom_left, emis_lon, lon2_bottom_left,
```

```
                                    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_bottom_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_bottom_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_bottom_left)) * \
                      math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                          2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the bottom center pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom(emis_lat, lat2_bottom, emis_lon, lon2_bottom, Radius_Earth,
    haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_bottom)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_bottom)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_bottom)
            ) * \
                      math.cos(math.radians(emis_lat[k])) * math.sin(haversine_dlon / 2) **
                          2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value for the bottom right pixel
# Use the JIT compiler to translate Python/numpy code into machine
#  code that is executed in parallel with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom_right(emis_lat, lat2_bottom_right, emis_lon, lon2_bottom_right
```

```
    ,
                                    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat[k] - lat2_bottom_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon[k] - lon2_bottom_right)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_bottom_right)) * \
                      math.cos(radians(emis_lat[k])) * math.sin(haversine_dlon / 2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

#
    ##########################################################################################################################

# Save picture data to master file (master file saves to text file with all image data
#  within the 'RawImages' folder
# Save image_matrix data to master matrix
@jit(nopython=True, parallel=True)
def SaveMasterMatrix(x_pixel_range, v_pixel_top, y_pixel_range, image_matrix,
                     all_pixel_data_multi_image, filename_image, filenames_total):
    for i in range(0, x_pixel_range):
        for j in range(v_pixel_top, y_pixel_range):
            # If a real value exists with latitude/longitude etc, then save to master matrix
            if numpy.isnan(image_matrix[i][j][5]) == False:
                for k in range(0, len(all_pixel_data_multi_image)):
                    # Write data to matrix if index is Nan
                    if numpy.isnan(all_pixel_data_multi_image[k][0]) == True:
                        # Save name of file to master array
                        filenames_total[k][0] = filename_image[i][j][0]

                        # Save year image was taken to master array
                        all_pixel_data_multi_image[k][0] = image_matrix[i][j][0]
                        # Save month image was taken to master array
                        all_pixel_data_multi_image[k][1] = image_matrix[i][j][1]
                        # Save day image was taken to master array
                        all_pixel_data_multi_image[k][2] = image_matrix[i][j][2]
                        # Save hour image was taken to master array
                        all_pixel_data_multi_image[k][3] = image_matrix[i][j][3]
                        # Save minute image was taken to master array
                        all_pixel_data_multi_image[k][4] = image_matrix[i][j][4]
                        # Save calculated geographic latitude to master array
                        all_pixel_data_multi_image[k][5] = image_matrix[i][j][5]
                        # Save calculated geographic longitude to master array
```

```
                        all_pixel_data_multi_image[k][6] = image_matrix[i][j][6]
                        # Save the horizontal pixel value of the image where ST was
                            calculated
                        #  to master array
                        all_pixel_data_multi_image[k][7] = image_matrix[i][j][7]
                        # Save the vertical pixel value of the image where ST was calculated
                        #  to master array
                        all_pixel_data_multi_image[k][8] = image_matrix[i][j][8]
                        # Save the ST in kelvin considering the MODIS emissivity of the land
                             surface
                        #  to master array
                        all_pixel_data_multi_image[k][9] = image_matrix[i][j][9]
                        # Save the ST in degC considering the MODIS emissivity of the land
                            surface to
                        #  master array
                        all_pixel_data_multi_image[k][10] = image_matrix[i][j][10]
                        break
    return all_pixel_data_multi_image

#
    ##########################################################################################################################################

# Calculate the start time of the script
start = time.time()

# Directory where RAW 'DJI_XXX.jpg' images to process are located
directory =  '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/RawImages'

# Return the number of images in 'RawImages' directory
numFiles = sum([len(files) for r, d, files in os.walk(directory)])

# Loop through each thermal image in the 'RawImages' directory
for file in os.listdir(directory):
    # Read the filename that would be shown in the Linux Terminal
    filename = os.fsdecode(file)
    print('The Image being processed now is: '+str(filename))

    # Extract variables in image metadata used for georeferencing calculations
    # Extract GPS Latitude from image via the Linux terminal with ExifTool
    # (ExifTool converts to decimal degrees)
    Latitude = subprocess.Popen(["exiftool -b -GPSLatitude " + directory + "/" + filename],
                                shell=True, stdout=subprocess.PIPE).communicate()[0]
    # Decode Latitude to data type string
    Latitude = Latitude.decode("utf-8")
    # Convert from string to float
    Latitude = float(Latitude)

    # Extract GPS Longitude from image via the Linux terminal with ExifTool
    #  (ExifTool converts to decimal degrees)
    Longitude = subprocess.Popen(["exiftool -b -GPSLongitude " + directory + "/" + filename
        ],
                                 shell=True, stdout=subprocess.PIPE).communicate()[0]
```

```
# Decode Longitude to data type string
Longitude = Longitude.decode("utf-8")
# Convert from string to float
Longitude = float(Longitude)

# Extract camera Gimbal Roll Degree from image via the Linux terminal with ExifTool
gRollDeg = subprocess.Popen(["exiftool -b -GimbalRollDegree " + directory + "/" +
    filename],
                            shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Gimbal Roll Degree to data type string
gRollDeg = gRollDeg.decode("utf-8")
# Convert from string to float
gRollDeg = float(gRollDeg)

# Extract camera Gimbal Yaw Degree from image via the Linux terminal with ExifTool
gYawDeg = subprocess.Popen(["exiftool -b -GimbalYawDegree " + directory + "/" + filename
    ],
                           shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Gimbal Yaw Degree to data type string
gYawDeg = gYawDeg.decode("utf-8")
# Convert from string to float
gYawDeg = float(gYawDeg)

# Extract camera Gimbal Pitch Degree from image via the Linux terminal with ExifTool
gPitchDeg = subprocess.Popen(["exiftool -b -GimbalPitchDegree " + directory + "/" +
    filename],
                             shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Gimbal Pitch Degree to data type string
gPitchDeg = gPitchDeg.decode("utf-8")
# Convert from string to float
gPitchDeg = float(gPitchDeg)

# Extract Flight (Gondola) Roll Degree from image as recorded by N3 via the Linux
    terminal with ExifTool
fRollDeg = subprocess.Popen(["exiftool -b -FlightRollDegree " + directory + "/" +
    filename],
                            shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Roll Degree to data type string
fRollDeg = fRollDeg.decode("utf-8")
# Convert from string to float
fRollDeg = float(fRollDeg)

# Extract Flight (Gondola) Yaw Degree from image as recorded by N3 via the Linux
    terminal with ExifTool
fYawDeg = subprocess.Popen(["exiftool -b -FlightYawDegree " + directory + "/" + filename
    ],
                           shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Yaw Degree to data type string
fYawDeg = fYawDeg.decode("utf-8")
# Convert from string to float
fYawDeg = float(fYawDeg)
```

```
# Extract Flight (Gondola) Pitch Degree from image as recorded by N3 via the Linux
    terminal with ExifTool
fPitchDeg = subprocess.Popen(["exiftool -b -FlightPitchDegree " + directory + "/" +
    filename],
                             shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Pitch Degree to data type string
fPitchDeg = fPitchDeg.decode("utf-8")
# Convert from string to float
fPitchDeg = float(fPitchDeg)

#
    ##########################################################################################################################

# Filtering Parameters for GPS georeferencing:
# If Gondola Roll > +/- 45 deg (since camera is self stabilized, roll should be minimal)
# If Gondola tilt (fPitchDeg) is > +45 degrees or < -135 degrees (as per mechanical
    range of
#  Zenmuse XT: https://www.dji.com/zenmuse-xt/info) This can affect the self
    stabilization of the camera
# If latitude or longitude = 0 degrees, Longitude > 180 degrees or
#  Longitude < 180 degrees, Latitude > 90 degrees or Latitude < 90 degrees
# If Camera Gimbal pitch is >= to 0 degrees (center of the image), GPS georeferencing
#  will not work as the camera line of sight will extend to the sky
if fRollDeg > 45 or fRollDeg < -45 or fPitchDeg > 45 or fPitchDeg < -135 or gPitchDeg >=
     0:
    continue
elif Latitude <= 0 or Latitude > 90 or Latitude < -90:
    continue
elif Longitude == 0 or Longitude > 180 or Longitude < -180:
    continue

# If the gimbal pitch plus half of the vertical field of view is <= -76 degrees, then
    skip the image
# If this was not included, the bottom of the image could theoretically be
#  positioned behind the camera which could complicate calculations
if gPitchDeg <= -76:
    continue

# If gimbal pitch is greater than 2 degrees, skip image
if gPitchDeg > -2:
    continue

#
    ##########################################################################################################################

# Parameters for temperature calculation
# For all Planck Constants below, reference Martiny et al. 1996,
#  "In Situ Calibration for Quantitative Infrared Thermography":
#  http://qirt.gel.ulaval.ca/archives/qirt1996/papers/001.pdf
# Also reference FLIR Systems, Installation manual: FLIR A3XX and FLIR A6XX series,
#  2010: http://91.143.108.245/Downloads/Flir/Dokumentation/T559498$a461_Manual.pdf
```

```
# Get Planck R1 constant from image metadata with ExifTool via Linux terminal
R1 = subprocess.Popen(["exiftool -b -PlanckR1 " + directory + "/" + filename],
                      shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Planck R1 constant to data type string
R1 = R1.decode("utf-8")
# Convert from string to float
R1 = float(R1)

# Get Planck R2 constant from image metadata with ExifTool via Linux terminal
R2 = subprocess.Popen(["exiftool -b -PlanckR2 " + directory + "/" + filename],
                      shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Planck R2 constant to data type string
R2 = R2.decode("utf-8")
# Convert from string to float
R2 = float(R2)

# Get Planck B constant from image metadata with ExifTool via Linux terminal
B = subprocess.Popen(["exiftool -b -PlanckB " + directory + "/" + filename],
                     shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Planck B constant to data type string
B = B.decode("utf-8")
# Convert from string to float
B = float(B)

# Get Planck O constant from image metadata with ExifTool via Linux terminal
planck_O = subprocess.Popen(["exiftool -b -PlanckO " + directory + "/" + filename],
                            shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Planck O constant to data type string
planck_O = planck_O.decode("utf-8")
# Convert from string to float
planck_O = float(planck_O)

# Get Planck F constant from image metadata with ExifTool via Linux terminal
F = subprocess.Popen(["exiftool -b -PlanckF " + directory + "/" + filename],
                     shell=True, stdout=subprocess.PIPE).communicate()[0]
# Decode Planck F constant to data type string
F = F.decode("utf-8")
# Convert from string to float
F = float(F)

#
    ##########################################################################################################################

# The next few lines is for TriSonica Altitude calculations to derive TANAB2 altitude
    above ground level
# Need to call in the date from each picture and convert to day of year
#  (doy) in minutes, then the hours in minutes and add the minutes
# Next,find the closest doy in TriSonica doy, return the index with the closest
#  value to identify the altitude of TANAB2.

# Get date and time when pictures were taken
# If the following variables do not exist as a local variable,
```

```
#  then initialize them (this only occurs for the first image)
if 'Year' not in locals():
    Year = numpy.empty(numFiles, dtype=object)
    Month = numpy.empty(numFiles, dtype=object)
    Days = numpy.empty(numFiles, dtype=object)
    Hour = numpy.empty(numFiles, dtype=object)
    Minutes = numpy.empty(numFiles, dtype=object)

# Get date and time from images via the Linux terminal with ExifTool
dates = subprocess.Popen(["exiftool -b -DateTimeOriginal " + directory + "/" + filename
    ],
                         shell=True, stdout=subprocess.PIPE).communicate()[0]
# Convert dates to data type string
dates = str(dates)
# Slice string to only include date and time
print('The Image Date and Time is: '+str(dates[2:21]))
# Date & time as 'YYYY:MM:DD HH:MM:SS' format
dates = dates[2:21]
# Convert date format to date time from string
dates = datetime.datetime.strptime(dates, "%Y:%m:%d %H:%M:%S")

# Separate image date and time into variables and change data type from datetime to
    string
yr = str(dates.year)
mnth = str(dates.month)
day = str(dates.day)
hr = str(dates.hour)
minute = str(dates.minute)

#
    ##########################################################################################################################

# Assign the Base Altitude (elevation above sea level) for each TANAB2 launch location
#  (MFT, Berm, and Mine for May 2018 campaign)
# Estimated elevations from Google Earth,

# Had to convert GPS coordinates to decimal degrees from Degrees Minutes Seconds via:
#  https://www.latlong.net/degrees-minutes-seconds-to-decimal-degrees,
#  also: https://www.fcc.gov/media/radio/dms-decimal
# Latitude of TANAB2 MFT Launch
BaseLatMFT = XX.XXXXXXXX
# Longitude of TANAB2 MFT Launch
BaseLonMFT = -XXX.XXXXXXXX
# Elevation of land above sea level of TANAB2 MFT Launch
BaseAltMFT = 398 # [m]

# Latitude of TANAB2 Berm Launch
BaseLatBerm = XX.XXXXXXXX
# Longitude of TANAB2 Berm Launch
BaseLonBerm = -XXX.XXXXXXXX
# Elevation of land above sea level of TANAB2 Berm Launch
BaseAltBerm = 337 # [m]
```

```
# Latitude of TANAB2 Mine Launch
BaseLatMine = XX.XXXXXXXX
# Longitude of TANAB2 Mine Launch
BaseLonMine = -XXX.XXXXXXXX
# Elevation of land above sea level of TANAB2 Mine Launch
BaseAltMine = 317 # [m]

# Assign all Base Latitude/Longitude/Altitude into a corresponding array for simplicity
BaseLat = [BaseLatMFT, BaseLatBerm, BaseLatMine]

# Change BaseLat from type list to type numpy array
BaseLat = numpy.asarray(BaseLat)
BaseLon = [BaseLonMFT, BaseLonBerm, BaseLonMine]

# Change BaseLon from type list to type numpy array
BaseLon = numpy.asarray(BaseLon)
BaseAlt = [BaseAltMFT, BaseAltBerm, BaseAltMine]

# Initialize array for distance between two geographic locations
# For each image, reinitialize arrays
# The goal is the identify where the TANAB2 was launched from (MFT, Berm, or Mine)
distance_LaunchSite_TANAB = numpy.zeros(3)

# Equatorial radius Radius of earth in km: https://nssdc.gsfc.nasa.gov/planetary/
    factsheet/earthfact.html
Radius_Earth = 6378.1

# To find distance between two pairs of latitudes and longitudes, use the Haversine
    function
for i in range(0, len(BaseLat)):
    # The geographic location of the TANAB2 launch location [radians]
    launchSite_lat_rads = math.radians(BaseLat[i])
    launchSite_lon_rads = math.radians(BaseLon[i])

    # The geographic location of the gondola/camera [radians]
    FLIR_image_lat_rads = math.radians(Latitude)
    FLIR_image_lon_rads = math.radians(Longitude)

    # Calculate the distance between the latitude and the longitude between the two
        coordinates
    delta_lat_rads = FLIR_image_lat_rads - launchSite_lat_rads
    delta_lon_rads = FLIR_image_lon_rads - launchSite_lon_rads

    # Assign two variables to simply haversine formula
    a_launch = math.sin(delta_lat_rads / 2)**2 + cos(launchSite_lat_rads) * \
               cos(FLIR_image_lat_rads) * sin(delta_lon_rads / 2)**2
    c_launch = 2 * math.atan2(sqrt(a_launch), sqrt(1 - a_launch))

    distance_LaunchSite_TANAB[i] = c_launch * Radius_Earth

# Find smallest value in distance and assign BaseAltitude of launch to each image
```

```
    BaseAlt_idx = distance_LaunchSite_TANAB.tolist().index(min(distance_LaunchSite_TANAB))

    # Initialize BaseAltitude variable as a type float variable
    BaseAltitude = 0.00000

    # If the minimum index returned was 0, the TANAB2 was launched at MFT, therefore assign
    #  the BaseAltitude as MFT BaseAlt above sea level
    if BaseAlt_idx == 0:
        BaseAltitude = BaseAlt[0]
    # If the minimum index returned was 1, the TANAB2 was launched at the Berm, therefore
        assign
    #  the BaseAltitude as Berm BaseAlt above sea level
    elif BaseAlt_idx == 1:
        BaseAltitude = BaseAlt[1]
    # If the minimum index returned was 2, the TANAB2 was launched at the Mine, therefore
        assign
    #  the BaseAltitude as Mine BaseAlt above sea level
    elif BaseAlt_idx == 2:
        BaseAltitude = BaseAlt[2]
    else:
        print('Something is wrong with the code above')
    print('The Base Altitude above sea level for the TANAB launch site is: '+str(
        BaseAltitude)+' [m] ')


#
    ##########################################################################################################################

    # Call in averaged data extracted from TriSonica
    #  (manual copying/pasting was completed to created the compiled averaged file.)

    # As of Sept 20/2018: includes average temperature from balloon flights and fix for
        second averaged
    #  data where the 0 second was averaged with the 59 second.
    trisonica_avg_fileName = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/TriSonica/' \
                             'TriSonica_May2018_Altitudes_averaged.txt'

    # Call in TriSonica averaged data
    trisonica_avg = numpy.genfromtxt(trisonica_avg_fileName, usecols=[5,7,13])
    # Call in second averaged data column (A data value is available for every second)
    # TriSonica recorded data at 10Hz, averaged these values about each whole second
    trisonica_seconds = trisonica_avg[:,0]
    # Call in the previously calculated day of the year (from Jan. 1/2018) in minutes
    trisonica_doy = trisonica_avg[:,1]

    # Calculate the day of year in seconds (from Jan. 1/2018)
    # Initialize TriSonica doy in terms of seconds (soy)
    trisonica_soy = numpy.zeros(len(trisonica_doy))

    # Convert from minute_doy to second_doy
    for i in range(0, len(trisonica_seconds)):
```

```
    trisonica_soy[i] = (trisonica_doy[i]*60) + trisonica_seconds[i]

# Call in the TriSonica derived altitude from pressures. The altitude is relative to the
    land surface
#  (Base Altitude must be added to get altitude above sea level)
trisonica_altitude = trisonica_avg[:,2]

# Day/hour/minute in May when picture was taken (some variables may be redundant as the
    day,
#  hour and minute from each image was assigned to variables above)
month_picture = dates.month
day_picture = dates.day
hour_picture = dates.hour
minute_picture = dates.minute
seconds_picture = dates.second

# Initialize doy seconds for pictures
doy_seconds_picture = 0

# Convert day/hour/minute into doy seconds based on the day the picture was recorded (
    day_picture)
# Assuming month of May in 2018
if day_picture == 5:
    doy_seconds_picture = (((125*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 6:
    doy_seconds_picture = (((126*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 7:
    doy_seconds_picture = (((127*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 8:
    doy_seconds_picture = (((128*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 9:
    doy_seconds_picture = (((129*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 10:
    doy_seconds_picture = (((130*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 11:
    doy_seconds_picture = (((131*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 12:
    doy_seconds_picture = (((132*24*60)+(hour_picture*60)+minute_picture)*60)+
```

```
        seconds_picture

elif day_picture == 13:
    doy_seconds_picture = (((133*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 14:
    doy_seconds_picture = (((134*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 15:
    doy_seconds_picture = (((135*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 16:
    doy_seconds_picture = (((136*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 17:
    doy_seconds_picture = (((137*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 18:
    doy_seconds_picture = (((138*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 19:
    doy_seconds_picture = (((139*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

# May 21st was the day the TriSonica did not work properly,,, it stopped around noon.
#  Extract absolute altitude from the image metadata with ExifTool and
#  subtract from Mine Base Alt (This operation is coded below)
elif day_picture == 21:
     print('Altitude will be dealt with via extracting absolute alt from image'
           '& calc the absolute of (image Alt - Mine Base Alt)')

elif day_picture == 23:
    doy_seconds_picture = (((143*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 24:
    doy_seconds_picture = (((144*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 26:
    doy_seconds_picture = (((146*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 27:
    doy_seconds_picture = (((147*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture
```

```
elif day_picture == 30:
    doy_seconds_picture = (((150*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

elif day_picture == 31:
    doy_seconds_picture = (((151*24*60)+(hour_picture*60)+minute_picture)*60)+
        seconds_picture

else:
    print('More Dates need to be included above')

# Initialize delta doy in seconds (soy) variable
#  (difference between image capture time and TriSonica data time index)
delta_soy = numpy.zeros(len(trisonica_doy))

# Fix for the day the TriSonica stopped measuring, Use Altitude recorded by N3 and saved
     in image metadata
if day_picture == 21:
    # Call in Positioning Data
    # GPS Altitude (meters above sea level (msl))
    Altitude = subprocess.Popen(["exiftool -b -GPSAltitude " + directory + "/" +
        filename],
                                shell=True, stdout=subprocess.PIPE).communicate()[0]
    # Decode Altitude to data type string
    Altitude = Altitude.decode("utf-8")
    # Convert from string to float
    Altitude = float(Altitude)

    # Subtract Base Altitude from absolute altitude
    Altitude_AGL = abs(Altitude - BaseAltMine)

# Match DOY_second indices from second averaged file & each individual image
# Loop through averaged doy indices from TriSonica data

for i in range(0, len(trisonica_soy)):
    # If the image is dated May 21, break loop as TriSonica
    #  data is not available on this date (use N3 data instead)
    if day_picture == 21:
        break
    else:
        # Loop through Base Altitude indices
        # Calculate difference between doy times
        #  (each individual image and each second averaged TriSonica index
        delta_soy[i] = abs(doy_seconds_picture - trisonica_soy[i])

        # When at the last value of the TriSonica averaged file,
        #  find index of minimum value and write corresponding BaseAlt to the Altitude
            variable
        # Also record the index of the Base Altitude file
        if i == (len(trisonica_soy) - 1):
            Altitude_AGL = trisonica_altitude[numpy.argmin(delta_soy)]
```

```
print('The Altitude above the TANAB launch location is: '+str(Altitude_AGL)+' [m]')

#
    ##########################################################################################################################

# Set up variables that are used within the georeferencing calculations below

# Reference this image for Yaw, Roll and Pitch frame of
#  references: http://blog-gh4-france.over-blog.com/2015/12/test-du-dji-ronin-m-sur-le-
    gh4.html
# Yaw of gimbal is calibrated to the True North, it is not necessary to add the flight
    yaw angle
Yaw = gYawDeg

# Assume the gimbal roll is zero (cannot be controlled)
Roll = fRollDeg

# Pitch and Roll angles are independent of each other, i.e. if one changes
#  (ex. the flight parameter), it will directly affect the gimbal but it will
#  not be accounted for by the gimbal pitch/roll variable
# Positive upward from horizontal; usually negative for lines of sight below horizontal
Pitch = gPitchDeg

# Field of View angles in degrees for the 19 mm Zenmuse XT thermal
#  camera (Specifications: https://www.dji.com/ca/zenmuse-xt/specs)
# Vertical Field of View [degree]
FoVV = 26
# Horizontal Field of View [degree]
FoVH = 32

# When the camera lens is exactly perpendicular to the ground (pointed towards to
    horizon),
#  the resulting pitch angle for the center of an image is 0 degrees
# If the camera is tilted towards the ground, the pitch angle is negative,
#  if camera tiled upwards, the pitch angle is positive

# References for mechanical rotational limits of the Zenmuse XT: https://www.dji.com/
    zenmuse-xt/info
# Find GPS coordinates for middle (tilt_center),
#  top middle (tilt_top), and bottom middle (tilt_bottom) of each image
# These three values are all related to the gimbal pitch angle from the image metadata
#  and the physical vertical field of view for the 19mm lens camera
# Tilt angle [degree]
tilt_center = Pitch
tilt_bottom = Pitch-(FoVV/2)
tilt_top = Pitch+(FoVV/2)

# Need to check if the sky will be in the image. Images with a top tilt
#  angle will have their tilt angle adjusted to include pixels below an assumed pixel
#  row which is a direct function of an assumed tilt angle
# The center tilt angle does not need to be considered as images
```

```
#  with a gimbal pitch angle >= 0 were already skipped over in the code near the top of
    the script

# If top of the image has a tilt angle >= 0 then, assign an assumed tilt angle is
    assigned
#  to omit pixels that may contain the sky (i.e. above horizontal)
if tilt_top >= -1:
    print('The tilt angle for the top of the image is: ' + str(tilt_top))
    # Change tilt angle so it is equal to an arbitrary tilt angle. -1 deg was assumed
        and can be changed.
    tilt_top = -1

print('The top tilt angle (deg) for the image is: ' + str(tilt_top))
print('The center tilt angle (deg) for the image is: ' + str(tilt_center))
print('The bottom tilt angle (deg) for the image is: ' + str(tilt_bottom))

#
    ##########################################################################################################

# Used this section of code to implement a sensitivity analysis. Can be revisited if
    required.

# Apply an offset to each Pitch angle used during the sensitivity analysis as of Dec.
    5/2018.
# December 17, 2018: After completing a few tests, the developed method does not need
    the
#  tilt angle adjusted, therefore set theta offset as 0
thetaOffset = 0

theta_top_degrees = tilt_top-thetaOffset
theta_center_degrees = tilt_center-thetaOffset
theta_bottom_degrees = tilt_bottom-thetaOffset

#
    ##########################################################################################################

# Convert theta angles to radians. All trigonometric functions in Python assume that
    angles are in radians
theta_top_rads = math.radians(theta_top_degrees)
theta_center_rads = math.radians(theta_center_degrees)
theta_bottom_rads = math.radians(theta_bottom_degrees)

# Note: If heading is 0 deg or 360 deg = north exactly, if 90 deg =  east, 180 deg =
    south, 270 deg = west
# Set Heading variable for georeferencing calculations
heading = Yaw

# Adjust heading if less than 0 degrees, add 360 degrees, so the angles will always be
    positive
if heading < 0:
    heading = heading + 360
```

```
# Convert Latitude/Longitude/Yaw to radians for georeferencing calculations
Yaw_rads = math.radians(heading)
Latitude_rads = math.radians(Latitude)
Longitude_rads = math.radians(Longitude)

# Call the LandSlopeEquations Function
# Function fits a polynomial relating the elevation above sea
#  level for the land in the 8 cardinal directions at each TANAB2 launch site to
#  a distance away from the launch site (up to 10km away from the origin of each TANAB2
    launch)
ground_elev_ASL_fitted, ground_elev_AGL = LandSlopeEquations(BaseAltitude, heading)

# Calculate latitude/longitude of top/virtual top of image
# Find coefficients for line of sight from camera that intersects with the ground
line_of_sight = numpy.zeros(4)
# Find slope of the line associated with line of sight which is negative because
    theta_top is negative
line_of_sight[2] = math.tan(theta_top_rads)
# Find altitude of the TANAB2 from the ground.
#  This is the y-intercept of the line of sight with respect to origin located at ground
     level
line_of_sight[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000,
#  the number of indices to use is equal to the length of ground_elev_AGL
# (as calculated in LandSlopeEquations function) (given information,
# the numerical difference between each value is calculated by the function
# x_test represents distance in meters away from the TANAB2 launch site
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with
#  respect to the detrended land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended
#  surface elevation with respect to distance away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test,ground_elev_AGL,3)

# Find the roots and select the smallest positive value to be the horizontal
#  distance away from the TANAB2 launch site
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Top = numpy.roots(coefficients_of_intersection)
print('The roots are: ' + str(Roots_Top))
print('The altitude of the balloon above grade level is: '+str(line_of_sight[3]))
print('The Tilt angle is: '+str(math.degrees(theta_top_degrees)))

# Create an array for the roots that are real numbers (not complex)
real_roots_top = numpy.empty(3)
real_roots_top[:] = numpy.nan

# Check for roots that are not complex numbers and write real roots to array
for i in range(0, len(Roots_Top)):
    if numpy.iscomplex(Roots_Top[i]) == False:
```

```
        real_roots_top[i] = Roots_Top[i]

print('The Real Roots for the Top Center are: '+str(real_roots_top))

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Top)):
    if real_roots_top[non_real_root] < 0 or numpy.isnan(real_roots_top[non_real_root])
        == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Top):
    continue

# Choose the root that is the smallest real positive solution to be the distance away
#  from the TANAB2 launch location to
# the top of the projected image on the land surface
d_top = min(i for i in real_roots_top if i > 0)
# Convert d_top to km
d_top_km = d_top/1000

# Get the latitude/longitude for the top, center and bottom of each image via the
    variation of the Haversine Formula
# Formulas for latitude/longitude are from:
#  https://www.movable-type.co.uk/scripts/latlong.html, as of Aug.17/2018, completed a
    test to
#  ensure that these formulas are correct
# Calculate latitude/longitude (lat2/lon2) for top center pixel of the image

# Note: Ensure all angles are in radians before using a trigonometric function
lat2_top = asin(sin(Latitude_rads)*cos(d_top_km/Radius_Earth)+
                cos(Latitude_rads)*sin(d_top_km/Radius_Earth)*cos(Yaw_rads))

lon2_top = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_top_km/Radius_Earth)*cos(
    Latitude_rads),
     cos(d_top_km/Radius_Earth)-sin(Latitude_rads)*sin(lat2_top))

# Convert coordinates to decimal degrees
lat2_top = math.degrees(lat2_top)
lon2_top = math.degrees(lon2_top)

#
    ##########################################################################################################################

# Calculate latitude/longitude for the center of the image
# Find coefficients for line of sight from camera that intersects with the ground
line_of_sight = numpy.zeros(4)
line_of_sight[2] = math.tan(theta_center_rads)
line_of_sight[3] = Altitude_AGL
```

```
# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000,
#  the number of indices to use is equal to the length of ground_elev_AGL
#  (as calculated in LandSlopeEquations function)
#  (given information, the numerical difference between each value is calculated by the
    function
# x_test represents distance in meters away from the TANAB2 launch
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with
#  respect to the detrended land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended
#  surface elevation with respect to distance away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test, ground_elev_AGL, 3)

# Find the roots and select the smallest positive value to be the horizontal
#  distance away from the TANAB2 launch site
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Center = numpy.roots(coefficients_of_intersection)
print('The roots are: ' + str(Roots_Center))
print('The altitude of the balloon is: ' + str(line_of_sight[3]))
print('The Tilt angle is: ' + str(math.degrees(theta_center_rads)))

# Create an array for the roots that are real numbers (not complex)
real_roots_center = numpy.empty(3)
real_roots_center[:] = numpy.nan
for i in range(0, len(Roots_Center)):
    if numpy.iscomplex(Roots_Center[i]) == False:
        real_roots_center[i] = Roots_Center[i]

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Center)):
    if real_roots_center[non_real_root] <= 0 or numpy.isnan(real_roots_center[
        non_real_root]) == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Center):
    continue

# Choose the root that is the smallest real positive solution to be the distance
#  away from the TANAB2 launch location to the center of the projected image on the land
     surface
d_center = min(i for i in real_roots_center if i > 0)

# Convert d_center to km
d_center_km = d_center / 1000
```

```
# Formulas for latitude/longitude are from:
#  https://www.movable-type.co.uk/scripts/latlong.html, as of Aug.17/2018, completed a
    test to
#  ensure that these formulas are correct
# Calculate latitude/longitude (lat2/lon2) for the middle center of the image

# Note: Ensure all angles are in radians before using a trig function
lat2_center = asin(sin(Latitude_rads)*cos(d_center_km/Radius_Earth)+
                   cos(Latitude_rads)*sin(d_center_km/Radius_Earth)*cos(Yaw_rads))

lon2_center = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_center_km/Radius_Earth)*cos(
    Latitude_rads),
                                     cos(d_center_km/Radius_Earth)-sin(Latitude_rads)*sin(
                                         lat2_center))

# Convert back to decimal degrees
lat2_center = math.degrees(lat2_center)
lon2_center = math.degrees(lon2_center)

#
    ##########################################################################################################################################

# Calculate latitude/longitude for the center of the image
# Find coefficients for line of sight from camera that intersects with the ground
line_of_sight = numpy.zeros(4)
line_of_sight[2] = math.tan(theta_bottom_rads)
line_of_sight[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000,
#  the number of indices to use is equal to the length of ground_elev_AGL
#  (as calculated in LandSlopeEquations function)
#  (given information, the numerical difference between each value is calculated by the
    function
# x_test represents distance in meters away from the TANAB2 launch site
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with
#  respect to the detrended land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended
#  surface elevation with respect to distance away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test, ground_elev_AGL, 3)

# Find the roots and select the smallest positive value to be the horizontal
#  distance away from the TANAB2 launch site
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Bottom = numpy.roots(coefficients_of_intersection)

print('The roots are: ' + str(Roots_Bottom))
print('The altitude of the balloon is: ' + str(line_of_sight[3]))
print('The Tilt angle is: ' + str(math.degrees(theta_bottom_rads)))
```

```
# Create an array for the roots that are real numbers (not complex)
real_roots_bottom = numpy.empty(3)
real_roots_bottom[:] = numpy.nan
for i in range(0, len(Roots_Bottom)):
    if numpy.iscomplex(Roots_Bottom[i]) == False:
        real_roots_bottom[i] = Roots_Bottom[i]

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Bottom)):
    if real_roots_bottom[non_real_root] < 0 or numpy.isnan(real_roots_bottom[
        non_real_root]) == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Bottom):
    continue

# Choose the root that is the smallest real positive solution to be the distance
# from the TANAB2 launch location to the bottom of the projected image on the land
    surface
d_bottom = min(i for i in real_roots_bottom if i > 0)
# Convert d_bottom to km
d_bottom_km = d_bottom / 1000

# Formulas for latitude/longitude are from: https://www.movable-type.co.uk/scripts/
    latlong.html)
# Calculate latitude/longitude (lat2/lon2) for the center bottom pixel of the image
# Note: Ensure all angles are in radians before using a trigonometric function
lat2_bottom = asin(sin(Latitude_rads)*cos(d_bottom_km/Radius_Earth)+
                   cos(Latitude_rads)*sin(d_bottom_km/Radius_Earth)*cos(Yaw_rads))

lon2_bottom = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_bottom_km/Radius_Earth)*cos(
    Latitude_rads),
                                     cos(d_bottom_km/Radius_Earth)-sin(Latitude_rads)*sin(
                                         lat2_bottom))


# Convert back to decimal degrees
lat2_bottom = math.degrees(lat2_bottom)
lon2_bottom = math.degrees(lon2_bottom)

# Print Results
print('The Altitude of the balloon with respect to grade level is: '+str(Altitude_AGL)+'
    \n')
print('The Origin lat is: '+str(Latitude))
print('The Origin lon is: '+str(Longitude)+'\n')
print('The lat_top is: ' + str(lat2_top))
print('The lon_top is: ' + str(lon2_top) + '\n')
print('The lat2_center is: ' + str(lat2_center))
```

```
print('The lon2_center is: ' + str(lon2_center) + '\n')
print('The lat2_bottom is: ' + str(lat2_bottom))
print('The lon2_bottom is: ' + str(lon2_bottom) + '\n')
print('The d_center is: '+str(d_center))
print('The d_bottom is: '+str(d_bottom))
print('The d_top is: '+str(d_top))

#
    ##########################################################################################################################

# Calculate GPS coordinates for pixels along the edge/corners of the image
# Find the top right and top left latitude/longitude for each image

# Find the geographic distance in km for both the top right and left of each image
d_geographic_top_km = d_top_km/cos(math.radians(FoVH / 2))

# For pixels on the left edge of the image
Yaw_left_rads = math.radians(heading - (FoVH / 2))

# Ensure this angle is strictly positive
if Yaw_left_rads < 0:
    Yaw_left_rads = Yaw_left_rads + 2 * numpy.pi

# For pixels on the right edge of the image
Yaw_right_rads = math.radians(heading + (FoVH / 2))

# Ensure this angle is strictly less than 360 degrees
if Yaw_right_rads > 2 * numpy.pi:
    Yaw_right_rads = Yaw_right_rads - 2 * numpy.pi

# Find the latitude/longitude for the top left pixel of the image
lat2_top_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_top_km / Radius_Earth) +
                          cos(Latitude_rads) * sin(d_geographic_top_km / Radius_Earth) *
                              cos(Yaw_left_rads))

lon2_top_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads)
                                            * sin(d_geographic_top_km / Radius_Earth) *
                                            cos(Latitude_rads), cos(d_geographic_top_km
                                                / Radius_Earth)
                                            - sin(Latitude_rads) * sin(
                                                lat2_top_left_rads))

# Convert back to decimal degrees
lat2_top_left = math.degrees(lat2_top_left_rads)
lon2_top_left = math.degrees(lon2_top_left_rads)

# Find the latitude/longitude for the top right pixel of the image
lat2_top_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_top_km / Radius_Earth)
    + cos(Latitude_rads)
                           * sin(d_geographic_top_km / Radius_Earth) * cos(
                               Yaw_right_rads))
```

```
lon2_top_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads)
                                             * sin(d_geographic_top_km / Radius_Earth) *
                                                 cos(Latitude_rads),
                                             cos(d_geographic_top_km / Radius_Earth)
                                             - sin(Latitude_rads) * sin(
                                                 lat2_top_right_rads))

# Convert back to decimal degrees
lat2_top_right = math.degrees(lat2_top_right_rads)
lon2_top_right = math.degrees(lon2_top_right_rads)

#
    ##########################################################################################################################

# Find the geographic distance in km for both the center right and left of each image
d_geographic_center_km = d_center_km / cos(math.radians(FoVH / 2))

# Find the latitude/longitude for the center left pixel of the image
lat2_center_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_center_km /
    Radius_Earth) + cos(Latitude_rads)
                             * sin(d_geographic_center_km / Radius_Earth) * cos(
                                 Yaw_left_rads))

lon2_center_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads)
                                               * sin(d_geographic_center_km /
                                                   Radius_Earth) * cos(Latitude_rads),
                                               cos(d_geographic_center_km / Radius_Earth
                                                   )
                                               - sin(Latitude_rads) * sin(
                                                   lat2_center_left_rads))

# Convert back to decimal degrees
lat2_center_left = math.degrees(lat2_center_left_rads)
lon2_center_left = math.degrees(lon2_center_left_rads)

# Find the latitude/longitude for the center right pixel of the image
lat2_center_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_center_km /
    Radius_Earth) + cos(Latitude_rads)
                              * sin(d_geographic_center_km / Radius_Earth) * cos(
                                  Yaw_right_rads))

lon2_center_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads)
                                                * sin(d_geographic_center_km /
                                                    Radius_Earth) * cos(Latitude_rads),
                                                cos(d_geographic_center_km /
                                                    Radius_Earth)
                                                - sin(Latitude_rads) * sin(
                                                    lat2_center_right_rads))

# Convert back to decimal degrees
lat2_center_right = math.degrees(lat2_center_right_rads)
lon2_center_right = math.degrees(lon2_center_right_rads)
```

```
#
    ##########################################################################################################################

# Find the geographic distance in km for both the bottom right and left of each image
d_geographic_bottom_km = d_bottom_km / cos(math.radians(FoVH / 2))

# Find the latitude/longitude for the bottom left pixel of the image
lat2_bottom_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_bottom_km /
    Radius_Earth) + cos(Latitude_rads)
                             * sin(d_geographic_bottom_km / Radius_Earth) * cos(
                                 Yaw_left_rads))

lon2_bottom_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads)
                                               * sin(d_geographic_bottom_km /
                                                   Radius_Earth) * cos(Latitude_rads),
                                               cos(d_geographic_bottom_km / Radius_Earth
                                                   )
                                               - sin(Latitude_rads) * sin(
                                                   lat2_bottom_left_rads))

# Convert back to decimal degrees
lat2_bottom_left = math.degrees(lat2_bottom_left_rads)
lon2_bottom_left = math.degrees(lon2_bottom_left_rads)

# Find the latitude/longitude for the bottom right pixel of the image
lat2_bottom_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_bottom_km /
    Radius_Earth) + cos(Latitude_rads)
                              * sin(d_geographic_bottom_km / Radius_Earth) * cos(
                                  Yaw_right_rads))

lon2_bottom_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads)
                                                * sin(d_geographic_bottom_km /
                                                    Radius_Earth) * cos(Latitude_rads),
                                                cos(d_geographic_bottom_km /
                                                    Radius_Earth)
                                                - sin(Latitude_rads) * sin(
                                                    lat2_bottom_right_rads))

# Convert back to decimal degrees
lat2_bottom_right = math.degrees(lat2_bottom_right_rads)
lon2_bottom_right = math.degrees(lon2_bottom_right_rads)

# Print calculated geographic coordinates
print('The lat2_top_left is: ' + str(lat2_top_left))
print('The lon2_top_left is: ' + str(lon2_top_left) + '\n')
print('The lat2_top_right is: ' + str(lat2_top_right))
print('The lon2_top_right is: ' + str(lon2_top_right) + '\n')

print('The lat2_center_left is: ' + str(lat2_center_left))
print('The lon2_center_left is: ' + str(lon2_center_left) + '\n')
print('The lat2_center_right is: ' + str(lat2_center_right))
```

```
print('The lon2_center_right is: ' + str(lon2_center_right) + '\n')

print('The lat2_bottom_left is: ' + str(lat2_bottom_left))
print('The lon2_bottom_left is: ' + str(lon2_bottom_left) + '\n')
print('The lat2_bottom_right is: ' + str(lat2_bottom_right))
print('The lon2_bottom_right is: ' + str(lon2_bottom_right) + '\n')

#
    ##########################################################################################################################

# Maximum pixel width and height of each image based on the DJI 19 mm lens Zenmuse XT
x_pixel_range = 640
y_pixel_range = 512

# If the tilt angle for the top of the image is greater than zero and an assumed top
    tilt angle was assigned,
# Calculate the new pixel top row, only if the top tilt angle is > or = to -1 deg
if tilt_top >= -1:
    # Get relation between trigonometric angles and pixels
    # See created figures in thesis for visual reference
    # The following variables are all in degrees
    # Note: d_top was already calculated by assuming a new top which looked down
    #  -1 degrees below the horizon
    gamma = math.degrees(math.atan(d_top/Altitude_AGL))
    beta = 90 - abs(Pitch) + (FoVV/2) - gamma
    kappa = 90-(FoVV/2)
    eta = 90-(FoVV/2)+beta

    y_pixel_top = int(((y_pixel_range/2)*sin(radians(beta)))/(sin(radians(FoVV/2))*sin(
        radians(180-eta))))

    print('The top of the image is located at: '+str(y_pixel_top))

# The value to divide the horizontal and vertical pixel resolution by
# Used to calculate the maximum pixel step for the horizontal direction
delta_x_pixel = 10
delta_y_pixel = 8

# Returns a maximum pixel step of 64 columns/rows based on
#  the 19mm-Zenmuse XT image horizontal and vertical resolution
x_max_step = int(x_pixel_range/delta_x_pixel)
y_max_step = int(y_pixel_range/delta_y_pixel)

# Initialize matrix for singular image for pixels.
#  Including: x/y pixel coordinates, latitude, longitude, temperature, and set values to
     Nan

# Note: The amount of data retreived per image should be less than the full image
#  as a result, the size of image_matrix could likely be optimized
image_matrix = numpy.zeros((x_pixel_range, y_pixel_range, 11))
image_matrix[:] = numpy.nan
```

```
# Create an array for the filename for the corresponding pixel retrieved from each image
# This is required as this script does not process images from 'RawImages' in
    chronological order
filename_image = numpy.chararray((x_pixel_range, y_pixel_range,1), itemsize=12)

# Initialize matrix to save data from all pictures (and a separate variable for
    filenames)
# Check if variable exists in locals (do only for the first image)
if 'all_pixel_data_multi_image' not in locals():
    all_pixel_data_multi_image = numpy.zeros((x_max_step*y_max_step*numFiles, 11))
    all_pixel_data_multi_image[:] = numpy.nan

    filenames_total = numpy.chararray((x_max_step*y_max_step*numFiles, 1), itemsize=12)

# Create variables to save known coordinates to
# Size will depend on the number of files in the folder (numFiles)
# These variables are to contain the data for the center and edges of each image
# This section of code is only executed for the 1st image processed
if 'file_names_array' not in globals():
    # filenames
    file_names_array = numpy.chararray(numFiles*1, itemsize=12)
    file_names_array[:] = b''

    # TANAB2 launch location
    lat_TANAB_array = numpy.zeros((numFiles, 1))
    lon_TANAB_array = numpy.zeros((numFiles, 1))

    # Top left of image
    # Create arrays for the latitude, longitude, and pixel locations for the top left
        corner of the image
    tLeft_lat_array = numpy.zeros((numFiles, 1))
    tLeft_lon_array = numpy.zeros((numFiles, 1))
    tLeft_x_pixel_array = numpy.zeros((numFiles, 1))
    tLeft_y_pixel_array = numpy.zeros((numFiles, 1))

    # Top center of image
    # Create arrays for the latitude, longitude, and pixel locations for the top middle
        of the image
    tCenter_lat_array = numpy.zeros((numFiles, 1))
    tCenter_lon_array = numpy.zeros((numFiles, 1))
    tCenter_x_pixel_array = numpy.zeros((numFiles, 1))
    tCenter_y_pixel_array = numpy.zeros((numFiles, 1))

    # Top right of image
    # Create arrays for the latitude, longitude, and pixel locations for the top right
        corner of the image
    tRight_lat_array = numpy.zeros((numFiles, 1))
    tRight_lon_array = numpy.zeros((numFiles, 1))
    tRight_x_pixel_array = numpy.zeros((numFiles, 1))
    tRight_y_pixel_array = numpy.zeros((numFiles, 1))

    # Center left of image
```

```
    # Create arrays for the latitude, longitude, and pixel locations for the center left
        edge of the image
    cLeft_lat_array = numpy.zeros((numFiles, 1))
    cLeft_lon_array = numpy.zeros((numFiles, 1))
    cLeft_x_pixel_array = numpy.zeros((numFiles, 1))
    cLeft_y_pixel_array = numpy.zeros((numFiles, 1))

    # Center of image
    # Create arrays for the latitude, longitude, and pixel locations for the middle (
        center) of the image
    center_lat_array = numpy.zeros((numFiles, 1))
    center_lon_array = numpy.zeros((numFiles, 1))
    center_x_pixel_array = numpy.zeros((numFiles, 1))
    center_y_pixel_array = numpy.zeros((numFiles, 1))

    # Center right of image
    # Create arrays for the latitude, longitude, and pixel locations for the center
        right edge of the image
    cRight_lat_array = numpy.zeros((numFiles, 1))
    cRight_lon_array = numpy.zeros((numFiles, 1))
    cRight_x_pixel_array = numpy.zeros((numFiles, 1))
    cRight_y_pixel_array = numpy.zeros((numFiles, 1))

    # Bottom left of image
    # Create arrays for the latitude, longitude, and pixel locations for the bottom left
        corner of the image
    bLeft_lat_array = numpy.zeros((numFiles, 1))
    bLeft_lon_array = numpy.zeros((numFiles, 1))
    bLeft_x_pixel_array = numpy.zeros((numFiles, 1))
    bLeft_y_pixel_array = numpy.zeros((numFiles, 1))

    # Bottom center image
    # Create arrays for the latitude, longitude, and pixel locations for the bottom
        center of the image
    bCenter_lat_array = numpy.zeros((numFiles, 1))
    bCenter_lon_array = numpy.zeros((numFiles, 1))
    bCenter_x_pixel_array = numpy.zeros((numFiles, 1))
    bCenter_y_pixel_array = numpy.zeros((numFiles, 1))

    # Bottom right of image
    # Create arrays for the latitude, longitude, and pixel locations for the bottom
        right corner of the image
    bRight_lat_array = numpy.zeros((numFiles, 1))
    bRight_lon_array = numpy.zeros((numFiles, 1))
    bRight_x_pixel_array = numpy.zeros((numFiles, 1))
    bRight_y_pixel_array = numpy.zeros((numFiles, 1))

# Write filenames to array. Used in kml (Google Earth) save
for j in range(0, numFiles):
    if file_names_array[j] == '':
        file_names_array[j] = filename
        break
```

```
#
    ##########################################################################################################################

# Check if the top pixel is not at the top of the image
# If the gimbal pitch angle for the top of the image based on the recorded pitch plus
#  half of the FoVV is > 0 degrees, use the calculated new top pixel row as the "top" of
     the image
if tilt_top >= -1:
    v_pixel_top = y_pixel_top
else:
    v_pixel_top = 0

# Calculate and implement geometric step to determine how many vertical pixels to skip
    over when calculating ST
# The goal is to have higher resolution for steps at the top of the image
#  as pixel rows at the top of the image would result in a larger geographic distance
#  away from the TANAB2 as compared to pixel rows near the bottom of the image.
# Data associated with pixels at the top of the image should return ST maps further away
     from
#  the TANAB2 launch sites and result in a more even and possibly consist spatial ST map

# Initialize the pixel step array
y_pixel_step = numpy.zeros((10, 1))
# Identify the coefficient to use in the geometric pixel step formula
aStepGeometric = 18
# Identify the constant to use in the geometric pixel step calculation
rStepGeometric = 1.41

# Start at the top of the image (Row 0), if a new "top" is chosen,
#  a filtering loop below skips over any pixels in
#  y_pixel_step that are out of the calculated vertical pixel range.
y_pixel_step[0] = 0

# The second index in the geometric step function was selected to be row 18.
y_pixel_step[1] = aStepGeometric
for GeometricStep in range(2, 10):
    # Calculate the geometric pixel step for the 8 remaining pixels and save to the
        appropriate array
    y_pixel_step[GeometricStep] = int(aStepGeometric * ((rStepGeometric) **
        GeometricStep))

print('The␣virtual␣pixel␣top␣is:␣'+str(v_pixel_top))
print('The␣y_pixel_step␣is␣as␣follows:␣'+str(y_pixel_step))

#
    ##########################################################################################################################

# This nested loop chooses pixels based on the predetermined horizontal
#  pixel step and the calculated geometric pixels step
# Within the loop, each pixel is georeferenced with the derived
#  mathematical formulas between pixels and geographic distance
```

```
# ST is calculated based on recorded pixel signal values
# The outer loop represents the horizontal (column) pixel step
for i in range(0, x_pixel_range, x_max_step):
    print('The pixel column number being processed now is: '+str(i))
    # Initialize a counter variable to be used to correspond to the geometric step
        matrix index
    # Count must be -1 as y_pixel_step[0] = 0
    #  (if count = 0, y_pixel_step[1] = 18 and if a new "top" is NOT used, j MUST equal
        0!
    count = -1
    # The inner loop represents the vertical (row) pixel step
    for j in y_pixel_step:
        # Add one to the counter variable
        count += 1
        # Check to see if the chosen geometric pixel step value is less than the virtual
            pixel top,
        #  if it is, continue to next y_pixel_step value
        if y_pixel_step[count] < v_pixel_top:
            continue

        print('The pixel row number being processed now is: '+str(j))

        # Need to convert data type of pixel step to int
        j = int(j)

        # Print pixel location in image matrix
        print('x Pixel Location: '+str(i))
        print('y Pixel Location: '+str(j)+'\n')

        # Pixel to Geographic distance relationship
        # Find Slope for line of sight for each specific pixel coordinate from the
            camera to the ground
        # Need Beta_new (tilt angle of camera given known y pixel coordinate)
        # Must correlate pixels to latitude/longitude... need to calculate new beta
            given pixel coordinates
        #  From Sine Law solve for beta, Where kappa = 90-FoVV/2
        # Refer to diagrams of TANAB2 camera with respect to
        #  image projection on the Earth's surface for further clarification
        kappa = 90-FoVV/2

        # Go from pixels to distance, Using the sine law, rearrange for Beta.
        beta_new_rads = -1*atan(((0.5*y_pixel_range-j)*
                                 sin(math.radians(0.5*FoVV)))/(0.5*y_pixel_range*sin(
                                     math.radians(kappa))))\
                        +(math.radians(0.5*FoVV))

        # Convert beta_new to degrees
        beta_new = math.degrees(beta_new_rads)

        # Next get gamma_new. This is the angle away from the horizontal axis (zero
            degrees)
        # corresponding to the current pixel
```

```
gamma_new = 90-abs(Pitch)+(FoVV/2)-beta_new
gamma_new_rads = math.radians(gamma_new)

# Calculate the slope for line of sight through the current pixel from the
    camera.
slope = (1/tan(gamma_new_rads))*-1

# Call LandSlopeEquations Function
ground_elev_ASL_fitted, ground_elev_AGL = LandSlopeEquations(BaseAltitude,
    heading)

print('The slope for the line of sight from the camera to the ground'
      ' on the current pixel is: '+str(slope))

# Find coefficients for line of sight from camera that intersects with the
    ground
line_of_sight_pixel = numpy.zeros(4)
line_of_sight_pixel[2] = slope
line_of_sight_pixel[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000,
#  the number of indices to use is equal to the length of ground_elev_AGL
#  (as calculated in LandSlopeEquations function)
#  (given information, the numerical difference between each value is calculated
     by the function
# x_test represents distance in meters away from the TANAB2 launch site
x_test_pixel = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site
#  with respect to the detrended land surface elevation

# Return the coefficients for a 3rd order polynomial of the detrended
#  surface elevation with respect to distance away from the TANAB2 launch site
land_elevation_equation_pixel_AGL = numpy.polyfit(x_test_pixel, ground_elev_AGL,
    3)

# Find the roots and select the smallest positive value to be the horizontal
#  distance away from the TANAB2 launch site
intersections_pixel = land_elevation_equation_pixel_AGL - line_of_sight_pixel
Roots_new = numpy.roots(intersections_pixel)

print('The roots are: ' + str(Roots_new))

# Create an array for the roots that are real numbers (not complex)
real_roots_new = numpy.empty(3)
real_roots_new[:] = numpy.nan
for m in range(0, len(Roots_new)):
    if numpy.iscomplex(Roots_new[m]) == False:
        real_roots_new[m] = Roots_new[m]

# Initialize a variable that counts the number of non real roots
```

```
root_counter_new = 0

# Identify the number of non real roots
for non_real_root_new in range(0, len(Roots_new)):
    if real_roots_new[non_real_root_new] < 0 or numpy.isnan(real_roots_new[
        non_real_root_new]) == True:
        root_counter_new += 1

# If no real roots exist, then continue to the next image
if root_counter_new == len(Roots_new):
    continue
else:
    # Choose the root that is the smallest real positive solution to be the
        distance
    #  from the TANAB2 launch location
    d_pixel_proj_ctr = min(n for n in real_roots_new if n > 0)
    # Convert d_center to km
    d_pixel_proj_ctr_km = d_pixel_proj_ctr / 1000

    # Put check in for d_pixel. If > 100 km (too far), continue on to next y
        pixel
    if d_pixel_proj_ctr_km > 100:
        continue
    else:
        print('The horizontal geographic pixel distance as'
              ' projected on the center of the image is: '+str(
                  d_pixel_proj_ctr_km))

        # Get the alpha angle. The angle away from the geographic distance away
        #  from the TANAB2 and parallel to the camera line of sight
        # The alpha value is used to calculate the geographic distance away
        #  from the TANAB2 for pixels that are not parallel to the camera line
            of sight

        # Find the angle from the center line of the image given index i for the
             current pixel
        # This will change the "effective" yaw angle
        if i == 0:
            alpha = - FoVH/2
            alpha_rads = math.radians(alpha)
        elif (i > 0) and (i < x_pixel_range/2):
            alpha = - (x_pixel_range/2-i) * FoVH / (x_pixel_range)
            alpha_rads = math.radians(alpha)
        elif i == x_pixel_range/2:
            alpha_rads = 0
        elif (i > x_pixel_range/2) and (i < x_pixel_range):
            alpha = (i - x_pixel_range / 2) * FoVH / (x_pixel_range)
            alpha_rads = math.radians(alpha)
        elif i == x_pixel_range:
            alpha = FoVH/2
            alpha_rads = math.radians(alpha)
```

```
print ("alpha_rads is equal to: " + str(alpha_rads))
print ("Yaw_rads is equal to: " + str(Yaw_rads))

# Find the d_hyp distance in km for each respective pixel
d_pixel_km = d_pixel_proj_ctr_km/(cos(alpha_rads))

# Ensure this angle is strictly positive and less than 2*pi radians
if Yaw_rads + alpha_rads < 0:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads + 2 * numpy.pi

elif Yaw_right_rads + alpha_rads > 2 * numpy.pi:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads - 2 * numpy.pi
else:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads

print("Yaw_rads_adjusted is equal to: " + str(Yaw_rads_adjusted))

# Find the geographic coordinates for each specific pixel coordinate,
#  must add the calculated alpha to the Yaw value so we use
    Yaw_rads_adjusted
lat2_pixel = asin(sin(Latitude_rads) * cos(d_pixel_km / Radius_Earth) +
    cos(Latitude_rads)
                  * sin(d_pixel_km / Radius_Earth) * cos(
                      Yaw_rads_adjusted))
lon2_pixel = Longitude_rads + atan2(sin(Yaw_rads_adjusted) * sin(
    d_pixel_km / Radius_Earth)
                                    * cos(Latitude_rads), cos(d_pixel_km
                                        / Radius_Earth)
                                    - sin(Latitude_rads) * sin(
                                        lat2_pixel))

# Convert back to decimal degrees
lat2_pixel = math.degrees(lat2_pixel)
lon2_pixel = math.degrees(lon2_pixel)

print('The lat2_pixel is: '+str(lat2_pixel)+', given a x pixel of: '+
    str(i))
print('The lon2 pixel is: '+str(lon2_pixel)+', given a y pixel of: '+
    str(j)+'\n')

#
    ##########################################################################################################

# Temperature Calculation
# Some reference source on temperatures:
#  http://91.143.108.245/Downloads/Flir/Dokumentation/
    T559498$a461_Manual.pdf
# Temperature formula reference:
#  https://graftek.biz/system/files/137/
# original/FLIR_AX5_GenICam_ICD_Guide_052013.pdf?1376925336
# Radiance relation to A/D counts reference: http://flir.custhelp.com/ci
    /fattach/get/1667/
```

```
# Useful reference from FLIR for thermal imaging and
#  A/D counts/Signals generated from Thermal cameras:
#  http://www.hoskin.ca/wp-content/uploads/2016/10/
# flir_thermal_camera_guide_for_research_professionals.pdf

# Extract the RAW total signal value contained with the specific pixel
    as denoted by i and j
# Return the value to a variable
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " +
    directory + "/" + filename +
                             " 2>/dev/zero | magick - -crop 1X1+" +
                                 str(i) + "+" + str(j) +
                             " -colorspace gray -format '%[mean]' info:
                                 "], shell=True,
                            stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

#
    ##############################################################################################

# Calculate temperature for each specific pixel in in K and degC when
    emissivity < 1.0
# Call in Emis_29, Emis_31, Emis_32 and
#  apply Wang et al 2005 BroadBand Emissivity (BBE) formula
# See the following for more inofrmation: https://doi.org/10.1029/2004
    JD005566

# Check if BBE variables are in locals (do this for the first image only
    )
if 'emis_filename' not in locals():

    # Emissivity values are derived from the MODIS MODIS11B3 monthly
        land surface emissivity file
    # MODIS image and a grid of coorinates at 500m resolution were
        overlayed on each other in QGIS
    # Using the point extract tool, emissivity data and the
    #  corresponding geographic coordinates were extracted and saved to
        both a CSV and text file
    emis_filename = '/export/home/users/username/Documents/DG_Temp/
        Mining_Facility_2018/MODIS/' \
                    'Emissivty/QGIS/MODIS_Lat_Lon_Emissivity.txt'

    # Call in emissivity data
    emis_data = numpy.genfromtxt(emis_filename, delimiter=',',
        skip_header=1)

    # Latitude of Emissivity values
```

```
emis_lat = emis_data[:, 0]
# Longitude of Emissivity Values
emis_lon = emis_data[:, 1]
# MODIS Band 32 Emissivity Values
emis_32_uncorrected = emis_data[:, 2]
# MODIS Band 29 Emissivity Values
emis_29_uncorrected = emis_data[:, 3]
# MODIS Band 31 Emissivity Values
emis_31_uncorrected = emis_data[:, 4]

# Initialize corrected emissivity variables
emis_32_corrected = numpy.zeros((len(emis_32_uncorrected), 1))
emis_29_corrected = numpy.zeros((len(emis_32_uncorrected), 1))
emis_31_corrected = numpy.zeros((len(emis_32_uncorrected), 1))


# Convert all emissivity values i.e. multiply by scale factor and
    add offset as per
#  MODIS documentation: https://lpdaac.usgs.gov/sites/
# default/files/public/product_documentation/mod11_user_guide.pdf
emis_scale = 0.002
emis_offset = 0.49

# Calculate the true emissivity values for each band by applying the
     appropriate
#  scale factor and additive offset for each index of each array
for k in range(0, len(emis_lat)):
    # Apply Emissivity scale/offset factors to Band 29
    emis_29_corrected[k] = (emis_29_uncorrected[k]*emis_scale)+
        emis_offset
    # Apply Emissivity scale/offset factors to Band 31
    emis_31_corrected[k] = (emis_31_uncorrected[k]*emis_scale)+
        emis_offset
    # Apply Emissivity scale/offset factors to Band 32
    emis_32_corrected[k] = (emis_32_uncorrected[k]*emis_scale)+
        emis_offset

# Create new array for BroadBand Emissivity (BBE), BBE is used to
    calculation ST
BBEmissivity = numpy.zeros((len(emis_lat), 1))

# Initialize coefficients for BBE formula as per Wang et al 2005 pg
    7 of 12 Table 2
BBE_constant_29 = 0.2122
BBE_constant_31 = 0.3859
BBE_constant_32 = 0.4029

# Calculate BBE for each index
# Haversine Distance Formula from: https://stackoverflow.com/
    questions/19412462/
# getting-distance-between-two-points-based-on-latitude-longitude
for k in range(0, len(emis_lat)):
```

```
            BBEmissivity[k] = (BBE_constant_29*emis_29_corrected[k]) +\
                              (BBE_constant_31*emis_31_corrected[k]) +\
                              (BBE_constant_32*emis_32_corrected[k])

        # Initialize Haversine distance array
        # This array is used to calculate the geographic distance between
            the
        #  BBE values and the specific pixel location
        # The BBE index with the smallest distance between the two
            geographic coordinates
        #  will be used as the emissivity value in the ST calculation
        haversine_d = numpy.empty((len(emis_lat), 1))

    # Initialize haversine formula variable to be used in the
        HaversinePixelCalc function
    haversine_d[:] = numpy.nan

    # Run following function in parallel with @jit compiler
    # Haversine Distance Formula from:
    #  https://stackoverflow.com/questions/19412462/
    # getting-distance-between-two-points-based-on-latitude-longitude
    # As of Feb.5/2019, tested the stackoverflow example above and the
        haversine
    #  distance formula works when tested with Google Earth
    haversine_d = HaversinePixelCalc(emis_lat, lat2_pixel, emis_lon,
                                     lon2_pixel, Radius_Earth, haversine_d)

    # Find minimum index of the output of the haversine formula with the
        smallest distance
    # This will be the index that has the surface emissivity
    #  value that will be used in the temperature calculation for the
        specific pixel
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to
         1
    # Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

    # Reflected Apparent Temperature as per FLIR manual
    #  (http://www.cctvcentersl.es/upload/Manuales/A3xxx_A6xxx_manual_eng.
        pdf)
    #  and image metadata (use ExifTool in Linux terminal)

    # NOTE: For other remote sensing applications
    #  (e.g. thermal plumes in waterbodies), this value may change.
    # Return the reflected apparent temperature (degrees C) from the
    #  image metadata with the Linux terminal and ExifTool
    # The reflected apparent temperature is set as a constant by the
        manufacturer
    #  and is not changed during the temperature calculation
    refl_temp_degC = subprocess.Popen(["exiftool -b -
        ReflectedApparentTemperature " + directory
```

```
                                        + "/" + filename], shell=True,
                                        stdout=subprocess.PIPE).communicate()
                                            [0]

# Decode the value and convert it's data type to a float
refl_temp_degC = refl_temp_degC.decode("utf-8")
refl_temp_degC = float(refl_temp_degC)

# Convert reflected apparent temp from degC to K
refl_temp_K = pytemperature.c2k(refl_temp_degC)

# Get Raw reflected apparent temperature signal value
# See the FLIR Manual: http://www.cctvcentersl.es/upload/Manuales/
    A3xxx_A6xxx_manual_eng.pdf
# RAWrefl remains the same for all pixels as it is a function
#  of the constant apparent reflected temperature
RAWrefl = (R1/(R2*(math.exp(B/(refl_temp_K))-F))-planck_O)

# Get RAW object signal value
RAWobj = (RAW_total-(1-BBEmissivity[min_idx])*RAWrefl)/BBEmissivity[
    min_idx]

# Rearrange the RAW object signal value to calculate the object
#  temperature (LST) of each pixel in degC and K
# Consider the case for when emissivity is not equivalent to 1
LST_kelvin = (B/numpy.log(R1/(R2*(RAWobj+planck_O))+F))
LST_degree = (B/numpy.log(R1/(R2*(RAWobj+planck_O))+F)-273.15)

# Save data to matrix for the specific image
# The data in this matrix is then saved to a master matrix which will
    include all
#  LST data for pixels from each file in the directory
# Save the filename for each corresponding pixel location, not used in
    data analysis,
#  only used as a check as the files are not processed chronologically
filename_image[i][j][0] = filename

# Save the year for each corresponding pixel location for when the image
     was recorded
image_matrix[i][j][0] = yr
# Save the month for each corresponding pixel location for when the
    image was recorded
image_matrix[i][j][1] = mnth
# Save the day for each corresponding pixel location for when the image
    was recorded
image_matrix[i][j][2] = day
# Save the hour for each corresponding pixel location for when the image
     was recorded
image_matrix[i][j][3] = hr
# Save the minute for each corresponding pixel location for when the
    image was recorded
image_matrix[i][j][4] = minute
```

```
                # Save the calculated latitude for each corresponding pixel location
                image_matrix[i][j][5] = lat2_pixel
                # Save the calculated longitude for each corresponding pixel location
                image_matrix[i][j][6] = lon2_pixel
                # Save the horizontal pixel coordinate that was processed to obtain LST
                image_matrix[i][j][7] = i
                # Save the vertical pixel coordinate that was processed to obtain LST
                image_matrix[i][j][8] = j
                # Save the LST in kelvin of each corresponding pixel location where
                    emissivity does not equal 1
                image_matrix[i][j][9] = LST_kelvin
                # Save the LST in degC of each corresponding pixel location where
                    emissivity does not equal 1
                image_matrix[i][j][10] = LST_degree

#
    ##########################################################################################################################

# Save known latitude, longitude, x, and y pixels to arrays
# TANAB2 launch location
for origin in range(0, numFiles):
    if int(lat_TANAB_array[origin]) == 0:
        lat_TANAB_array[origin] = Latitude
        lon_TANAB_array[origin] = Longitude
        break

# Save known location for the top left pixel
# Initialize variables identifying top left pixel in terms of horizontal/vertical pixel
    row/column location
horiz_pixel = 0
vert_pixel = v_pixel_top

# Extract the RAW total value for the top left pixel through the terminal with ExifTool
    and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b --RawThermalImage " + directory + "/" +
    filename +
                             " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                 "+" + str(vert_pixel) +
                             " -colorspace gray -format '%[mean]' info:"], shell=True,
                            stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW total as it's a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW total from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Calculate Haversine distance for the top left pixel to identify the emissivity value
    to use in the ST calculation
```

```
haversine_d = HaversinePixelCalc_top_left(emis_lat, lat2_top_left, emis_lon,
                                          lon2_top_left, Radius_Earth, haversine_d)

# Find the index with the smallest distance between the 2 geographic coordinates
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent
#  reflective temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_top_left
image_matrix[horiz_pixel][vert_pixel][6] = lon2_top_left
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(tLeft_lat_array[i]) == 0:
        tLeft_lat_array[i] = lat2_top_left
        tLeft_lon_array[i] = lon2_top_left
        tLeft_x_pixel_array[i] = horiz_pixel
        tLeft_y_pixel_array[i] = vert_pixel
        break
#
    ##########################################################################################################################

# Save known location for the top center pixel
# Initialize variables identifying top center pixel in terms of horizontal/vertical
    pixel row/column location
horiz_pixel = int(x_pixel_range/2)
vert_pixel = v_pixel_top

# Extract the RAW total value for the top center pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b --RawThermalImage " + directory + "/" +
```

```
    filename +
                                    " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                        "+" + str(vert_pixel) +
                                    " -colorspace gray -format '%[mean]' info:"], shell=True,
                                   stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine top center pixel function to identify the emissivity value to use in
    the ST calculation
haversine_d = HaversinePixelCalc_top_center(emis_lat, lat2_top, emis_lon, lon2_top,
    Radius_Earth, haversine_d)

# Find the index with the smallest distance between the 2 geographic coordinates
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_top
image_matrix[horiz_pixel][vert_pixel][6] = lon2_top
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(tCenter_lat_array[i]) == 0:
```

```
            tCenter_lat_array[i] = lat2_top
            tCenter_lon_array[i] = lon2_top
            tCenter_x_pixel_array[i] = horiz_pixel
            tCenter_y_pixel_array[i] = vert_pixel
            break

#
    ##########################################################################################################################

# Save known coordinates for the top right pixel
# Initialize variables identifying top right pixel in terms of horizontal/vertical pixel
    row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = v_pixel_top

# Extract the RAW total value for the top right pixel through the terminal with ExifTool
    and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info:"], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine top right pixel function to identify the emissivity value to use in the
    ST calculation
haversine_d = HaversinePixelCalc_top_right(emis_lat, lat2_top_right, emis_lon,
    lon2_top_right,
                                           Radius_Earth, haversine_d)

# Find the index with the smallest distance between the 2 geographic coordinates
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)
```

```
# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_top_right
image_matrix[horiz_pixel][vert_pixel][6] = lon2_top_right
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(tRight_lat_array[i]) == 0:
        tRight_lat_array[i] = lat2_top_right
        tRight_lon_array[i] = lon2_top_right
        tRight_x_pixel_array[i] = horiz_pixel
        tRight_y_pixel_array[i] = vert_pixel
        break
#
    ##########################################################################################################################

# For the center left pixel
# Initialize variables identifying the center left pixel in terms of horizontal/vertical
    pixel row/column location
horiz_pixel = 0
vert_pixel = int(y_pixel_range / 2)

# Extract the RAW total value for the center left pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b --RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info: "], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine center left pixel function to identify the emissivity value to use in
```

```
    the ST calculation
haversine_d = HaversinePixelCalc_center_left(emis_lat, lat2_center_left, emis_lon,
                                             lon2_center_left, Radius_Earth, haversine_d
                                                 )

# Find the index with the smallest distance between the 2 geographic coordinates
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_center_left
image_matrix[horiz_pixel][vert_pixel][6] = lon2_center_left
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(cLeft_lat_array[i]) == 0:
        cLeft_lat_array[i] = lat2_center_left
        cLeft_lon_array[i] = lon2_center_left
        cLeft_x_pixel_array[i] = horiz_pixel
        cLeft_y_pixel_array[i] = vert_pixel
        break
#
    ##########################################################################################################

# For the center pixel
# Initialize variables identifying the center pixel in terms of horizontal/vertical
    pixel row/column location
horiz_pixel = int(x_pixel_range/2)
vert_pixel = int(y_pixel_range/2)

# Extract the RAW total value for the center pixel through the terminal with ExifTool
```

```
    and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
                                  " -colorspace gray -format '%[mean]' info:"], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine center pixel function to identify the emissivity value to use in the ST
    calculation
haversine_d = HaversinePixelCalc_center(emis_lat, lat2_center, emis_lon, lon2_center,
    Radius_Earth, haversine_d)

# Find minimum distance index
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in degC
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_center
image_matrix[horiz_pixel][vert_pixel][6] = lon2_center
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
```

```
for i in range(0, numFiles):
    if int(center_lat_array[i]) == 0:
        center_lat_array[i] = lat2_center
        center_lon_array[i] = lon2_center
        center_x_pixel_array[i] = horiz_pixel
        center_y_pixel_array[i] = vert_pixel
        break

#
    ##########################################################################################################

# For the center right pixel
# Initialize variables identifying the center right pixel in terms of horizontal/
    vertical pixel row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = int(y_pixel_range/2)

# Extract the RAW total value for the center right pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                             " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                 "+" + str(vert_pixel) +
                             " -colorspace gray -format '%[mean]' info:"], shell=True,
                            stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine center right pixel function to identify the emissivity value to use in
    the ST calculation
haversine_d = HaversinePixelCalc_center_right(emis_lat, lat2_center_right, emis_lon,
                                              lon2_center_right, Radius_Earth,
                                                  haversine_d)

# Find minimum distance index
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in Kelvin and degC respectively
```

```
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_center_right
image_matrix[horiz_pixel][vert_pixel][6] = lon2_center_right
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(cRight_lat_array[i]) == 0:
        cRight_lat_array[i] = lat2_center_right
        cRight_lon_array[i] = lon2_center_right
        cRight_x_pixel_array[i] = horiz_pixel
        cRight_y_pixel_array[i] = vert_pixel
        break

#
    ##########################################################################################################################

# For the bottom left pixel
# Initialize variables identifying the bottom left pixel in terms of horizontal/vertical
     pixel row/column location
horiz_pixel = 0
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom left pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
                                  " -colorspace gray -format '%[mean]' info:"], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
```

```
haversine_d[:] = numpy.nan

# Call Haversine bottom left pixel function to identify the emissivity value to use in
    the ST calculation
haversine_d = HaversinePixelCalc_bottom_left(emis_lat, lat2_bottom_left, emis_lon,
                                             lon2_bottom_left, Radius_Earth, haversine_d
                                                 )

# Find minimum distance index
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom_left
image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom_left
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(bLeft_lat_array[i]) == 0:
        bLeft_lat_array[i] = lat2_bottom_left
        bLeft_lon_array[i] = lon2_bottom_left
        bLeft_x_pixel_array[i] = horiz_pixel
        bLeft_y_pixel_array[i] = vert_pixel
        break
#
    ##########################################################################################################################

# For the bottom center pixel
# Initialize variables identifying the bottom center pixel in terms of horizontal/
    vertical pixel row/column location
horiz_pixel = int(x_pixel_range/2)
```

```
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom center pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
                                  " -colorspace gray -format '%[mean]' info: "], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine bottom pixel function to identify the emissivity value to use in the ST
     calculation
haversine_d = HaversinePixelCalc_bottom(emis_lat, lat2_bottom, emis_lon, lon2_bottom,
    Radius_Earth, haversine_d)

# Find minimum distance index
min_idx = numpy.argmin(haversine_d)

# For the temperature calc. assume that transmissivity is approx 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temp is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]

# Calculate temperature of each pixel in kelvin and degC respectively
LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

# Save data to matrix for specific image
filename_image[horiz_pixel][vert_pixel][0] = filename

image_matrix[horiz_pixel][vert_pixel][0] = yr
image_matrix[horiz_pixel][vert_pixel][1] = mnth
image_matrix[horiz_pixel][vert_pixel][2] = day
image_matrix[horiz_pixel][vert_pixel][3] = hr
image_matrix[horiz_pixel][vert_pixel][4] = minute
image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom
image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom
image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
```

```
image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

# Save latitude, longitude, x, and y pixel values to arrays
for i in range(0, numFiles):
    if int(bCenter_lat_array[i]) == 0:
        bCenter_lat_array[i] = lat2_bottom
        bCenter_lon_array[i] = lon2_bottom
        bCenter_x_pixel_array[i] = horiz_pixel
        bCenter_y_pixel_array[i] = vert_pixel
        break

#
    ##########################################################################################################################

# For the bottom right pixel
# Initialize variables identifying the bottom right pixel in terms of horizontal/
    vertical pixel row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom right pixel through the terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info:"], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a data type string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan

# Call Haversine bottom right pixel function
haversine_d = HaversinePixelCalc_bottom_right(emis_lat, lat2_bottom_right, emis_lon,
                                              lon2_bottom_right, Radius_Earth,
                                                  haversine_d)

# Find minimum distance index
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to 1
# Source: Usamentiaga et al. doi: https://doi.org/10.3390/s140712305

# Calculate RAW object signal value, RAWrefl does not change as the apparent reflective
    temperature is constant
RAWobj = (RAW_total - (1 - BBEmissivity[min_idx]) * RAWrefl) / BBEmissivity[min_idx]
```

```
    # Calculate temperature of each pixel in kelvin and degC respectively
    LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
    LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

    # Save data to matrix for specific image
    filename_image[horiz_pixel][vert_pixel][0] = filename

    image_matrix[horiz_pixel][vert_pixel][0] = yr
    image_matrix[horiz_pixel][vert_pixel][1] = mnth
    image_matrix[horiz_pixel][vert_pixel][2] = day
    image_matrix[horiz_pixel][vert_pixel][3] = hr
    image_matrix[horiz_pixel][vert_pixel][4] = minute
    image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom_right
    image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom_right
    image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
    image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
    image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
    image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

    # Save latitude, longitude, x, and y pixel values to arrays
    for i in range(0, numFiles):
        if int(bRight_lat_array[i]) == 0:
            bRight_lat_array[i] = lat2_bottom_right
            bRight_lon_array[i] = lon2_bottom_right
            bRight_x_pixel_array[i] = horiz_pixel
            bRight_y_pixel_array[i] = vert_pixel
            break
    #
        ##########################################################################################################################


    # Save image matrix data to master matrix
    all_pixel_data_multi_image = SaveMasterMatrix(x_pixel_range, v_pixel_top,
                                                  y_pixel_range, image_matrix,
                                                      all_pixel_data_multi_image,
                                                  filename_image, filenames_total)

# The number of elements is equivalent to the total length of the all_pixel
# _data_multi_image array (the total length is the maximum number
#  of data points that could be extracted from each image
num_elements = int((numFiles*x_max_step*y_max_step)-1)

# Variable corresponding to the row where the Nan values start
row_nan = 0

# Find the index where the Nan rows start (do this so Nan values are not written to the file
    )
# This variable will be used when saving the data to omit any Nan data values from the
    Processed_data text file
for i in range(0, num_elements):
    if numpy.isnan(all_pixel_data_multi_image[i][5]) == True:
```

```
        row_nan = i
        break

# Today's date is
today_date = datetime.date.today().strftime("%B_%d_%Y")

# Write Geographic, ST and Image Pixel Data to File
# XXXMEDIA refers to the naming convention of the DJI Zenmuse XT, where XXX starts at 100
    and increases by 1
#  for every new folder. Up to 999 images can be stored in each folder. Replace XXX with the
     image folder number
#  that is currently being processed
outputFileName = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                 'XXXMEDIA_temperature.txt'
outputFile = open(outputFileName, 'w')
outputFile.write("#,Date,,Time,,Lat,,Long,and,Temp,for,each,image,\n")
outputFile.write("#By:,Ryan,Byerlay,\n")
outputFile.write("#Created,on,"+today_date+",\n")
outputFile.write("#Recorded,Time,is,Local,Time,(MDT),\n")
outputFile.write("#0:,Picture,File,Name,\t,#1:Year,\t,,#2:Month,\t,,#3:Day,\t,#4:Hour,\t,#5:
    Minute"
                 ",\t,#6:,Latitude,\t,#7:,Longitude,\t,#8:,X,Pixel,Coordinate,\t,#9:,Y,Pixel
                     ,Coordinate,\t"
                 ",#12:,Temperature,(K),(Emis,!=,1),\t,#13:,Temperature,(C),(Emis,!=,1),\n")

# Save data to file
for i in range(0, row_nan):
     outputFile.write("%s,\t,%i,\t,%i,\t,%i,\t,%i,\t,%i,\t,%f,\t,%f,\t,%i,\t,%i,\t,%f,\t,%f,
         \n" %
                      (filenames_total[i][0], int(all_pixel_data_multi_image[i][0]),
                       int(all_pixel_data_multi_image[i][1]), int(all_pixel_data_multi_image
                           [i][2]),
                       int(all_pixel_data_multi_image[i][3]), int(all_pixel_data_multi_image
                           [i][4]),
                       all_pixel_data_multi_image[i][5], all_pixel_data_multi_image[i][6],
                       int(all_pixel_data_multi_image[i][7]), int(all_pixel_data_multi_image
                           [i][8]),
                       all_pixel_data_multi_image[i][9], all_pixel_data_multi_image[i][10]))
outputFile.close()

# Function to populate known latitude/longitude coordinates for each image
def determineFileName(filenames_total, file_names_array):
    for w in range(0, numFiles):
        if filenames_total[k][0] == file_names_array[w]:
            # NOTE x_latitudes[0] and y_longitudes[0] correspond to the TANAB2 location
            # For every new filename, populate the known latitude, longitude,
            #  x pixel, and y pixel arrays
            x_Latitudes = [lat_TANAB_array[w], tLeft_lat_array[w], tCenter_lat_array[w],
                tRight_lat_array[w],
                           cLeft_lat_array[w], center_lat_array[w], cRight_lat_array[w],
                               bLeft_lat_array[w],
```

```
                    bCenter_lat_array[w], bRight_lat_array[w]]
            y_Longitudes = [lon_TANAB_array[w], tLeft_lon_array[w], tCenter_lon_array[w],
                tRight_lon_array[w],
                    cLeft_lon_array[w], center_lon_array[w], cRight_lon_array[w],
                        bLeft_lon_array[w],
                    bCenter_lon_array[w], bRight_lon_array[w]]
            x_pixels = [tLeft_x_pixel_array[w], tCenter_x_pixel_array[w],
                tRight_x_pixel_array[w],
                        cLeft_x_pixel_array[w], center_x_pixel_array[w],
                            cRight_x_pixel_array[w],
                        bLeft_x_pixel_array[w], bCenter_x_pixel_array[w],
                            bRight_x_pixel_array[w]]
            y_pixels = [tLeft_y_pixel_array[w], tCenter_y_pixel_array[w],
                tRight_y_pixel_array[w],
                        cLeft_y_pixel_array[w], center_y_pixel_array[w],
                            cRight_y_pixel_array[w],
                        bLeft_y_pixel_array[w], bCenter_y_pixel_array[w],
                            bRight_y_pixel_array[w]]
            break
    return x_Latitudes, y_Longitudes, x_pixels, y_pixels

#
    ##########################################################################################################################

# Save kml (Google Earth) file
# Save edge coordinates as red markers and inner image coordinates as yellow markers
kml = simplekml.Kml(open=1)
pt_label = ['Balloon', 'Top Left', 'Top Center', 'Top Right', 'Center Left', 'Center', '
    Center Right',
            'Bottom Left', 'Bottom Center', 'Bottom Right']
# Loop through all rows of final save matrix
for k in range(0, row_nan):
    # Find indices where the index and index+1 has mismatched file names
    # If file names are not equal, then save kml file for the specific image file
    if filenames_total[k][0] != filenames_total[k+1][0]:
        # Initialize variables
        # consider edge coordinates and TANAB2 location for latitudes and longitudes
        x_Latitudes = numpy.zeros(10)
        y_Longitudes = numpy.zeros(10)
        # Only consider pixel coordinates for the specific image
        x_pixels = numpy.zeros(9)
        y_pixels = numpy.zeros(9)

        # Concatenate latitudes, longitudes, and pixel arrays accordingly and return
        x_Latitudes, y_Longitudes, x_pixels, y_pixels = determineFileName(filenames_total,
            file_names_array)

        # Save edge coordinate points to kml file
        # Set original counter value
        i = 0
        while i <= 9:
            known_pnts = kml.newpoint(name=str(pt_label[i]), coords=[(float(y_Longitudes[i])
```

```
                , float(x_Latitudes[i]))])
            known_pnts.style.iconstyle.color = simplekml.Color.red
            # Increase counter by 1
            i += 1

        # Save existing kml file, update XXX to match the folder with the images that you
            are processing
        kml.save("/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
            Google_Earth_Projections/"
                "XXXMEDIA/GPS_visualize_"
                + str(filenames_total[k][0])+"_OCT_15_2019_test.kml")

        # Delete old kml file variables including kml, known_pnts, x_coordinates,
        #  y_coordinates then continue to the next image file
        del kml
        del known_pnts
        del x_Latitudes
        del y_Longitudes

        # Create new kml file
        if 'kml' not in locals():
            kml = simplekml.Kml(open=1)
    # Save specific coordinate to existing kml file
    else:
        pnt = kml.newpoint(name='P('+str(all_pixel_data_multi_image[k][7])+','
                                +str(all_pixel_data_multi_image[k][8])+')',coords =
        [(all_pixel_data_multi_image[k][6], all_pixel_data_multi_image[k][5])])

# Get ending run time
end = time.time()

# Print script run time in seconds
print('The total run time of this script is: '+str(end-start)+' s')
```

A.2.5 Data Separation for Diurnal Temperature Mapping

```
import numpy
from numba import jit

# Current as of October 16, 2019
# Previously completed: Manually concatenated all processed image text
#  file containing surface temperatures for individual pixels
# Import the combined file into script, manually combined all XXXMEDIA_temperature.txt files
    into one.
filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/' \
           'Processed_Data/May_2018_Mine_compiled_temperature.txt'

data = numpy.genfromtxt(filename, skip_header=6)

# Separate data
filename = data[:,0]
```

```
year = data[:,1]
month = data[:,2]
day = data[:,3]
hour = data[:,4]
min = data[:,5]
lat = data[:,6]
lon = data[:,7]
xpix = data[:,8]
ypix = data[:,9]
tempk_emis = data[:,10]
tempc_emis = data[:,11]

len_year = int(len(year))

# Create new arrays for data (six four-hour intervals)
zero_four_array = numpy.zeros((14, len_year))
four_eight_array = numpy.zeros((14, len_year))
eight_twelve_array = numpy.zeros((14, len_year))
twelve_sixteen_array = numpy.zeros((14, len_year))
sixteen_twenty_array = numpy.zeros((14, len_year))
twenty_twentyfour_array = numpy.zeros((14, len_year))

# Process function in parallel to find indices with hours corresponding to the six
#  four-hour time intervals delineated above
@jit(nopython=True, parallel=True)
def FindHour(year, month, day, hour, min, lat, lon, xpix, ypix, tempk_emis, tempc_emis,
    zero_four_array,
            four_eight_array, eight_twelve_array, twelve_sixteen_array,
                sixteen_twenty_array, twenty_twentyfour_array):
    for i in range(0, len(year)):
        print(i)
        if (int(hour[i]) < 0):
            print('There is a problem with the hour in index: '+str(i))

        # 00:00 to 04:00 Check
        if ((int(hour[i]) >= 0) and (int(hour[i] <= 3))) or int(hour[i]) == 24:
            for j in range(0, len(year)):
                if zero_four_array[1][j] == 0:
                    zero_four_array[1][j] = year[i]
                    zero_four_array[2][j] = month[i]
                    zero_four_array[3][j] = day[i]
                    zero_four_array[4][j] = hour[i]
                    zero_four_array[5][j] = min[i]
                    zero_four_array[6][j] = lat[i]
                    zero_four_array[7][j] = lon[i]
                    zero_four_array[8][j] = xpix[i]
                    zero_four_array[9][j] = ypix[i]
                    zero_four_array[10][j] = tempk_emis[i]
                    zero_four_array[11][j] = tempc_emis[i]
                    break

        # 04:00 to 08:00 check
```

```
elif (int(hour[i]) >= 4 and int(hour[i]) <= 7):
    for j in range(0, len(year)):
        if four_eight_array[1][j] == 0:
            four_eight_array[1][j] = year[i]
            four_eight_array[2][j] = month[i]
            four_eight_array[3][j] = day[i]
            four_eight_array[4][j] = hour[i]
            four_eight_array[5][j] = min[i]
            four_eight_array[6][j] = lat[i]
            four_eight_array[7][j] = lon[i]
            four_eight_array[8][j] = xpix[i]
            four_eight_array[9][j] = ypix[i]
            four_eight_array[10][j] = tempk_emis[i]
            four_eight_array[11][j] = tempc_emis[i]
            break

# 08:00 to 12:00 check
elif (int(hour[i]) >= 8 and int(hour[i]) <= 11) :
    for j in range(0, len(year)):
        if eight_twelve_array[1][j] == 0:
            eight_twelve_array[1][j] = year[i]
            eight_twelve_array[2][j] = month[i]
            eight_twelve_array[3][j] = day[i]
            eight_twelve_array[4][j] = hour[i]
            eight_twelve_array[5][j] = min[i]
            eight_twelve_array[6][j] = lat[i]
            eight_twelve_array[7][j] = lon[i]
            eight_twelve_array[8][j] = xpix[i]
            eight_twelve_array[9][j] = ypix[i]
            eight_twelve_array[10][j] = tempk_emis[i]
            eight_twelve_array[11][j] = tempc_emis[i]
            break

# 12:00 to 16:00 check
elif (int(hour[i]) >= 12 and int(hour[i]) <= 15) :
    for j in range(0, len(year)):
        if twelve_sixteen_array[1][j] == 0:
            twelve_sixteen_array[1][j] = year[i]
            twelve_sixteen_array[2][j] = month[i]
            twelve_sixteen_array[3][j] = day[i]
            twelve_sixteen_array[4][j] = hour[i]
            twelve_sixteen_array[5][j] = min[i]
            twelve_sixteen_array[6][j] = lat[i]
            twelve_sixteen_array[7][j] = lon[i]
            twelve_sixteen_array[8][j] = xpix[i]
            twelve_sixteen_array[9][j] = ypix[i]
            twelve_sixteen_array[10][j] = tempk_emis[i]
            twelve_sixteen_array[11][j] = tempc_emis[i]
            break

# 16:00 to 20:00 check
elif (int(hour[i]) >= 16 and int(hour[i]) <= 19):
```

```
            for j in range(0, len(year)):
                if sixteen_twenty_array[1][j] == 0:
                    sixteen_twenty_array[1][j] = year[i]
                    sixteen_twenty_array[2][j] = month[i]
                    sixteen_twenty_array[3][j] = day[i]
                    sixteen_twenty_array[4][j] = hour[i]
                    sixteen_twenty_array[5][j] = min[i]
                    sixteen_twenty_array[6][j] = lat[i]
                    sixteen_twenty_array[7][j] = lon[i]
                    sixteen_twenty_array[8][j] = xpix[i]
                    sixteen_twenty_array[9][j] = ypix[i]
                    sixteen_twenty_array[10][j] = tempk_emis[i]
                    sixteen_twenty_array[11][j] = tempc_emis[i]
                    break

        # 20:00 to 24:00 check
        elif (int(hour[i]) >= 20 and int(hour[i]) <= 23):
            for j in range(0, len(year)):
                if twenty_twentyfour_array[1][j] == 0:
                    twenty_twentyfour_array[1][j] = year[i]
                    twenty_twentyfour_array[2][j] = month[i]
                    twenty_twentyfour_array[3][j] = day[i]
                    twenty_twentyfour_array[4][j] = hour[i]
                    twenty_twentyfour_array[5][j] = min[i]
                    twenty_twentyfour_array[6][j] = lat[i]
                    twenty_twentyfour_array[7][j] = lon[i]
                    twenty_twentyfour_array[8][j] = xpix[i]
                    twenty_twentyfour_array[9][j] = ypix[i]
                    twenty_twentyfour_array[10][j] = tempk_emis[i]
                    twenty_twentyfour_array[11][j] = tempc_emis[i]
                    break

    return zero_four_array, eight_twelve_array, twelve_sixteen_array, sixteen_twenty_array,
        twenty_twentyfour_array

# Call function to separate hours and create six four-hour arrays with the appropriate data
FindHour(year, month, day, hour, min, lat, lon, xpix, ypix, tempk_emis, tempc_emis,
    zero_four_array,
        four_eight_array, eight_twelve_array, twelve_sixteen_array, sixteen_twenty_array,
            twenty_twentyfour_array)

#
    ##########################################################################################################################

# Save arrays to text files
# Declare file names
filename_zero_four ='/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                    'Separated_Hours/Manufacturer_Calibrated/Zero_Four_Data_Processed.txt'
filename_four_eight = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                      'Separated_Hours/Manufacturer_Calibrated/Four_Eight_Data_Processed.txt
```

```
                          '
filename_eight_twelve = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                        'Separated_Hours/Manufacturer_Calibrated/Eight_Twelve_Data_Processed
                            .txt'
filename_twelve_sixteen = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                          'Separated_Hours/Manufacturer_Calibrated/
                              Twelve_Sixteen_Data_Processed.txt'
filename_sixteen_twenty = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                          'Separated_Hours/Manufacturer_Calibrated/
                              Sixteen_Twenty_Data_Processed.txt'
filename_twenty_twentyfour = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                            'Separated_Hours/Manufacturer_Calibrated/
                                Twenty_Twentyfour_Data_Processed.txt'

#
    ##########################################################################################################################

# Zero_four (0000-0400) data
outputFile_zero_four = open(filename_zero_four, 'w')
outputFile_zero_four.write("# Date, Time, Lat, Long and Temp for each image from 00:00 to
    03:59 \n")
outputFile_zero_four.write("#By: Ryan Byerlay \n")
outputFile_zero_four.write("#Recorded Time is Local Time (MDT) \n")
outputFile_zero_four.write("#Note: Under column #6 time with a single digit representing the
    minutes is = to 0X"
                           " where X is the number in the column so at the top of the hr
                               only 0 would be present \n")
outputFile_zero_four.write("#0: Picture File Name \t #1:Year \t #2:Month \t #3:Day \t #4:
    Hour \t #5:Minute"
                           " \t #6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t
                               #9: Y Pixel Coordinate"
                           " \t #10: Temperature (K) (Emis != 1) \t #11: Temperature (C) (
                               Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if zero_four_array[1][i] != 0:
        outputFile_zero_four.write("%s \t %i \t %i \t %i \t %i \t %i \t"
                                   " %f \t %f \t %i \t %i \t %f \t %f \n" %
                                   (zero_four_array[0][i], int(zero_four_array[1][i]), int(
                                       zero_four_array[2][i]),
                                    int(zero_four_array[3][i]), int(zero_four_array[4][i]),
                                        int(zero_four_array[5][i]),
                                    zero_four_array[6][i], zero_four_array[7][i], int(
                                        zero_four_array[8][i]),
                                    int(zero_four_array[9][i]), zero_four_array[10][i],
                                        zero_four_array[11][i]))
outputFile_zero_four.close()
```

```
#
    ##########################################################################################################################

# Four_eight (0400-0800) data
outputFile_four_eight = open(filename_four_eight, 'w')
outputFile_four_eight.write("# Date, Time, Lat, Long and Temp for each image from 04:00 to
    07:59 \n")
outputFile_four_eight.write("#By: Ryan Byerlay \n")
outputFile_four_eight.write("#Recorded Time is Local Time (MDT) \n")
outputFile_four_eight.write("#NOTE: Under column #6 time with a single digit representing
    the minutes is = to 0X"
                            " where X is the number in the column so at the top of the hr
                                only 0 would be present \n")
outputFile_four_eight.write("#0: Picture File Name \t #1:Year \t\t #2:Month \t\t #3:Day \t #4:
    Hour \t #5:Minute"
                            " \t #6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t
                                #9: Y Pixel Coordinate"
                            " \t #10: Temperature (K) (Emis != 1) \t #11: Temperature (C) (
                                Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if four_eight_array[1][i] != 0:
        outputFile_four_eight.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i \t %
            i \t %f \t %f \n"
                                    % (four_eight_array[0][i], int(four_eight_array[1][i]),
                                        int(four_eight_array[2][i]), int(four_eight_array[3][
                                        i]),
                                        int(four_eight_array[4][i]), int(four_eight_array[5][
                                        i]),
                                        four_eight_array[6][i], four_eight_array[7][i], int(
                                        four_eight_array[8][i]),
                                        int(four_eight_array[9][i]), four_eight_array[10][i],
                                         four_eight_array[11][i]))
outputFile_four_eight.close()
#
    ##########################################################################################################################

# Eight_twelve (0800-1200) data
outputFile_eight_twelve = open(filename_eight_twelve, 'w')
outputFile_eight_twelve.write("# Date, Time, Lat, Long and Temp for each image from 08:00 to
    11:59 \n")
outputFile_eight_twelve.write("#By: Ryan Byerlay \n")
outputFile_eight_twelve.write("#Recorded Time is Local Time (MDT) \n")
outputFile_eight_twelve.write("#NOTE: Under column #6 time with a single digit representing
    the minutes is"
                              " = to 0X where X is the number in the column so at the top of
                                  the hr only 0"
                              " would be present \n")
outputFile_eight_twelve.write("#0: Picture File Name \t #1:Year \t\t #2:Month \t\t #3:Day \t
    #4:Hour \t"
```

```
                                "␣#5:Minute␣\t␣#6:␣Latitude␣\t␣#7:␣Longitude␣\t␣#8:␣X␣Pixel␣
                                    Coordinate"
                                "␣\t␣#9:␣Y␣Pixel␣Coordinate␣\t␣␣#10:␣Temperature␣(K)␣(Emis␣!=␣
                                    1)␣\t"
                                "␣#11:␣Temperature␣(C)␣(Emis␣!=␣1)␣\n")

# Save data to file
for i in range(0, len_year):
    if eight_twelve_array[1][i] != 0:
        outputFile_eight_twelve.write("%s␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%f␣\t␣%f␣\t␣%i␣\t
            ␣%i␣\t␣%f␣\t␣%f␣\n" %
                                      (eight_twelve_array[0][i], int(eight_twelve_array[1][i
                                          ]),
                                       int(eight_twelve_array[2][i]), int(eight_twelve_array
                                           [3][i]),
                                       int(eight_twelve_array[4][i]), int(eight_twelve_array
                                           [5][i]),
                                       eight_twelve_array[6][i], eight_twelve_array[7][i],
                                       int(eight_twelve_array[8][i]), int(eight_twelve_array
                                           [9][i]),
                                       eight_twelve_array[10][i], eight_twelve_array[11][i])
                                           )
outputFile_eight_twelve.close()

#
    ##########################################################################################################################################

# Twelve_sixteen (1200-1600) data
outputFile_twelve_sixteen = open(filename_twelve_sixteen, 'w')
outputFile_twelve_sixteen.write("#␣Date,␣Time,␣Lat,␣Long␣and␣Temp␣for␣each␣image␣from␣12:00␣
    to␣15:59␣\n")
outputFile_twelve_sixteen.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_twelve_sixteen.write("#Recorded␣Time␣is␣Local␣Time␣(MDT)␣\n")
outputFile_twelve_sixteen.write("#NOTE:␣Under␣column␣#6␣time␣with␣a␣single␣digit␣
    representing␣the␣minutes␣is␣="
                                "␣to␣0X␣where␣X␣is␣the␣number␣in␣the␣column␣so␣at␣the␣top␣of
                                    ␣the␣hr␣only␣0"
                                "␣would␣be␣present␣\n")
outputFile_twelve_sixteen.write("#0:␣Picture␣File␣Name␣\t␣#1:Year␣\t␣␣#2:Month␣\t␣␣#3:Day␣\t
    ␣#4:Hour␣\t␣#5:Minute"
                                "␣\t␣#6:␣Latitude␣\t␣#7:␣Longitude␣\t␣#8:␣X␣Pixel␣Coordinate
                                    ␣\t"
                                "␣#9:␣Y␣Pixel␣Coordinate␣\t␣#10:␣Temperature␣(K)␣(Emis␣!=␣1)
                                    "
                                "␣\t␣#11:␣Temperature␣(C)␣(Emis␣!=␣1)␣\n")

# Save data to file
for i in range(0, len_year):
    if twelve_sixteen_array[1][i] != 0:
        outputFile_twelve_sixteen.write("%s␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%f␣\t␣%f␣\t␣%i␣
            \t␣%i␣\t␣%f␣\t␣%f␣\n" %
                                        (twelve_sixteen_array[0][i], int(
```

```
                                        twelve_sixteen_array[1][i]),
                                    int(twelve_sixteen_array[2][i]), int(
                                        twelve_sixteen_array[3][i]),
                                    int(twelve_sixteen_array[4][i]), int(
                                        twelve_sixteen_array[5][i]),
                                    twelve_sixteen_array[6][i], twelve_sixteen_array
                                        [7][i],
                                    int(twelve_sixteen_array[8][i]), int(
                                        twelve_sixteen_array[9][i]),
                                    twelve_sixteen_array[10][i], twelve_sixteen_array
                                        [11][i]))
outputFile_twelve_sixteen.close()

#
    ##########################################################################################################################

# Sixteen_twenty (1600-2000) data
outputFile_sixteen_twenty = open(filename_sixteen_twenty, 'w')
outputFile_sixteen_twenty.write("#␣Date,␣Time,␣Lat,␣Long␣and␣Temp␣for␣each␣image␣from␣16:00␣
    to␣19:59␣\n")
outputFile_sixteen_twenty.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_sixteen_twenty.write("#Recorded␣Time␣is␣Local␣Time␣(MDT)␣\n")
outputFile_sixteen_twenty.write("#NOTE:␣Under␣column␣#6␣time␣with␣a␣single␣digit␣
    representing␣the␣minutes␣is␣="
                                "␣to␣0X␣where␣X␣is␣the␣number␣in␣the␣column␣so␣at␣the␣top␣of
                                    ␣the␣hr␣only␣0"
                                "␣would␣be␣present␣\n")
outputFile_sixteen_twenty.write("#0:␣Picture␣File␣Name␣\t␣#1:Year␣\t␣␣#2:Month␣\t␣␣#3:Day␣\t
    ␣#4:Hour␣\t␣#5:Minute"
                                "␣\t␣#6:␣Latitude␣\t␣#7:␣Longitude␣\t␣#8:␣X␣Pixel␣Coordinate
                                    "
                                "␣\t␣#9:␣Y␣Pixel␣Coordinate␣\t␣#10:␣Temperature␣(K)␣(Emis␣!=
                                    ␣1)"
                                "␣\t␣#11:␣Temperature␣(C)␣(Emis␣!=␣1)␣\n")

# Save data to file
for i in range(0, len_year):
    if sixteen_twenty_array[1][i] != 0:
        outputFile_sixteen_twenty.write("%s␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%f␣\t␣%f␣\t␣%i␣
            \t␣%i␣\t␣%f␣\t␣%f␣\n" %
                                        (sixteen_twenty_array[0][i], int(
                                            sixteen_twenty_array[1][i]),
                                         int(sixteen_twenty_array[2][i]), int(
                                             sixteen_twenty_array[3][i]),
                                         int(sixteen_twenty_array[4][i]), int(
                                             sixteen_twenty_array[5][i]),
                                         sixteen_twenty_array[6][i], sixteen_twenty_array
                                             [7][i],
                                         int(sixteen_twenty_array[8][i]), int(
                                             sixteen_twenty_array[9][i]),
                                         sixteen_twenty_array[10][i], sixteen_twenty_array
                                             [11][i]))
```

```
outputFile_sixteen_twenty.close()

#
    ##########################################################################################################################################

# Twenty_twenty-four (2000-2400) data
outputFile_twenty_twentyfour = open(filename_twenty_twentyfour, 'w')
outputFile_twenty_twentyfour.write("# Date, Time, Lat, Long and Temp for each image from
    20:00 to 23:59 \n")
outputFile_twenty_twentyfour.write("#By: Ryan Byerlay \n")
outputFile_twenty_twentyfour.write("#Recorded Time is Local Time (MDT) \n")
outputFile_twenty_twentyfour.write("#NOTE: Under column #6 time with a single digit
    representing the minutes is ="
                                   " to 0X where X is the number in the column so at the top
                                       of the hr only 0"
                                   " would be present \n")
outputFile_twenty_twentyfour.write("#0: Picture File Name \t #1:Year \t  #2:Month \t  #3:Day
    \t #4:Hour \t"
                                   " #5:Minute \t #6: Latitude \t #7: Longitude \t #8: X
                                       Pixel Coordinate"
                                   " \t #9: Y Pixel Coordinate \t #10: Temperature (K) (Emis
                                       != 1)"
                                   " \t #11: Temperature (C) (Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if twenty_twentyfour_array[1][i] != 0:
        outputFile_twenty_twentyfour.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t
            %i \t %i \t %f \t %f \n" %
                                           (twenty_twentyfour_array[0][i], int(
                                               twenty_twentyfour_array[1][i]),
                                            int(twenty_twentyfour_array[2][i]), int(
                                                twenty_twentyfour_array[3][i]),
                                            int(twenty_twentyfour_array[4][i]), int(
                                                twenty_twentyfour_array[5][i]),
                                            twenty_twentyfour_array[6][i],
                                                twenty_twentyfour_array[7][i],
                                            int(twenty_twentyfour_array[8][i]), int(
                                                twenty_twentyfour_array[9][i]),
                                            twenty_twentyfour_array[10][i],
                                                twenty_twentyfour_array[11][i]))
outputFile_twenty_twentyfour.close()
```

A.2.6 Applying Thermal Camera Calibration Constants to Land Surface Temperatures

```
import numpy

# Current as of October 18, 2019
# Apply thermal camera calibration constants to calculated surface temperatures based on
```

```
#  land surface material and geographic position

# State original FLIR factory Planck constants
R1_flir = 17096.453
R2_flir = 0.046642166
R_flir = R1_flir/R2_flir
B_flir = 1428
O_flir =-342
F_flir =1

# Soil FLIR constants
R_soil = 549800
B_soil = 1510
O_soil = -171
F_soil = 1.5

# Developed land (Concrete) FLIR Constants
R_concrete = 247614
B_concrete = 1322
O_concrete = -513
F_concrete = 1.5

# Water FLIR Constants
R_water = 549789
B_water = 1507
O_water = -171
F_water = 1.5

#
    ##########################################################################################################################

# Load data for 00:00 to 04:00
zero_four_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                     'Separated_Hours/Manufacturer_Calibrated/Zero_Four_Data_Processed.txt'

zero_four_data = numpy.genfromtxt(zero_four_filename, usecols=[6,7,10])

# Consider emissivity does not equal 1
zero_four_lat_pre_filter = zero_four_data[:,0]
zero_four_lon_pre_filter = zero_four_data[:,1]
zero_four_tempK_pre_filter = zero_four_data[:,2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
zero_four_lat = numpy.zeros((len(zero_four_lat_pre_filter)))
zero_four_lat[:] = numpy.nan
zero_four_lon = numpy.zeros((len(zero_four_lat_pre_filter)))
zero_four_lon[:] = numpy.nan
zero_four_tempK = numpy.zeros((len(zero_four_lat_pre_filter)))
zero_four_tempK[:] = numpy.nan
```

```
# Filter out geographic coordinates for the heat maps
for point in range(0, len(zero_four_lat)):
    print(zero_four_lon_pre_filter[point])
    print(zero_four_lat_pre_filter[point])
    if zero_four_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((zero_four_lon_pre_filter[point] >= -XXX.XXXXXX and zero_four_lon_pre_filter[point
        ] <= -XXX.XXXXXX)
          and (zero_four_lat_pre_filter[point] >= XX.XXXXXX and zero_four_lat_pre_filter[
              point] <= XX.XXXXXX)):
        continue
    elif ((zero_four_lon_pre_filter[point] >= -XXX.XXXXXX and zero_four_lon_pre_filter[point
        ] <= -XXX.XXXXXX)
          and (zero_four_lat_pre_filter[point] >= XX.XXXXXX and zero_four_lat_pre_filter[
              point] <= XX.XXXXXX)):
        continue
    elif ((zero_four_lon_pre_filter[point] >= -XXX.XXXXXX and zero_four_lon_pre_filter[point
        ] <= -XXX.XXXXX)
          and (zero_four_lat_pre_filter[point] >= XX.XXXXXX and zero_four_lat_pre_filter[
              point] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(zero_four_lat)):
            # Check if index is Nan
            if numpy.isnan(zero_four_lat[index]) == True:
                zero_four_lat[index] = zero_four_lat_pre_filter[point]
                zero_four_lon[index] = zero_four_lon_pre_filter[point]
                zero_four_tempK[index] = zero_four_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(zero_four_lat)):
    if numpy.isnan(zero_four_lat[nan_index]) == True:
        # Remove Nan values from arrays
        zero_four_lat = zero_four_lat[0:nan_index]
        zero_four_lon = zero_four_lon[0:nan_index]
        zero_four_tempK = zero_four_tempK[0:nan_index]
        break

#
    ##########################################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula:
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_zero_four = numpy.zeros((len(zero_four_lat)))

# Initialize new corrected temperature array
corrected_temp_zero_four = numpy.zeros((len(zero_four_lon)))
```

```
# Calculate pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(zero_four_lat)):
    Upixel_zero_four[i] = (R_flir/(numpy.exp(B_flir/zero_four_tempK[i])-F_flir))-O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(zero_four_tempK)):
    # If longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    # The tailings pond corresponds to developed land parameters
    if (zero_four_lon[i] >= -XXX.XXXX) or (zero_four_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land constants
        if ((zero_four_lon[i] >= -XXX.XXXXXX and zero_four_lon[i] <= -XXX.XXXXXX) and
            (zero_four_lat[i] >= XX.XXXXXX and zero_four_lat[i] <= XX.XXXXXX))\
                or ((zero_four_lon[i] >= -XXX.XXXXXX and zero_four_lon[i] <= -XXX.XXXXXX)
                    and (zero_four_lat[i] >= XX.XXXXXX and zero_four_lat[i] <= XX.XXXXXX)):
            # Developed Land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp, use the water parameters
        elif ((zero_four_lon[i] >= -XXX.XXXXXX and zero_four_lon[i] <= -XXX.XXXXXX) and
              (zero_four_lat[i] >= XX.XXXXXXX and zero_four_lat[i] <= XX.XXXXXX)):
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
            water, use soil constants
        elif ((zero_four_lon[i] >= -XXX.XXXXXX and zero_four_lon[i] <= -XXXX.XXXXXXX)
              and (zero_four_lat[i] >= XX.XXXXXX and zero_four_lat[i] <= XX.XXXXXX)):
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil
        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land (concrete) constants
    else:
        # Developed land (concrete)
        R = R_concrete
```

```
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Calculate corrected temperature
        corrected_temp_zero_four[i] = B / (numpy.log(R / (Upixel_zero_four[i] + O) + F))

#
    ##########################################################################################################################

# Save To file
outputFileName_zero_four = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                           '/Separated_Hours/Campus_Calibrated/Zero_four_data_calibrated.txt
                               '
outputFile_zero_four = open(outputFileName_zero_four, 'w')
outputFile_zero_four.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected Temp
    [K] \n")

# Save data to file
for i in range(0, len(zero_four_lat)):
    outputFile_zero_four.write("%f \t %f \t %f \n" % (zero_four_lon[i], zero_four_lat[i],
        corrected_temp_zero_four[i]))
outputFile_zero_four.close()

#
    ##########################################################################################################################

# Load data for 04:00 to 08:00
four_eight_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                      'Separated_Hours/Manufacturer_Calibrated/Four_Eight_Data_Processed.txt'

four_eight_data = numpy.genfromtxt(four_eight_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
four_eight_lat_pre_filter = four_eight_data[:, 0]
four_eight_lon_pre_filter = four_eight_data[:, 1]
four_eight_tempK_pre_filter = four_eight_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
four_eight_lat = numpy.zeros((len(four_eight_lat_pre_filter)))
four_eight_lat[:] = numpy.nan
four_eight_lon = numpy.zeros((len(four_eight_lat_pre_filter)))
four_eight_lon[:] = numpy.nan
four_eight_tempK = numpy.zeros((len(four_eight_lat_pre_filter)))
four_eight_tempK[:] = numpy.nan

# Filter out geographic coordinates for the heat maps
for point in range(0, len(four_eight_lat)):
    print(four_eight_lon_pre_filter[point])
```

```
    print(four_eight_lat_pre_filter[point])
    if four_eight_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((four_eight_lon_pre_filter[point] >= -XXX.XXXXXX and four_eight_lon_pre_filter[
        point] <= -XXX.XXXXXX) and
            (four_eight_lat_pre_filter[point] >= XX.XXXXXX and four_eight_lat_pre_filter[point
                ] <= XX.XXXXXX)):
        continue
    elif ((four_eight_lon_pre_filter[point] >= -XXX.XXXXXX and four_eight_lon_pre_filter[
        point] <= -XXX.XXXXXX) and
            (four_eight_lat_pre_filter[point] >= XX.XXXXXX and four_eight_lat_pre_filter[point
                ] <= XX.XXXXXX)):
        continue
    elif ((four_eight_lon_pre_filter[point] >= -XXX.XXXXXX and four_eight_lon_pre_filter[
        point] <= -XXX.XXXXXX) and
            (four_eight_lat_pre_filter[point] >= XX.XXXXXX and four_eight_lat_pre_filter[point
                ] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(four_eight_lat)):
            # Check if index is Nan
            if numpy.isnan(four_eight_lat[index]) == True:
                four_eight_lat[index] = four_eight_lat_pre_filter[point]
                four_eight_lon[index] = four_eight_lon_pre_filter[point]
                four_eight_tempK[index] = four_eight_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(four_eight_lat)):
    if numpy.isnan(four_eight_lat[nan_index]) == True:
        # Remove Nan values from arrays
        four_eight_lat = four_eight_lat[0:nan_index]
        four_eight_lon = four_eight_lon[0:nan_index]
        four_eight_tempK = four_eight_tempK[0:nan_index]
        break

#
    ##########################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_four_eight = numpy.zeros((len(four_eight_lat)))

# Initialize new corrected temperature array
corrected_temp_four_eight = numpy.zeros((len(four_eight_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(four_eight_lat)):
    Upixel_four_eight[i] = (R_flir / (numpy.exp(B_flir / four_eight_tempK[i]) - F_flir)) -
```

```
        O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(four_eight_tempK)):
    # If Longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    #  The waterbodies correspond to developed land parameters
    if (four_eight_lon[i] >= -XXX.XXXX) or (four_eight_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land constants
        if ((four_eight_lon[i] >= -XXX.XXXXXX and four_eight_lon[i] <= -XXX.XXXXXX) and
            (four_eight_lat[i] >= XX.XXXXXX and four_eight_lat[i] <= XX.XXXXXX)) or\
                ((four_eight_lon[i] >= -XXX.XXXXXX and four_eight_lon[i] <= -XXX.XXXXXX) and
                 (four_eight_lat[i] >= XX.XXXXXX and four_eight_lat[i] <= XX.XXXXXX)):
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp, use the water parameters
        elif ((four_eight_lon[i] >= -XXX.XXXXXX and four_eight_lon[i] <= -XXX.XXXXXX) and
              (four_eight_lat[i] >= XX.XXXXXX and four_eight_lat[i] <= XX.XXXXXX)):
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
             water, use soil constants
        elif ((four_eight_lon[i] >= -XXX.XXXXXX and four_eight_lon[i] <= -XXX.XXXXXX) and
              (four_eight_lat[i] >= XX.XXXXXX and four_eight_lat[i] <= XX.XXXXXX)):
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil
        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land constants
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
```

```
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_four_eight[i] = B / (numpy.log(R / (Upixel_four_eight[i] + O) + F))

#
    ##########################################################################################################################

# Save To file
outputFileName_four_eight = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                            '/Separated_Hours/Campus_Calibrated/Four_eight_data_calibrated.
                                txt'
outputFile_four_eight = open(outputFileName_four_eight, 'w')
outputFile_four_eight.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected Temp
    [K] \n")

# Save data to file
for i in range(0, len(four_eight_lat)):
    outputFile_four_eight.write("%f \t %f \t %f \n" % (four_eight_lon[i],
                                                       four_eight_lat[i],
                                                           corrected_temp_four_eight[i]))
outputFile_four_eight.close()

#
    ##########################################################################################################################

# Load data for 08:00 to 12:00
eight_twelve_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                        'Separated_Hours/Manufacturer_Calibrated/Eight_Twelve_Data_Processed.
                            txt'

eight_twelve_data = numpy.genfromtxt(eight_twelve_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
eight_twelve_lat_pre_filter = eight_twelve_data[:, 0]
eight_twelve_lon_pre_filter = eight_twelve_data[:, 1]
eight_twelve_tempK_pre_filter = eight_twelve_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
eight_twelve_lat = numpy.zeros((len(eight_twelve_lat_pre_filter)))
eight_twelve_lat[:] = numpy.nan
eight_twelve_lon = numpy.zeros((len(eight_twelve_lat_pre_filter)))
eight_twelve_lon[:] = numpy.nan
eight_twelve_tempK = numpy.zeros((len(eight_twelve_lat_pre_filter)))
eight_twelve_tempK[:] = numpy.nan

# Filter out geographic coordinates for the heat maps
for point in range(0, len(eight_twelve_lat)):
    print(eight_twelve_lon_pre_filter[point])
```

```
    print(eight_twelve_lat_pre_filter[point])
    if eight_twelve_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((eight_twelve_lon_pre_filter[point] >= -XXX.XXXXXX and eight_twelve_lon_pre_filter
        [point] <= -XXX.XXXXXX)
          and (eight_twelve_lat_pre_filter[point] >= XX.XXXXXX and
              eight_twelve_lat_pre_filter[point] <= XX.XXXXXXX)):
        continue
    elif ((eight_twelve_lon_pre_filter[point] >= -XXX.XXXXXXX and
        eight_twelve_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (eight_twelve_lat_pre_filter[point] >= XX.XXXXXX and
              eight_twelve_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((eight_twelve_lon_pre_filter[point] >= -XXX.XXXXXXX and
        eight_twelve_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (eight_twelve_lat_pre_filter[point] >= XX.XXXXXXX and
              eight_twelve_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(eight_twelve_lat)):
            # Check if index is Nan
            if numpy.isnan(eight_twelve_lat[index]) == True:
                eight_twelve_lat[index] = eight_twelve_lat_pre_filter[point]
                eight_twelve_lon[index] = eight_twelve_lon_pre_filter[point]
                eight_twelve_tempK[index] = eight_twelve_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(eight_twelve_lat)):
    if numpy.isnan(eight_twelve_lat[nan_index]) == True:
        # Remove Nan values from arrays
        eight_twelve_lat = eight_twelve_lat[0:nan_index]
        eight_twelve_lon = eight_twelve_lon[0:nan_index]
        eight_twelve_tempK = eight_twelve_tempK[0:nan_index]
        break

#
    ##########################################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_eight_twelve = numpy.zeros((len(eight_twelve_lat)))

# Initialize new corrected temperature array
corrected_temp_eight_twelve = numpy.zeros((len(eight_twelve_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(eight_twelve_lat)):
    Upixel_eight_twelve[i] = (R_flir / (numpy.exp(B_flir / eight_twelve_tempK[i]) - F_flir))
```

```
        - O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(eight_twelve_tempK)):
    # If longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    # The waterbodies correspond to developed land parameters
    if (eight_twelve_lon[i] >= -XXX.XXXXX) or (eight_twelve_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land
        if ((eight_twelve_lon[i] >= -XXX.XXXXXX and eight_twelve_lon[i] <= -XXX.XXXXXX) and
            (eight_twelve_lat[i] >= XX.XXXXXX and eight_twelve_lat[i] <= XX.XXXXXX)) or\
                ((eight_twelve_lon[i] >= -XXX.XXXXXX and eight_twelve_lon[i] <= -XXX.XXXXXX)
                    and
                 (eight_twelve_lat[i] >= XX.XXXXXX and eight_twelve_lat[i] <= XX.XXXXXX)):
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp
        elif ((eight_twelve_lon[i] >= -XXX.XXXXXX and eight_twelve_lon[i] <= -XXX.XXXXXX)
            and
              (eight_twelve_lat[i] >= XX.XXXXXXX and eight_twelve_lat[i] <= XX.XXXXXX)):
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
             water, use soil constants
        elif ((eight_twelve_lon[i] >= -XXX.XXXXXX and eight_twelve_lon[i] <= -XXX.XXXXXX)
            and
              (eight_twelve_lat[i] >= XX.XXXXXXX and eight_twelve_lat[i] <= XX.XXXXXXXX)):
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil

        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land (concrete)
    else:
```

```
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_eight_twelve[i] = B / (numpy.log(R / (Upixel_eight_twelve[i] + O) + F))

#
    ##########################################################################################################################

# Save To file
outputFileName_eight_twelve = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                              '/Separated_Hours/Campus_Calibrated/Eight_twelve_data_calibrated.
                                  txt'
outputFile_eight_twelve = open(outputFileName_eight_twelve, 'w')
outputFile_eight_twelve.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(eight_twelve_lat)):
    outputFile_eight_twelve.write("%f \t %f \t %f \n" % (eight_twelve_lon[i],
        eight_twelve_lat[i],
                                                         corrected_temp_eight_twelve[i]))
outputFile_eight_twelve.close()

#
    ##########################################################################################################################

# Load data for 12:00 to 16:00
twelve_sixteen_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                      'Separated_Hours/Manufacturer_Calibrated/Twelve_Sixteen_Data_Processed.
                          txt'

twelve_sixteen_data = numpy.genfromtxt(twelve_sixteen_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
twelve_sixteen_lat_pre_filter = twelve_sixteen_data[:, 0]
twelve_sixteen_lon_pre_filter = twelve_sixteen_data[:, 1]
twelve_sixteen_tempK_pre_filter = twelve_sixteen_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
twelve_sixteen_lat = numpy.zeros((len(twelve_sixteen_lat_pre_filter)))
twelve_sixteen_lat[:] = numpy.nan
twelve_sixteen_lon = numpy.zeros((len(twelve_sixteen_lat_pre_filter)))
twelve_sixteen_lon[:] = numpy.nan
twelve_sixteen_tempK = numpy.zeros((len(twelve_sixteen_lat_pre_filter)))
twelve_sixteen_tempK[:] = numpy.nan
```

```
# Filter out geographic coordinates for the heat maps
for point in range(0, len(twelve_sixteen_lat)):
    print(twelve_sixteen_lon_pre_filter[point])
    print(twelve_sixteen_lat_pre_filter[point])
    if twelve_sixteen_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((twelve_sixteen_lon_pre_filter[point] >= -XXX.XXXXXXX and
           twelve_sixteen_lon_pre_filter[point] <= -XXX.XXXXXXX) and
          (twelve_sixteen_lat_pre_filter[point] >= XX.XXXXX and
              twelve_sixteen_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((twelve_sixteen_lon_pre_filter[point] >= -XXX.XXXXXX and
        twelve_sixteen_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (twelve_sixteen_lat_pre_filter[point] >= XX.XXXXXX and
               twelve_sixteen_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((twelve_sixteen_lon_pre_filter[point] >= -XXX.XXXXXX and
        twelve_sixteen_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (twelve_sixteen_lat_pre_filter[point] >= XX.XXXXXXX and
               twelve_sixteen_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(twelve_sixteen_lat)):
            # Check if index is Nan
            if numpy.isnan(twelve_sixteen_lat[index]) == True:
                twelve_sixteen_lat[index] = twelve_sixteen_lat_pre_filter[point]
                twelve_sixteen_lon[index] = twelve_sixteen_lon_pre_filter[point]
                twelve_sixteen_tempK[index] = twelve_sixteen_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(twelve_sixteen_lat)):
    if numpy.isnan(twelve_sixteen_lat[nan_index]) == True:
        # Remove Nan values from arrays
        twelve_sixteen_lat = twelve_sixteen_lat[0:nan_index]
        twelve_sixteen_lon = twelve_sixteen_lon[0:nan_index]
        twelve_sixteen_tempK = twelve_sixteen_tempK[0:nan_index]
        break

#
    ##########################################################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_twelve_sixteen = numpy.zeros((len(twelve_sixteen_lat)))

# Initialize new corrected temperature array
corrected_temp_twelve_sixteen = numpy.zeros((len(twelve_sixteen_lon)))
```

```
# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(twelve_sixteen_lat)):
    Upixel_twelve_sixteen[i] = (R_flir / (numpy.exp(B_flir / twelve_sixteen_tempK[i]) -
        F_flir)) - O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(twelve_sixteen_tempK)):
    # If Longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid carbon waste/water areas).
    # The waterbodies correspond to developed land (concrete) parameters
    if (twelve_sixteen_lon[i] >= -XXX.XXXX) or (twelve_sixteen_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land parameters
        if ((twelve_sixteen_lon[i] >= -XXX.XXXXXX and twelve_sixteen_lon[i] <= -XXX.XXXXXX)
            and
            (twelve_sixteen_lat[i] >= XX.XXXXXX and twelve_sixteen_lat[i] <= XX.XXXXXX)) or\
                ((twelve_sixteen_lon[i] >= -XXX.XXXXXX and twelve_sixteen_lon[i] <= -XXX.
                    XXXXXX) and
                 (twelve_sixteen_lat[i] >= XX.XXXXXX and twelve_sixteen_lat[i] <= XX.XXXXXXX
                     )):
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp
        elif ((twelve_sixteen_lon[i] >= -XXX.XXXXXX and twelve_sixteen_lon[i] <= -XXX.XXXXXX
            ) and
              (twelve_sixteen_lat[i] >= XX.XXXXXXX and twelve_sixteen_lat[i] <= XX.XXXXXXX))
                  :
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
             water, use soil constants
        elif ((twelve_sixteen_lon[i] >= -XXX.XXXXXX and twelve_sixteen_lon[i] <= -XXX.XXXXXX
            ) and
              (twelve_sixteen_lat[i] >= XX.XXXXXXX and twelve_sixteen_lat[i] <= XX.XXXXXXX))
                  :
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil

        # For all other areas, use developed land constants
        else:
```

```
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land constants
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_twelve_sixteen[i] = B / (numpy.log(R / (Upixel_twelve_sixteen[i] + O) + F
        ))

#
    ##########################################################################################################################

# Save To file
outputFileName_twelve_sixteen = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                            '/Separated_Hours/Campus_Calibrated/
                                Twelve_sixteen_data_calibrated.txt'
outputFile_twelve_sixteen = open(outputFileName_twelve_sixteen, 'w')
outputFile_twelve_sixteen.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(twelve_sixteen_lat)):
    outputFile_twelve_sixteen.write("%f \t %f \t %f \n" % (twelve_sixteen_lon[i],
        twelve_sixteen_lat[i],
                                                           corrected_temp_twelve_sixteen[i])
                                                               )
outputFile_twelve_sixteen.close()

#
    ##########################################################################################################################

# Load data for 16:00 to 20:00
sixteen_twenty_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                      'Separated_Hours/Manufacturer_Calibrated/Sixteen_Twenty_Data_Processed.
                          txt'

sixteen_twenty_data = numpy.genfromtxt(sixteen_twenty_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
sixteen_twenty_lat_pre_filter = sixteen_twenty_data[:, 0]
sixteen_twenty_lon_pre_filter = sixteen_twenty_data[:, 1]
```

```
sixteen_twenty_tempK_pre_filter = sixteen_twenty_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
sixteen_twenty_lat = numpy.zeros((len(sixteen_twenty_lat_pre_filter)))
sixteen_twenty_lat[:] = numpy.nan
sixteen_twenty_lon = numpy.zeros((len(sixteen_twenty_lat_pre_filter)))
sixteen_twenty_lon[:] = numpy.nan
sixteen_twenty_tempK = numpy.zeros((len(sixteen_twenty_lat_pre_filter)))
sixteen_twenty_tempK[:] = numpy.nan

# Filter out geographic coordinates for the heat maps
for point in range(0, len(sixteen_twenty_lat)):
    print(sixteen_twenty_lon_pre_filter[point])
    print(sixteen_twenty_lat_pre_filter[point])
    if sixteen_twenty_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((sixteen_twenty_lon_pre_filter[point] >= -XXX.XXXXXX and
        sixteen_twenty_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (sixteen_twenty_lat_pre_filter[point] >= XX.XXXXXXX and
               sixteen_twenty_lat_pre_filter[point] <= XX.XXXXXXX)):
        continue
    elif ((sixteen_twenty_lon_pre_filter[point] >= -XXX.XXXXXXX and
        sixteen_twenty_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (sixteen_twenty_lat_pre_filter[point] >= XX.XXXXXX and
               sixteen_twenty_lat_pre_filter[point] <= XX.XXXXXXX)):
        continue
    elif ((sixteen_twenty_lon_pre_filter[point] >= -XXX.XXXXXX and
        sixteen_twenty_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (sixteen_twenty_lat_pre_filter[point] >= XX.XXXXXX and
               sixteen_twenty_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(sixteen_twenty_lat)):
            # Check if index is Nan
            if numpy.isnan(sixteen_twenty_lat[index]) == True:
                sixteen_twenty_lat[index] = sixteen_twenty_lat_pre_filter[point]
                sixteen_twenty_lon[index] = sixteen_twenty_lon_pre_filter[point]
                sixteen_twenty_tempK[index] = sixteen_twenty_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(sixteen_twenty_lat)):
    if numpy.isnan(sixteen_twenty_lat[nan_index]) == True:
        # Remove Nan values from arrays
        sixteen_twenty_lat = sixteen_twenty_lat[0:nan_index]
        sixteen_twenty_lon = sixteen_twenty_lon[0:nan_index]
        sixteen_twenty_tempK = sixteen_twenty_tempK[0:nan_index]
        break

#
    ##########################################################################################################################
```

```
# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_sixteen_twenty = numpy.zeros((len(sixteen_twenty_lat)))

# Initialize new corrected temperature array
corrected_temp_sixteen_twenty = numpy.zeros((len(sixteen_twenty_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(sixteen_twenty_lat)):
    Upixel_sixteen_twenty[i] = (R_flir / (numpy.exp(B_flir / sixteen_twenty_tempK[i]) -
        F_flir)) - O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(sixteen_twenty_tempK)):
    # If longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    # The waterbodies correspond to developed land (concrete) parameters
    if (sixteen_twenty_lon[i] >= -XXX.XXXX) or (sixteen_twenty_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land (concrete) parameters
        if ((sixteen_twenty_lon[i] >= -XXX.XXXXXX and sixteen_twenty_lon[i] <= -XXX.XXXXXX)
            and
            (sixteen_twenty_lat[i] >= XX.XXXXXX and sixteen_twenty_lat[i] <= XX.XXXXXX)) or\
                ((sixteen_twenty_lon[i] >= -XXX.XXXXXX and sixteen_twenty_lon[i] <= -XXX.
                    XXXXXX) and
                 (sixteen_twenty_lat[i] >= XX.XXXXXX and sixteen_twenty_lat[i] <= XX.XXXXXX)
                     ):
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp
        elif ((sixteen_twenty_lon[i] >= -XXX.XXXXXX and sixteen_twenty_lon[i] <= -XXX.XXXXXX
            ) and
              (sixteen_twenty_lat[i] >= XX.XXXXXX and sixteen_twenty_lat[i] <= XX.XXXXXX)):
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
             water, use soil constants
        elif ((sixteen_twenty_lon[i] >= -XXX.XXXXXX and sixteen_twenty_lon[i] <= -XXX.XXXXXX
            ) and
              (sixteen_twenty_lat[i] >= XX.XXXXXX and sixteen_twenty_lat[i] <= XX.XXXXXXX)):
```

```
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil
        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land constants
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_sixteen_twenty[i] = B / (numpy.log(R / (Upixel_sixteen_twenty[i] + O) + F
        ))

#
    ##########################################################################################################################

# Save To file
outputFileName_sixteen_twenty = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                            '/Separated_Hours/Campus_Calibrated/
                                Sixteen_twenty_data_calibrated.txt'
outputFile_sixteen_twenty = open(outputFileName_sixteen_twenty, 'w')
outputFile_sixteen_twenty.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(sixteen_twenty_lat)):
    outputFile_sixteen_twenty.write("%f \t %f \t %f \n" % (sixteen_twenty_lon[i],
        sixteen_twenty_lat[i],
                                                           corrected_temp_sixteen_twenty[i])
                                                               )
outputFile_sixteen_twenty.close()

#
    ##########################################################################################################################

# Load data for 20:00 to 24:00
twenty_twentyfour_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                     'Separated_Hours/Manufacturer_Calibrated/
```

```python
                    Twenty_Twentyfour_Data_Processed.txt'

twenty_twentyfour_data = numpy.genfromtxt(twenty_twentyfour_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
twenty_twentyfour_lat_pre_filter = twenty_twentyfour_data[:, 0]
twenty_twentyfour_lon_pre_filter = twenty_twentyfour_data[:, 1]
twenty_twentyfour_tempK_pre_filter = twenty_twentyfour_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
twenty_twentyfour_lat = numpy.zeros((len(twenty_twentyfour_lat_pre_filter)))
twenty_twentyfour_lat[:] = numpy.nan
twenty_twentyfour_lon = numpy.zeros((len(twenty_twentyfour_lat_pre_filter)))
twenty_twentyfour_lon[:] = numpy.nan
twenty_twentyfour_tempK = numpy.zeros((len(twenty_twentyfour_lat_pre_filter)))
twenty_twentyfour_tempK[:] = numpy.nan

# Filter out geographic coordinates for the heat maps
for point in range(0, len(twenty_twentyfour_lat)):
    print(twenty_twentyfour_lon_pre_filter[point])
    print(twenty_twentyfour_lat_pre_filter[point])
    if twenty_twentyfour_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((twenty_twentyfour_lon_pre_filter[point] >= -XXX.XXXXXX and
           twenty_twentyfour_lon_pre_filter[point] <= -XXX.XXXXXX) and
          (twenty_twentyfour_lat_pre_filter[point] >= XX.XXXXXX and
           twenty_twentyfour_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((twenty_twentyfour_lon_pre_filter[point] >= -XXX.XXXXXX and
           twenty_twentyfour_lon_pre_filter[point] <= -XXX.XXXXXX) and
          (twenty_twentyfour_lat_pre_filter[point] >= XX.XXXXXX and
           twenty_twentyfour_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((twenty_twentyfour_lon_pre_filter[point] >= -XXX.XXXXXX and
           twenty_twentyfour_lon_pre_filter[point] <= -XXX.XXXXXX) and
          (twenty_twentyfour_lat_pre_filter[point] >= XX.XXXXXX and
           twenty_twentyfour_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    else:
        for index in range(0, len(twenty_twentyfour_lat)):
            # Check if index is Nan
            if numpy.isnan(twenty_twentyfour_lat[index]) == True:
                twenty_twentyfour_lat[index] = twenty_twentyfour_lat_pre_filter[point]
                twenty_twentyfour_lon[index] = twenty_twentyfour_lon_pre_filter[point]
                twenty_twentyfour_tempK[index] = twenty_twentyfour_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(twenty_twentyfour_lat)):
    if numpy.isnan(twenty_twentyfour_lat[nan_index]) == True:
        # Remove Nan values from good arrays
```

```
        twenty_twentyfour_lat = twenty_twentyfour_lat[0:nan_index]
        twenty_twentyfour_lon = twenty_twentyfour_lon[0:nan_index]
        twenty_twentyfour_tempK = twenty_twentyfour_tempK[0:nan_index]
        break

#
    ##########################################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_twenty_twentyfour = numpy.zeros((len(twenty_twentyfour_lat)))

# Initialize new corrected temperature array
corrected_temp_twenty_twentyfour = numpy.zeros((len(twenty_twentyfour_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(twenty_twentyfour_lat)):
    Upixel_twenty_twentyfour[i] = (R_flir / (numpy.exp(B_flir / twenty_twentyfour_tempK[i])
        - F_flir)) - O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(twenty_twentyfour_tempK)):
    # If Longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    # The waterbodies correspond to developed land (concrete) parameters
    if (twenty_twentyfour_lon[i] >= -XXX.XXXX) or (twenty_twentyfour_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land parameters (concrete)
        if ((twenty_twentyfour_lon[i] >= -XXX.XXXXXX and twenty_twentyfour_lon[i] <= -XXX.
            XXXXXX) and
            (twenty_twentyfour_lat[i] >= XX.XXXXXX and twenty_twentyfour_lat[i] <= XX.XXXXXX
                )) or\
                ((twenty_twentyfour_lon[i] >= -XXX.XXXXXX and twenty_twentyfour_lon[i] <= -
                    XXX.XXXXXX) and
                 (twenty_twentyfour_lat[i] >= XX.XXXXXX and twenty_twentyfour_lat[i] <= XX.
                     XXXXXXX)):
            # Developed Land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp
        elif ((twenty_twentyfour_lon[i] >= -XXX.XXXXXX and twenty_twentyfour_lon[i] <= -XXX.
            XXXXXX) and
              (twenty_twentyfour_lat[i] >= XX.XXXXXX and twenty_twentyfour_lat[i] <= XX.
                  XXXXXX)):
            # Water
            R = R_water
```

```
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
            water, use soil constants
        elif ((twenty_twentyfour_lon[i] >= -XXX.XXXXXXX and twenty_twentyfour_lon[i] <= -XXX
            .XXXXXX) and
              (twenty_twentyfour_lat[i] >= XX.XXXXXX and twenty_twentyfour_lat[i] <= XX.
                  XXXXXX)):
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil
        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land parameters (concrete)
    else:
        # Concrete
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_twenty_twentyfour[i] = B / (numpy.log(R / (Upixel_twenty_twentyfour[i] +
        O) + F))

#
    ##########################################################################################################################

# Save To file
outputFileName_twenty_twentyfour = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data' \
                           '/Separated_Hours/Campus_Calibrated/
                               Twenty_twentyfour_data_calibrated.txt'
outputFile_twenty_twentyfour = open(outputFileName_twenty_twentyfour, 'w')
outputFile_twenty_twentyfour.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use
    Corrected Temp [K] \n")

# Save data to file
for i in range(0, len(twenty_twentyfour_lat)):
    outputFile_twenty_twentyfour.write("%f \t %f \t %f \n" % (twenty_twentyfour_lon[i],
        twenty_twentyfour_lat[i],
                                                              corrected_temp_twenty_twentyfour
```

```
                                                                                    [i]))
outputFile_twenty_twentyfour.close()

#
    ##########################################################################################################

# Load data for May 24 FLIR/MODIS comparison
# This data was created from TempExtract_May_2018_Mine_Campaign.py, where only images
    recorded
#  on May 24, 2018 were processed
May_24_SA_FLIR_filename = '/export/home/users/username/Documents/DG_Temp' \
                          '/Mining_Facility_2018/Processed_Data/Separated_Hours/
                              May24_Data_Processed.txt'

May_24_SA_FLIR_data = numpy.genfromtxt(May_24_SA_FLIR_filename, usecols=[6, 7, 10])

# Consider emissivity does not equal 1
May_24_SA_FLIR_lat_pre_filter = May_24_SA_FLIR_data[:, 0]
May_24_SA_FLIR_lon_pre_filter = May_24_SA_FLIR_data[:, 1]
May_24_SA_FLIR_tempK_pre_filter = May_24_SA_FLIR_data[:, 2]

# Filter the data based on known geographic coordinates and correct temperature accordingly
# Initialize new arrays to be equal to Nan
May_24_SA_FLIR_lat = numpy.zeros((len(May_24_SA_FLIR_lat_pre_filter)))
May_24_SA_FLIR_lat[:] = numpy.nan
May_24_SA_FLIR_lon = numpy.zeros((len(May_24_SA_FLIR_lat_pre_filter)))
May_24_SA_FLIR_lon[:] = numpy.nan
May_24_SA_FLIR_tempK = numpy.zeros((len(May_24_SA_FLIR_lat_pre_filter)))
May_24_SA_FLIR_tempK[:] = numpy.nan

# Filter out geographic coordinates for the heat maps
for point in range(0, len(May_24_SA_FLIR_lat)):
    print(May_24_SA_FLIR_lon_pre_filter[point])
    print(May_24_SA_FLIR_lat_pre_filter[point])
    if May_24_SA_FLIR_lon_pre_filter[point] >= -XXX.XXXXXX:
        continue
    elif ((May_24_SA_FLIR_lon_pre_filter[point] >= -XXX.XXXXXX and
        May_24_SA_FLIR_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (May_24_SA_FLIR_lat_pre_filter[point] >= XX.XXXXXX and
                May_24_SA_FLIR_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((May_24_SA_FLIR_lon_pre_filter[point] >= -XXX.XXXXXX and
        May_24_SA_FLIR_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (May_24_SA_FLIR_lat_pre_filter[point] >= XX.XXXXXX and
                May_24_SA_FLIR_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    elif ((May_24_SA_FLIR_lon_pre_filter[point] >= -XXX.XXXXXX and
        May_24_SA_FLIR_lon_pre_filter[point] <= -XXX.XXXXXX)
          and (May_24_SA_FLIR_lat_pre_filter[point] >= XX.XXXXXX and
                May_24_SA_FLIR_lat_pre_filter[point] <= XX.XXXXXX)):
        continue
    else:
```

```
        for index in range(0, len(May_24_SA_FLIR_lat)):
            # Check if index is Nan
            if numpy.isnan(May_24_SA_FLIR_lat[index]) == True:
                May_24_SA_FLIR_lat[index] = May_24_SA_FLIR_lat_pre_filter[point]
                May_24_SA_FLIR_lon[index] = May_24_SA_FLIR_lon_pre_filter[point]
                May_24_SA_FLIR_tempK[index] = May_24_SA_FLIR_tempK_pre_filter[point]
                break

# Identify index where Nan starts
for nan_index in range(0, len(May_24_SA_FLIR_lat)):
    if numpy.isnan(May_24_SA_FLIR_lat[nan_index]) == True:
        # Remove Nan values from arrays
        May_24_SA_FLIR_lat = May_24_SA_FLIR_lat[0:nan_index]
        May_24_SA_FLIR_lon = May_24_SA_FLIR_lon[0:nan_index]
        May_24_SA_FLIR_tempK = May_24_SA_FLIR_tempK[0:nan_index]
        break

#
    ##########################################################################################################################

# Apply Temperature Correction Constants
# Calculate Upixel using Horny, 2003 formula
#  https://www.sciencedirect.com/science/article/pii/S1350449502001834?via%3Dihub

# Initialize Upixel
Upixel_May_24_SA_FLIR = numpy.zeros((len(May_24_SA_FLIR_lat)))

# Initialize new corrected temperature array
corrected_temp_May_24_SA_FLIR = numpy.zeros((len(May_24_SA_FLIR_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(May_24_SA_FLIR_lat)):
    Upixel_May_24_SA_FLIR[i] = (R_flir / (numpy.exp(B_flir / May_24_SA_FLIR_tempK[i]) -
        F_flir)) - O_flir

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(May_24_SA_FLIR_tempK)):
    # If longitude is greater than -XXX.XXXX or longitude is less than -XXX.XXXX, use soil
        parameters
    #  (except the points that overlay the solid waste carbon/water areas).
    # The waterbodies correspond to developed land (concrete) parameters
    if (May_24_SA_FLIR_lon[i] >= -XXX.XXXX) or (May_24_SA_FLIR_lon[i] <= -XXX.XXXX):
        # Solid waste carbon coordinates, use developed land (concrete) parameters
        if ((May_24_SA_FLIR_lon[i] >= -XXX.XXXXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXXXX)
            and
            (May_24_SA_FLIR_lat[i] >= XX.XXXXXX and May_24_SA_FLIR_lat[i] <= XX.XXXXXX)) or\
                ((May_24_SA_FLIR_lon[i] >= -XXX.XXXXXX and May_24_SA_FLIR_lon[i] <= -XXX.
                    XXXXXX) and
                 (May_24_SA_FLIR_lat[i] >= XX.XXXXXX and May_24_SA_FLIR_lat[i] <= XX.XXXXXX)
                     ):
            # Developed Land (concrete)
            R = R_concrete
```

```
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Water body near the camp
        elif ((May_24_SA_FLIR_lon[i] >= -XXX.XXXXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXXXX
            ) and
              (May_24_SA_FLIR_lat[i] >= XX.XXXXXX and May_24_SA_FLIR_lat[i] <= XX.XXXXXX)):
            # Water
            R = R_water
            B = B_water
            O = O_water
            F = F_water

        # For the following area, including the Berm, west of the mine, and east of the open
             water, use soil constants
        elif ((May_24_SA_FLIR_lon[i] >= -XXX.XXXXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXXXX
            ) and
              (May_24_SA_FLIR_lat[i] >= XX.XXXXXX and May_24_SA_FLIR_lat[i] <= XX.XXXXXX)):
            # Soil
            R = R_soil
            B = B_soil
            O = O_soil
            F = F_soil
        # For all other areas, use developed land constants
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

    # Tailings Pond, use developed land (concrete) parameters
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_May_24_SA_FLIR[i] = B / (numpy.log(R / (Upixel_May_24_SA_FLIR[i] + O) + F
        ))

#
    ##########################################################################################################################

# Save To file
outputFileName_May_24_SA_FLIR = '/export/home/users/rbyerlay/Documents/
    Geoscientific_Information_Manoj_Paper/' \
                                'GI_Review_CurveFit/May_24_FLIR_data_calibrated.txt'
outputFile_May_24_SA_FLIR = open(outputFileName_May_24_SA_FLIR, 'w')
```

```
outputFile_May_24_SA_FLIR.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(May_24_SA_FLIR_lat)):
    outputFile_May_24_SA_FLIR.write("%f \t %f \t %f \n" % (May_24_SA_FLIR_lon[i],
        May_24_SA_FLIR_lat[i],
                                                           corrected_temp_May_24_SA_FLIR[i])
                                                               )
outputFile_May_24_SA_FLIR.close()
```

A.2.7 Surface Temperature Map and Boxplots

```
# Current as of October 18, 2019
# Plots Diurnal ST as Maps and Boxplots
# Plots for May 24, 2018 (FLIR, MODIS and Percentage Error/Absolute error)

import numpy
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

#
    ##########################################################################################################################

# Load data for 00:00 to 04:00
zero_four_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                     'Separated_Hours/Campus_Calibrated/Zero_four_data_calibrated.txt'

zero_four_data = numpy.genfromtxt(zero_four_filename)

zero_four_lon = zero_four_data[:,0]
zero_four_lat = zero_four_data[:,1]
zero_four_tempK = zero_four_data[:,2]

#
    ##########################################################################################################################

# Load in data for 04:00 to 08:00
four_eight_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                      'Separated_Hours/Campus_Calibrated/Four_eight_data_calibrated.txt'

four_eight_data = numpy.genfromtxt(four_eight_filename)

four_eight_lon = four_eight_data[:,0]
four_eight_lat = four_eight_data[:,1]
four_eight_tempK = four_eight_data[:,2]

#
    ##########################################################################################################################
```

```
# Load in data for 08:00 to 12:00
eight_twelve_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    Processed_Data/' \
                        'Separated_Hours/Campus_Calibrated/Eight_twelve_data_calibrated.txt'

eight_twelve_data = numpy.genfromtxt(eight_twelve_filename)

eight_twelve_lon = eight_twelve_data[:,0]
eight_twelve_lat = eight_twelve_data[:,1]
eight_twelve_tempK = eight_twelve_data[:,2]

#
    ##########################################################################################################################

# # Load in data for 12:00 to 16:00
twelve_sixteen_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                        'Separated_Hours/Campus_Calibrated/Twelve_sixteen_data_calibrated.txt'

twelve_sixteen_data = numpy.genfromtxt(twelve_sixteen_filename)

twelve_sixteen_lon = twelve_sixteen_data[:,0]
twelve_sixteen_lat = twelve_sixteen_data[:,1]
twelve_sixteen_tempK = twelve_sixteen_data[:,2]

#
    ##########################################################################################################################

# Load in data for 16:00 to 20:00
sixteen_twenty_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                        'Separated_Hours/Campus_Calibrated/Sixteen_twenty_data_calibrated.txt'

sixteen_twenty_data = numpy.genfromtxt(sixteen_twenty_filename)

sixteen_twenty_lon = sixteen_twenty_data[:,0]
sixteen_twenty_lat = sixteen_twenty_data[:,1]
sixteen_twenty_tempK = sixteen_twenty_data[:,2]

#
    ##########################################################################################################################

# Load in data for 20:00 to 24:00
twenty_twentyfour_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                        'Separated_Hours/Campus_Calibrated/Twenty_twentyfour_data_calibrated.
                            txt'

twenty_twentyfour_data = numpy.genfromtxt(twenty_twentyfour_filename, delimiter=',')

twenty_twentyfour_lon = twenty_twentyfour_data[:,0]
```

```
twenty_twentyfour_lat = twenty_twentyfour_data[:,1]
twenty_twentyfour_tempK = twenty_twentyfour_data[:,2]

#
    ##########################################################################################################################

# For MODIS and Relative/Absolute error data
# For Relative Error
# May_24_SA_MODIS_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/
# Separated_Hours/Campus_Calibrated/May_24_MODIS_data_calibrated.txt'
# For Absolute Error
May_24_SA_MODIS_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/Processed_Data/' \
                           'Separated_Hours/Campus_Calibrated/
                               MODIS_LST_May_24_Calibrated_Absolute_Error.txt'
May_24_SA_MODIS_data = numpy.genfromtxt(May_24_SA_MODIS_filename, delimiter=',')

# MODIS, FLIR, and Relative/Absolute error
May_24_SA_MODIS_lon = May_24_SA_MODIS_data[:,0]
May_24_SA_MODIS_lat = May_24_SA_MODIS_data[:,1]

# For FLIR
May_24_SA_FLIR_lat = May_24_SA_MODIS_lat
May_24_SA_FLIR_lon = May_24_SA_MODIS_lon
May_24_SA_FLIR_tempK = May_24_SA_MODIS_data[:,2]

May_24_SA_MODIS_tempK = May_24_SA_MODIS_data[:,4]
May_24_SA_PE = May_24_SA_MODIS_data[:,5]

#
    ##########################################################################################################################

# Temperature distribution boundaries
# Jan.19/19 Newly chosen boundaries
Latmin = XX.XXXXXX
Latmax = XX.XXXXXX

Lonmax = -XXX.XXXXXX
Lonmin = -XXX.XXXXXX

#
    ##########################################################################################################################

# Choose the number of horizontal and vertical "bins" to plot. These variables correspond to
    horizontal
#  and spatial resolution respectively
# For 2000 m (latitude) by 2500m (longitude) resolution
# nLat = 10
# nLon = 10

# For 1000 m resolution
```

```
nLat = 20
nLon = 25

# For 500m resolution
# nLat = 40
# nLon = 48

# For 100m resolution
# nLat = 200
# nLon = 250

# For 200m resolution
# nLat = 100
# nLon = 125

#
    ##########################################################################################################################################

# Create temperature array for each time interval
# 00:00 to 04:00
TMatrix_zero_four = numpy.zeros(((nLat+1), (nLon+1), (len(zero_four_lat))))
TMatrix_zero_four[:] = numpy.nan

# 04:00 to 08:00
TMatrix_four_eight = numpy.zeros(((nLat+1), (nLon+1), (len(four_eight_lat))))
TMatrix_four_eight[:] = numpy.nan

# 08:00 to 12:00
TMatrix_eight_twelve = numpy.zeros(((nLat+1), (nLon+1), (len(eight_twelve_lat))))
TMatrix_eight_twelve[:] = numpy.nan

# 12:00 to 16:00
TMatrix_twelve_sixteen = numpy.zeros(((nLat+1), (nLon+1), (len(twelve_sixteen_lat))))
TMatrix_twelve_sixteen[:] = numpy.nan

# 16:00 to 20:00
TMatrix_sixteen_twenty = numpy.zeros(((nLat+1), (nLon+1), (len(sixteen_twenty_lat))))
TMatrix_sixteen_twenty[:] = numpy.nan

# 20:00 to 24:00
TMatrix_twenty_twentyfour = numpy.zeros(((nLat+1), (nLon+1), (len(twenty_twentyfour_lat))))
TMatrix_twenty_twentyfour[:] = numpy.nan

# For May 24 FLIR
TMatrix_May_24_SA_FLIR = numpy.zeros(((nLat+1), (nLon+1), (len(May_24_SA_FLIR_lat))))
TMatrix_May_24_SA_FLIR[:] = numpy.nan

# For May 24 MODIS
TMatrix_May_24_SA_MODIS = numpy.zeros(((nLat+1), (nLon+1), (len(May_24_SA_MODIS_lat))))
TMatrix_May_24_SA_MODIS[:] = numpy.nan

# For May 24 Percentage Error or Absolute Error
```

```
PE_Matrix_May_24_SA = numpy.zeros(((nLat+1), (nLon+1), (len(May_24_SA_MODIS_lat))))
PE_Matrix_May_24_SA[:] = numpy.nan

#
    ##########################################################################################################################

# Create median temperature array for each interval
# For 00:00 to 04:00
TMatrix_zero_four_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_zero_four_median[:] = numpy.nan

# For 04:00 to 08:00
TMatrix_four_eight_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_four_eight_median[:] = numpy.nan

# For 08:00 to 12:00
TMatrix_eight_twelve_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_eight_twelve_median[:] = numpy.nan

# For 12:00 to 16:00
TMatrix_twelve_sixteen_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_twelve_sixteen_median[:] = numpy.nan

# For 16:00 to 20:00
TMatrix_sixteen_twenty_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_sixteen_twenty_median[:] = numpy.nan

# For 20:00 to 24:00
TMatrix_twenty_twentyfour_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_twenty_twentyfour_median[:] = numpy.nan

# For May 24 FLIR
TMatrix_May_24_SA_FLIR_Median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_May_24_SA_FLIR_Median[:] = numpy.nan

# For May 24 MODIS
TMatrix_May_24_SA_MODIS_Median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_May_24_SA_MODIS_Median[:] = numpy.nan

# For May 24 Percentage Error or Absolute Error
PE_Matrix_May_24_SA_Median = numpy.zeros(((nLat+1), (nLon+1)))
PE_Matrix_May_24_SA_Median[:] = numpy.nan

#
    ##########################################################################################################################

# Initialize Boxplot arrays for Mine and Pond for each time interval
# For 00:00 to 04:00
Boxplot_mine_temps_zero_four = numpy.zeros(len(zero_four_lat))
Boxplot_mine_temps_zero_four[:] = numpy.nan

Boxplot_pond_temps_zero_four = numpy.zeros(len(zero_four_lat))
```

```
Boxplot_pond_temps_zero_four[:] = numpy.nan

# For 04:00 to 08:00
Boxplot_mine_temps_four_eight = numpy.zeros(len(four_eight_lat))
Boxplot_mine_temps_four_eight[:] = numpy.nan

Boxplot_pond_temps_four_eight = numpy.zeros(len(four_eight_lat))
Boxplot_pond_temps_four_eight[:] = numpy.nan

# For 08:00 to 12:00
Boxplot_mine_temps_eight_twelve = numpy.zeros(len(eight_twelve_lat))
Boxplot_mine_temps_eight_twelve[:] = numpy.nan

Boxplot_pond_temps_eight_twelve = numpy.zeros(len(eight_twelve_lat))
Boxplot_pond_temps_eight_twelve[:] = numpy.nan

# For 12:00 to 16:00
Boxplot_mine_temps_twelve_sixteen = numpy.zeros(len(twelve_sixteen_lat))
Boxplot_mine_temps_twelve_sixteen[:] = numpy.nan

Boxplot_pond_temps_twelve_sixteen = numpy.zeros(len(twelve_sixteen_lat))
Boxplot_pond_temps_twelve_sixteen[:] = numpy.nan

# For 16:00 to 20:00
Boxplot_mine_temps_sixteen_twenty = numpy.zeros(len(sixteen_twenty_lat))
Boxplot_mine_temps_sixteen_twenty[:] = numpy.nan

Boxplot_pond_temps_sixteen_twenty = numpy.zeros(len(sixteen_twenty_lat))
Boxplot_pond_temps_sixteen_twenty[:] = numpy.nan

# For 20:00 to 24:00
Boxplot_mine_temps_twenty_twentyfour = numpy.zeros(len(twenty_twentyfour_lat))
Boxplot_mine_temps_twenty_twentyfour[:] = numpy.nan

Boxplot_pond_temps_twenty_twentyfour = numpy.zeros(len(twenty_twentyfour_lat))
Boxplot_pond_temps_twenty_twentyfour[:] = numpy.nan

#
    ##########################################################################################################################

# Mine and Pond GPS boundaries (in decimal degree format), Based on May 2018 Landsat 8 OLI
    Image
# Mine boundaries
min_mine_lat = XX.XXXXXX
max_mine_lat = XX.XXXXXX
min_mine_lon = -XXX.XXXXXX
max_mine_lon = -XXX.XXXXXX

# Pond Boundaries
min_pond_lat = XX.XXXXXX
max_pond_lat = XX.XXXXXX
min_pond_lon = -XXX.XXXXXX
```

```
max_pond_lon = -XXX.XXXXXX

#
    ##########################################################################################################################################

# Compute binned data
# For hour averaging
# From 00:00 to 04:00
for i in range(0, len(zero_four_lat)):
    print(zero_four_lat[i])
    print(zero_four_lon[i])
    print('The Max lat is: '+str(Latmax)+'\n')
    print('The Min lat is: '+str(Latmin)+'\n')
    print('The Max lon is: '+str(Lonmax)+'\n')
    print('The Min lon is: '+str(Lonmin)+'\n')
    if numpy.isnan(zero_four_lat[i]) == False or numpy.isnan(zero_four_lon[i]) == False:
        LatIndex_zero_four = int((zero_four_lat[i] - Latmin) * nLat / (Latmax - Latmin))
        LonIndex_zero_four = int((zero_four_lon[i] - Lonmin) * nLon / (Lonmax - Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if zero_four_lat[i] > Latmax or zero_four_lat[i] < Latmin or zero_four_lon[i] <
            Lonmax or\
                zero_four_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of site boundary
        elif (zero_four_lat[i] >= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
            zero_four_lon[i] >= -XXX.XXXX)\
                or (zero_four_lat[i] >= XX.XXXX and zero_four_lon[i] <=-XXX.XXXX and
                    zero_four_lon[i] >= -XXX.XXXX):
            continue
        elif (zero_four_lat[i] >= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
            zero_four_lon[i] >= -XXX.XXXX)\
                or (zero_four_lat[i] <= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
                    zero_four_lon[i] >= -XXX.XXXX):
            continue
        elif (zero_four_lat[i] <= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
            zero_four_lon[i] >= -XXX.XXXX)\
                or (zero_four_lat[i] >= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
                    zero_four_lon[i] >= -XXX.XXXX):
            continue
        elif (zero_four_lat[i] >= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
            zero_four_lon[i] >= -XXX.XXXX)\
                or (zero_four_lat[i] <= XX.XXXX and zero_four_lon[i] <= -XXX.XXXX and
                    zero_four_lon[i] >= -XXX.XXXX):
            continue

        else:
            TMatrix_zero_four[LatIndex_zero_four][LonIndex_zero_four][i] = zero_four_tempK[i
                ]
```

```
        # Check if image is within mine or pond boundaries
        # Check mine latitude and longitude boundaries
        if (zero_four_lat[i] >= min_mine_lat and zero_four_lat[i] <= max_mine_lat) and\
                (zero_four_lon[i] >= min_mine_lon and zero_four_lon[i] <= max_mine_lon):
            Boxplot_mine_temps_zero_four[i] = zero_four_tempK[i]

        # Check pond latitude and longitude boundaries
        elif (zero_four_lat[i] >= min_pond_lat and zero_four_lat[i] <= max_pond_lat) and\
                (zero_four_lon[i] >= min_pond_lon and zero_four_lon[i] <= max_pond_lon):
            Boxplot_pond_temps_zero_four[i] = zero_four_tempK[i]
        else:
            print('Image not within mine or pond bounds')


# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(zero_four_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_zero_four[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(zero_four_lat)) & (TMatrix_zero_four[i][j][k] == numpy.nan):
            TMatrix_zero_four_median[i][j] = numpy.nan
        else:
            TMatrix_zero_four_median[i][j] = numpy.nanpercentile(TMatrix_zero_four[i][j
                ][:],50)

#
    ##########################################################################################################################################

# From 04:00 to 08:00
for i in range(0, len(four_eight_lat)):
    if numpy.isnan(four_eight_lat[i]) == False or numpy.isnan(four_eight_lon[i]) == False:
        LatIndex_four_eight = int((four_eight_lat[i] - Latmin) * nLat / (Latmax - Latmin))
        LonIndex_four_eight = int((four_eight_lon[i] - Lonmin) * nLon / (Lonmax - Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if four_eight_lat[i] > Latmax or four_eight_lat[i] < Latmin or four_eight_lon[i] <
            Lonmax or\
                four_eight_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of the site boundary
        elif (four_eight_lat[i] >= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
            four_eight_lon[i] >= -XXX.XXXX)\
```

```
                or (four_eight_lat[i] >= XX.XXXX and four_eight_lon[i] <=-XXX.XXXX and
                    four_eight_lon[i] >= -XXX.XXXX):
            continue
        elif (four_eight_lat[i] >= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
            four_eight_lon[i] >= -XXX.XXXX)\
                or (four_eight_lat[i] <= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
                    four_eight_lon[i] >= -XXX.XXXX):
            continue
        elif (four_eight_lat[i] <= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
            four_eight_lon[i] >= -XXX.XXXX)\
                or (four_eight_lat[i] >= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
                    four_eight_lon[i] >= -XXX.XXXX):
            continue
        elif (four_eight_lat[i] >= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
            four_eight_lon[i] >= -XXX.XXXX)\
                or (four_eight_lat[i] <= XX.XXXX and four_eight_lon[i] <= -XXX.XXXX and
                    four_eight_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_four_eight[LatIndex_four_eight][LonIndex_four_eight][i] =
                four_eight_tempK[i]

        # Check if image is within mine or pond boundaries
        # Check mine latitude and longitude boundaries
        if (four_eight_lat[i] >= min_mine_lat and four_eight_lat[i] <= max_mine_lat) and\
                (four_eight_lon[i] >= min_mine_lon and four_eight_lon[i] <= max_mine_lon):
            Boxplot_mine_temps_four_eight[i] = four_eight_tempK[i]

        # Check pond latitude and longitude boundaries
        elif (four_eight_lat[i] >= min_pond_lat and four_eight_lat[i] <= max_pond_lat) and\
                (four_eight_lon[i] >= min_pond_lon and four_eight_lon[i] <= max_pond_lon):
            Boxplot_pond_temps_four_eight[i] = four_eight_tempK[i]
        else:
            print('Image not within mine or pond bounds')

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(four_eight_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_four_eight[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(four_eight_lat)) & (TMatrix_four_eight[i][j][k] == numpy.nan):
            TMatrix_four_eight_median[i][j] = numpy.nan
        else:
            TMatrix_four_eight_median[i][j] = numpy.nanpercentile(TMatrix_four_eight[i][j
                ][:],50)
```

```
#
    ##########################################################################################################

# From 08:00 to 12:00
for i in range(0, len(eight_twelve_lat)):
    if numpy.isnan(eight_twelve_lat[i]) == False or numpy.isnan(eight_twelve_lon[i]) ==
        False:
        LatIndex_eight_twelve = int((eight_twelve_lat[i] - Latmin) * nLat / (Latmax - Latmin
            ))
        LonIndex_eight_twelve = int((eight_twelve_lon[i] - Lonmin) * nLon / (Lonmax - Lonmin
            ))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if eight_twelve_lat[i] > Latmax or eight_twelve_lat[i] < Latmin or eight_twelve_lon[
            i] < Lonmax\
                or eight_twelve_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of the site boundary
        elif (eight_twelve_lat[i] >= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
            eight_twelve_lon[i] >= -XXX.XXXX)\
                or (eight_twelve_lat[i] >= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
                    eight_twelve_lon[i] >= -XXX.XXXX):
            continue
        elif (eight_twelve_lat[i] >= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
            eight_twelve_lon[i] >= -XXX.XXXX)\
                or (eight_twelve_lat[i] <= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
                    eight_twelve_lon[i] >= -XXX.XXXX):
            continue
        elif (eight_twelve_lat[i] <= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
            eight_twelve_lon[i] >= -XXX.XXXX)\
                or (eight_twelve_lat[i] >= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
                    eight_twelve_lon[i] >= -XXX.XXXX):
            continue
        elif (eight_twelve_lat[i] >= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
            eight_twelve_lon[i] >= -XXX.XXXX)\
                or (eight_twelve_lat[i] <= XX.XXXX and eight_twelve_lon[i] <= -XXX.XXXX and
                    eight_twelve_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_eight_twelve[LatIndex_eight_twelve][LonIndex_eight_twelve][i] =
                eight_twelve_tempK[i]

        # Check if image is within mine or pond boundaries
        # Check mine latitude and longitude boundaries
        if (eight_twelve_lat[i] >= min_mine_lat and eight_twelve_lat[i] <= max_mine_lat)
            and\
                (eight_twelve_lon[i] >= min_mine_lon and eight_twelve_lon[i] <= max_mine_lon
                    ):
            Boxplot_mine_temps_eight_twelve[i] = eight_twelve_tempK[i]
```

```
        # Check pond latitude and longitude boundaries
        elif (eight_twelve_lat[i] >= min_pond_lat and eight_twelve_lat[i] <= max_pond_lat)
            and\
                (eight_twelve_lon[i] >= min_pond_lon and eight_twelve_lon[i] <= max_pond_lon
                    ):
            Boxplot_pond_temps_eight_twelve[i] = eight_twelve_tempK[i]
        else:
            print('Image not within mine or pond bounds')

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(eight_twelve_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_eight_twelve[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(eight_twelve_lat)) & (TMatrix_eight_twelve[i][j][k] == numpy.nan):
            TMatrix_eight_twelve_median[i][j] = numpy.nan
        else:
            TMatrix_eight_twelve_median[i][j] = numpy.nanpercentile(TMatrix_eight_twelve[i][
                j][:],50)

#
    ##########################################################################################################################

# From 12:00 to 16:00
for i in range(0, len(twelve_sixteen_lat)):
    if numpy.isnan(twelve_sixteen_lat[i]) == False or numpy.isnan(twelve_sixteen_lon[i]) ==
        False:
        LatIndex_twelve_sixteen = int((twelve_sixteen_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_twelve_sixteen = int((twelve_sixteen_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if twelve_sixteen_lat[i] > Latmax or twelve_sixteen_lat[i] < Latmin or
            twelve_sixteen_lon[i] < Lonmax\
                or twelve_sixteen_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of site boundary
        elif (twelve_sixteen_lat[i] >= XX.XXXX and twelve_sixteen_lon[i] <= -XXX.XXXX and
              twelve_sixteen_lon[i] >= -XXX.XXXX) or (twelve_sixteen_lat[i] >= XX.XXXX and
                                                      twelve_sixteen_lon[i] <= -XXX.XXXX and
                                                      twelve_sixteen_lon[i] >= -XXX.XXXX):
```

```
            continue
        elif (twelve_sixteen_lat[i] >= XX.XXXX and twelve_sixteen_lon[i] <= -XXX.XXXX and
              twelve_sixteen_lon[i] >= -XXX.XXXX) or (twelve_sixteen_lat[i] <= XX.XXXX and
                                                      twelve_sixteen_lon[i] <= -XXX.XXXX and
                                                      twelve_sixteen_lon[i] >= -XXX.XXXX):
            continue
        elif (twelve_sixteen_lat[i] <= XX.XXXX and twelve_sixteen_lon[i] <= -XXX.XXXX and
              twelve_sixteen_lon[i] >= -XXX.XXXX) or (twelve_sixteen_lat[i] >= XX.XXXX and
                                                      twelve_sixteen_lon[i] <= -XXX.XXXX and
                                                      twelve_sixteen_lon[i] >= -XXX.XXXX):
            continue
        elif (twelve_sixteen_lat[i] >= XX.XXXX and twelve_sixteen_lon[i] <= -XXX.XXXX and
              twelve_sixteen_lon[i] >= -XXX.XXXX) or (twelve_sixteen_lat[i] <= XX.XXXX and
                                                      twelve_sixteen_lon[i] <= -XXX.XXXX and
                                                      twelve_sixteen_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_twelve_sixteen[LatIndex_twelve_sixteen][LonIndex_twelve_sixteen][i] =
                twelve_sixteen_tempK[i]

        # Check if image is within mine or pond boundaries
        # Check mine latitude and longitude boundaries
        if (twelve_sixteen_lat[i] >= min_mine_lat and twelve_sixteen_lat[i] <= max_mine_lat)
            and\
                (twelve_sixteen_lon[i] >= min_mine_lon and twelve_sixteen_lon[i] <=
                    max_mine_lon):
            Boxplot_mine_temps_twelve_sixteen[i] = twelve_sixteen_tempK[i]

        # Check pond latitude and longitude boundaries
        elif (twelve_sixteen_lat[i] >= min_pond_lat and twelve_sixteen_lat[i] <=
            max_pond_lat) and\
                (twelve_sixteen_lon[i] >= min_pond_lon and twelve_sixteen_lon[i] <=
                    max_pond_lon):
            Boxplot_pond_temps_twelve_sixteen[i] = twelve_sixteen_tempK[i]
        else:
            print('Image not within mine or pond bounds')

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(twelve_sixteen_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_twelve_sixteen[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(twelve_sixteen_lat)) & (TMatrix_twelve_sixteen[i][j][k] == numpy.nan):
            TMatrix_twelve_sixteen_median[i][j] = numpy.nan
        else:
            TMatrix_twelve_sixteen_median[i][j] = numpy.nanpercentile(TMatrix_twelve_sixteen
```

```
                [i][j][:],50)

#
    ##########################################################################################################################

# From 16:00 to 20:00
for i in range(0, len(sixteen_twenty_lat)):
    if numpy.isnan(sixteen_twenty_lat[i]) == False or numpy.isnan(sixteen_twenty_lon[i]) ==
        False:
        LatIndex_sixteen_twenty = int((sixteen_twenty_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_sixteen_twenty = int((sixteen_twenty_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if sixteen_twenty_lat[i] > Latmax or sixteen_twenty_lat[i] < Latmin or
            sixteen_twenty_lon[i] < Lonmax\
                or sixteen_twenty_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside the site boundary
        elif (sixteen_twenty_lat[i] >= XX.XXXX and sixteen_twenty_lon[i] <= -XXX.XXXX and
              sixteen_twenty_lon[i] >= -XXX.XXXX) or (sixteen_twenty_lat[i] >= XX.XXXX and
                                                      sixteen_twenty_lon[i] <= -XXX.XXXX and
                                                      sixteen_twenty_lon[i] >= -XXX.XXXX):
            continue
        elif (sixteen_twenty_lat[i] >= XX.XXXX and sixteen_twenty_lon[i] <= -XXX.XXXX and
              sixteen_twenty_lon[i] >= -XXX.XXXX) or (sixteen_twenty_lat[i] <= XX.XXXX and
                                                      sixteen_twenty_lon[i] <= -XXX.XXXX and
                                                      sixteen_twenty_lon[i] >= -XXX.XXXX):
            continue
        elif (sixteen_twenty_lat[i] <= XX.XXXX and sixteen_twenty_lon[i] <= -XXX.XXXX and
              sixteen_twenty_lon[i] >= -XXX.XXXX) or (sixteen_twenty_lat[i] >= XX.XXXX and
                                                      sixteen_twenty_lon[i] <= -XXX.XXXX and
                                                      sixteen_twenty_lon[i] >= -XXX.XXXX):
            continue
        elif (sixteen_twenty_lat[i] >= XX.XXXX and sixteen_twenty_lon[i] <= -XXX.XXXX and
              sixteen_twenty_lon[i] >= -XXX.XXXX) or (sixteen_twenty_lat[i] <= XX.XXXX and
                                                      sixteen_twenty_lon[i] <= -XXX.XXXX and
                                                      sixteen_twenty_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_sixteen_twenty[LatIndex_sixteen_twenty][LonIndex_sixteen_twenty][i] =
                sixteen_twenty_tempK[i]

        # Check if image is within mine or pond boundaries
        # Check mine latitude and longitude boundaries
        if (sixteen_twenty_lat[i] >= min_mine_lat and sixteen_twenty_lat[i] <= max_mine_lat)
            and\
                (sixteen_twenty_lon[i] >= min_mine_lon and sixteen_twenty_lon[i] <=
```

```
                max_mine_lon):
            Boxplot_mine_temps_sixteen_twenty[i] = sixteen_twenty_tempK[i]

        # Check pond latitude and longitude boundaries
        elif (sixteen_twenty_lat[i] >= min_pond_lat and sixteen_twenty_lat[i] <=
            max_pond_lat) and\
                (sixteen_twenty_lon[i] >= min_pond_lon and sixteen_twenty_lon[i] <=
                    max_pond_lon):
            Boxplot_pond_temps_sixteen_twenty[i] = sixteen_twenty_tempK[i]
        else:
            print('Image not within mine or pond bounds')


# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(sixteen_twenty_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_sixteen_twenty[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(sixteen_twenty_lat)) & (TMatrix_sixteen_twenty[i][j][k] == numpy.nan):
            TMatrix_sixteen_twenty_median[i][j] = numpy.nan
        else:
            TMatrix_sixteen_twenty_median[i][j] = numpy.nanpercentile(TMatrix_sixteen_twenty
                [i][j][:],50)

#
    ##########################################################################################################################

# From 20:00 to 24:00
for i in range(0, len(twenty_twentyfour_lat)):
    if numpy.isnan(twenty_twentyfour_lat[i]) == False or numpy.isnan(twenty_twentyfour_lon[i
        ]) == False:
        LatIndex_twenty_twentyfour = int((twenty_twentyfour_lat[i] - Latmin) * nLat / (
            Latmax - Latmin))
        LonIndex_twenty_twentyfour = int((twenty_twentyfour_lon[i] - Lonmin) * nLon / (
            Lonmax - Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if twenty_twentyfour_lat[i] > Latmax or twenty_twentyfour_lat[i] < Latmin or
            twenty_twentyfour_lon[i] < Lonmax\
                or twenty_twentyfour_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of site boundary
        elif (twenty_twentyfour_lat[i] >= XX.XXXX and twenty_twentyfour_lon[i] <= -XXX.XXXX
```

```
    and
      twenty_twentyfour_lon[i] >= -XXX.XXXX) or (twenty_twentyfour_lat[i] >= XX.XXXX
          and
                                                 twenty_twentyfour_lon[i] <= -
                                                     XXX.XXXX and
                                                 twenty_twentyfour_lon[i] >= -
                                                     XXX.XXXX):
    continue
elif (twenty_twentyfour_lat[i] >= XX.XXXX and twenty_twentyfour_lon[i] <= -XXX.XXXX
    and
      twenty_twentyfour_lon[i] >= -XXX.XXXX) or (twenty_twentyfour_lat[i] <= XX.XXXX
          and
                                                 twenty_twentyfour_lon[i] <= -XXX.
                                                     XXXX and
                                                 twenty_twentyfour_lon[i] >= -XXX.
                                                     XXXX):
    continue
elif (twenty_twentyfour_lat[i] <= XX.XXXX and twenty_twentyfour_lon[i] <= -XXX.XXXX
    and
      twenty_twentyfour_lon[i] >= -XXX.XXXX) or (twenty_twentyfour_lat[i] >= XX.XXXX
          and
                                                 twenty_twentyfour_lon[i] <= -XXX.
                                                     XXXX and
                                                 twenty_twentyfour_lon[i] >= -XXX.
                                                     XXXX):
    continue
elif (twenty_twentyfour_lat[i] >= XX.XXXX and twenty_twentyfour_lon[i] <= -XXX.XXXX
    and
      twenty_twentyfour_lon[i] >= -XXX.XXXX) or (twenty_twentyfour_lat[i] <= XX.XXXX
          and
                                                 twenty_twentyfour_lon[i] <= -XXX.
                                                     XXXX and
                                                 twenty_twentyfour_lon[i] >= -XXX.
                                                     XXXX):
    continue
else:
    TMatrix_twenty_twentyfour[LatIndex_twenty_twentyfour][LonIndex_twenty_twentyfour
        ][i]\
        = twenty_twentyfour_tempK[i]

# Check if image is within mine or pond boundaries
# Check mine latitude and longitude boundaries
if (twenty_twentyfour_lat[i] >= min_mine_lat and twenty_twentyfour_lat[i] <=
    max_mine_lat) and\
        (twenty_twentyfour_lon[i] >= min_mine_lon and twenty_twentyfour_lon[i] <=
            max_mine_lon):
    Boxplot_mine_temps_twenty_twentyfour[i] = twenty_twentyfour_tempK[i]

# Check pond latitude and longitude boundaries
elif (twenty_twentyfour_lat[i] >= min_pond_lat and twenty_twentyfour_lat[i] <=
    max_pond_lat) and\
        (twenty_twentyfour_lon[i] >= min_pond_lon and twenty_twentyfour_lon[i] <=
```

```
                    max_pond_lon):
                Boxplot_pond_temps_twenty_twentyfour[i] = twenty_twentyfour_tempK[i]
            else:
                print('Image not within mine or pond bounds')

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(twenty_twentyfour_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_twenty_twentyfour[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(twenty_twentyfour_lat)) & (TMatrix_twenty_twentyfour[i][j][k] == numpy.
            nan):
            TMatrix_twenty_twentyfour_median[i][j] = numpy.nan
        else:
            TMatrix_twenty_twentyfour_median[i][j] = numpy.nanpercentile(
                TMatrix_twenty_twentyfour[i][j][:],50)

#
    ##########################################################################################################################################

# For May 24 FLIR temperatures
for i in range(0, len(May_24_SA_FLIR_lat)):
    if numpy.isnan(May_24_SA_FLIR_lat[i]) == False or numpy.isnan(May_24_SA_FLIR_lon[i]) ==
        False:
        LatIndex_May_24_SA_FLIR = int((May_24_SA_FLIR_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_May_24_SA_FLIR = int((May_24_SA_FLIR_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if May_24_SA_FLIR_lat[i] > Latmax or May_24_SA_FLIR_lat[i] < Latmin or
            May_24_SA_FLIR_lon[i] < Lonmax or\
                May_24_SA_FLIR_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of site boundary
        elif (May_24_SA_FLIR_lat[i] >= XX.XXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
              May_24_SA_FLIR_lon[i] >= -XXX.XXXX) or (May_24_SA_FLIR_lat[i] >= XX.XXXX and
                                                      May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
                                                      May_24_SA_FLIR_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_FLIR_lat[i] >= XX.XXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
              May_24_SA_FLIR_lon[i] >= -XXX.XXXX) or (May_24_SA_FLIR_lat[i] <= XX.XXXX and
                                                      May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
```

```
                                                    May_24_SA_FLIR_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_FLIR_lat[i] <= XX.XXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
              May_24_SA_FLIR_lon[i] >= -XXX.XXXX) or (May_24_SA_FLIR_lat[i] >= XX.XXXX and
                                                    May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
                                                    May_24_SA_FLIR_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_FLIR_lat[i] >= XX.XXXX and May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
              May_24_SA_FLIR_lon[i] >= -XXX.XXXX) or (May_24_SA_FLIR_lat[i] <= XX.XXXX and
                                                    May_24_SA_FLIR_lon[i] <= -XXX.XXXX and
                                                    May_24_SA_FLIR_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_May_24_SA_FLIR[LatIndex_May_24_SA_FLIR][LonIndex_May_24_SA_FLIR][i] =
                May_24_SA_FLIR_tempK[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(May_24_SA_FLIR_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_May_24_SA_FLIR[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(May_24_SA_FLIR_lat)) & (TMatrix_May_24_SA_FLIR[i][j][k] == numpy.nan):
            TMatrix_May_24_SA_FLIR_Median[i][j] = numpy.nan
        else:
            TMatrix_May_24_SA_FLIR_Median[i][j] = numpy.nanpercentile(TMatrix_May_24_SA_FLIR
                [i][j][:],50)

#
    ##########################################################################################################################

# For May 24 MODIS temperatures
for i in range(0, len(May_24_SA_MODIS_lat)):
    if numpy.isnan(May_24_SA_MODIS_lat[i]) == False or numpy.isnan(May_24_SA_MODIS_lon[i])
        == False:
        LatIndex_May_24_SA_MODIS = int((May_24_SA_MODIS_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_May_24_SA_MODIS = int((May_24_SA_MODIS_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if May_24_SA_MODIS_lat[i] > Latmax or May_24_SA_MODIS_lat[i] < Latmin or
            May_24_SA_MODIS_lon[i] < Lonmax\
                or May_24_SA_MODIS_lon[i] > Lonmin:
            continue
```

```
        # Manually crop temperatures outside of site boundary
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] >= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] <= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] <= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] >= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] <= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        else:
            TMatrix_May_24_SA_MODIS[LatIndex_May_24_SA_MODIS][LonIndex_May_24_SA_MODIS][i] =
                May_24_SA_MODIS_tempK[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(May_24_SA_MODIS_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_May_24_SA_MODIS[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(May_24_SA_MODIS_lat)) & (TMatrix_May_24_SA_MODIS[i][j][k] == numpy.nan)
            :
            TMatrix_May_24_SA_MODIS_Median[i][j] = numpy.nan
        else:
            TMatrix_May_24_SA_MODIS_Median[i][j] = numpy.nanpercentile(
                TMatrix_May_24_SA_MODIS[i][j][:],50)

#
    ##########################################################################################################################

# For May 24 Percentage Error/Absolute Error
```

```
# Note: latitudes and longitudes are the same as MODIS above
for i in range(0, len(May_24_SA_MODIS_lat)):
    if numpy.isnan(May_24_SA_MODIS_lat[i]) == False or numpy.isnan(May_24_SA_MODIS_lon[i])
        == False:
        LatIndex_May_24_SA_PE = int((May_24_SA_MODIS_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_May_24_SA_PE = int((May_24_SA_MODIS_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if May_24_SA_MODIS_lat[i] > Latmax or May_24_SA_MODIS_lat[i] < Latmin or
            May_24_SA_MODIS_lon[i] < Lonmax\
                or May_24_SA_MODIS_lon[i] > Lonmin:
            continue

        # Manually crop temperatures outside of site boundary
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] >= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] <= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] <= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] >= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        elif (May_24_SA_MODIS_lat[i] >= XX.XXXX and May_24_SA_MODIS_lon[i] <= -XXX.XXXX and
              May_24_SA_MODIS_lon[i] >= -XXX.XXXX) or (May_24_SA_MODIS_lat[i] <= XX.XXXX and
                                                       May_24_SA_MODIS_lon[i] <= -XXX.XXXX
                                                           and
                                                       May_24_SA_MODIS_lon[i] >= -XXX.XXXX):
            continue
        else:
            PE_Matrix_May_24_SA[LatIndex_May_24_SA_PE][LonIndex_May_24_SA_PE][i] =
                May_24_SA_PE[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(May_24_SA_MODIS_lat)):
            # If a real number is encountered, a median can be calculated
```

```
            if PE_Matrix_May_24_SA[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be equal to Nan, otherwise
            calculate the median
        #  temperature excluding Nan values
        if (k == len(May_24_SA_MODIS_lat)) & (PE_Matrix_May_24_SA[i][j][k] == numpy.nan):
            PE_Matrix_May_24_SA_Median[i][j] = numpy.nan
        else:
            PE_Matrix_May_24_SA_Median[i][j] = numpy.nanpercentile(PE_Matrix_May_24_SA[i][j
                ][:],50)


#
    ##########################################################################################################################

# Color plotting
# State maximum and minimum colour bar ranges for each respective time interval
# 0-4
color_bar_min_zero_four = 260
color_bar_max_zero_four = 295

# 4-8
color_bar_min_four_eight = 265
color_bar_max_four_eight = 290

# 8-12
color_bar_min_eight_twelve = 275
color_bar_max_eight_twelve = 300

# 12-16
color_bar_min_twelve_sixteen = 280
color_bar_max_twelve_sixteen = 320

# 16-20
color_bar_min_sixteen_twenty = 280
color_bar_max_sixteen_twenty = 310

# 20-24
color_bar_min_twenty_twentyfour = 275
color_bar_max_twenty_twentyfour = 300

# May 24 surface temperature colour bar range
color_bar_min_May_24_SA = 295
color_bar_max_May_24_SA = 325

# Figure size
figuresize = (10,6)

# Font size
font_size = 16
title_font_size = 16
```

```
# X/Y distance and colour bar tick sizes
tick_size = 11
cbar_tick_size = 16

# Figure parameters
# For surface temperature maps
axes_label_fontsize = 36
axes_ticks_fontsize = 34

# For boxplots
axes_bxplt_label_fontsize = 42
axes_bxplt_ticks_fontsize = 40

# For colour bars
axes_clrbar_label_fontsize = 32
axes_clrbar_ticks_fontsize = 30

# Position of x and y labels away from respective axes in points
x_labelpad = -5
y_labelpad = -5

#
    ##########################################################################################################################

# For May 24 Plots
May_24_axes_label_fontsize = 18
May_24_axes_ticks_fontsize = 17
May_24_axes_clrbar_ticks_fontsize = 17

# FLIR ST Map Parameters
May_24_cbar_FLIR_min = 295
May_24_cbar_FLIR_max = 325
May_24_cbar_FLIR_label = '$T$ [K]'
May_24_cbar_FLIR_fontsize = 18

# MODIS ST Map Parameters
May_24_cbar_MODIS_min = 295
May_24_cbar_MODIS_max = 325
May_24_cbar_MODIS_label = '$T$ [K]'
May_24_cbar_MODIS_fontsize = 18

# Percentage Error Map Parameters
# May_24_cbar_PE_min = 0
# May_24_cbar_PE_max = 5
# May_24_cbar_PE_label = 'Relative error in percentage'
# May_24_cbar_PE_fontsize = 12

# Absolute Error in Kelvin
May_24_cbar_PE_min = -15
May_24_cbar_PE_max = 15
May_24_cbar_PE_label = 'Absolute Error [K]'
May_24_cbar_PE_fontsize = 18
```

```
#
    ##########################################################################################################################

# Filename resolution
if nLat == 20:
    filename_res = '1km'
elif nLat == 40:
    filename_res = '500m'
elif nLat == 100:
    filename_res = '200m'
elif nLat == 10 and nLon == 10:
    filename_res = '2000m_x_2500m'
else:
    print('There are problems with the resolution size in the outputted filename')

#
    ##########################################################################################################################

# Import mining facility boundary coordinates,
property_bounds_filename = '/export/home/users/username/Documents/DG_Temp/
    Mining_Facility_2018/QGIS' \
                           '/FacilityPerimeter_May_2018.txt'

property_bounds_data = numpy.genfromtxt(property_bounds_filename, delimiter=',')

property_bounds_lon = property_bounds_data[:,0]
property_bounds_lat = property_bounds_data[:,1]

property_bounds_lon_update = numpy.zeros((len(property_bounds_lon),1))
property_bounds_lat_update = numpy.zeros((len(property_bounds_lon),1))

# Import pond property boundaries, used Landsat 8 OLI May 17, 2018 image
pond_boundary_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018
    /QGIS' \
                           '/PondPerimeter_May_2018.txt'

pond_bounds_data = numpy.genfromtxt(pond_boundary_filename, delimiter=',')

pond_bounds_lon = pond_bounds_data[:,0]
pond_bounds_lat = pond_bounds_data[:,1]

# Import mine boundary coordinates, used Landsat 8 OLI May 17, 2018 image
mine_bounds_filename = '/export/home/users/username/Documents/DG_Temp/Mining_Facility_2018/
    QGIS' \
                           '/MinePerimeter_May_2018.txt'

mine_bounds_data = numpy.genfromtxt(mine_bounds_filename, delimiter=',')

mine_bounds_lon = mine_bounds_data[:,0]
mine_bounds_lat = mine_bounds_data[:,1]
```

```
# Declare TANAB2 launch locations (MFT, Berm, Mine)
base_lon = [-XXX.XXXX, -XXX.XXXX, -XXX.XXXX]
base_lat = [XX.XXXX, XX.XXXX, XX.XXXX]

# This is equal to 5km in decimal degrees (longitude only)
five_km_decimal_deg = 0.083314

# This is equal to 2km in decimal degrees (latitude only)
two_km_decimal_deg = 0.017384

# Longitude locations for x ticks
xticks_array = [Lonmax, Lonmax+(five_km_decimal_deg), Lonmax+(five_km_decimal_deg*2),
                Lonmax+(five_km_decimal_deg*3), Lonmax+(five_km_decimal_deg*4), Lonmin]

# Latitude locations for y ticks
yticks_array = [Latmin, Latmin+two_km_decimal_deg, Latmin+(two_km_decimal_deg*2), Latmin+(
    two_km_decimal_deg*3),
                Latmin+(two_km_decimal_deg*4), Latmin+(two_km_decimal_deg*5), Latmin+(
                    two_km_decimal_deg*6), Latmax]

# Longitude labels for x ticks
xticks_label = ['0', '5', '10', '15', '20', '22.5']
xaxis_label = '$X$ [km]'

# Latitude labels for y ticks
yticks_label = ['0', '2', '4', '6', '8', '10', '12', '13.7']
yaxis_label = '$Y$ [km]'

# colour bar label
color_bar_label = '$T$ [K]'

# TANAB2_dot size
launch_size = 25

# Use latex font for labels
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

# Directory to save images
direct_save ='/export/home/users/username/Documents/DG_Temp/' \
             'Mining_Facility_2018/Processed_Data/Figures/'

#
    ##########################################################################################################################

# Create surface temperature maps and boxplots for each time interval and for FLIR, MODIS,
    and Percentage
#  Error/Absolute Error data
# At 00:00 to 04:00
Lataxis_zero_four = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_zero_four = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_zero_four,LatAxis_zero_four = numpy.meshgrid(Lonaxis_zero_four,Lataxis_zero_four)
```

```
fig_zero_four, ax = plt.subplots(figsize=figuresize)
Tpcolor_zero_four=plt.pcolor(LonAxis_zero_four,LatAxis_zero_four,TMatrix_zero_four_median,
                             vmin=color_bar_min_zero_four, vmax=color_bar_max_zero_four)
cbar_zero_four = plt.colorbar(Tpcolor_zero_four)
cbar_zero_four.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_zero_four.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array,xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_zero_four.show()

fig_zero_four.savefig(direct_save+'0000_0400_map.png')
plt.show()

# Filter Nan for boxplot
Boxplot_mine_temps_zero_four_filtered = Boxplot_mine_temps_zero_four[~numpy.isnan(
    Boxplot_mine_temps_zero_four)]
Boxplot_pond_temps_zero_four_filtered = Boxplot_pond_temps_zero_four[~numpy.isnan(
    Boxplot_pond_temps_zero_four)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_zero_four_filtered != [] or Boxplot_pond_temps_zero_four_filtered !=
    []:
    # Plot Boxplot with filtered Nan data
    fig_bp_zero_four, ax = plt.subplots(figsize=figuresize)
    bp = plt.boxplot([Boxplot_pond_temps_zero_four_filtered,
        Boxplot_mine_temps_zero_four_filtered],
                     labels=['Tailings Pond', 'Mine'])
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.xticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.tight_layout()
    fig_bp_zero_four.savefig(direct_save+'0000_0400_boxplot.png')
    plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_zero_four = numpy.zeros((nLat, 1))
LonAxis_median_zero_four = numpy.zeros((nLon, 1))

LatAxisIndex_zero_four = numpy.empty((nLat+1,1))
LatAxisIndex_zero_four[:] = numpy.nan
```

```
LonAxisIndex_zero_four = numpy.empty((nLon+1,1))
LonAxisIndex_zero_four[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_zero_four[a] = ((Lataxis_zero_four[a])+(Lataxis_zero_four[(a+1)]))/2

for j in range(0, nLon):
    LonAxis_median_zero_four[j] = (Lonaxis_zero_four[j]+Lonaxis_zero_four[j+1])/2

# Save latitude/longitude indices and median temperatures
output_zero_four_filename = direct_save+'Figure_Data/Zero_Four_MedianTemp_'+filename_res+'.
    txt'
outputFile_zero_four = open(output_zero_four_filename, 'w')
outputFile_zero_four.write("# Latitude, Longitude indices, median temperature and latitude/
    longitude"
                           " bounds for mining facility, pond and mine \n")
outputFile_zero_four.write("#By: Ryan Byerlay \n")
outputFile_zero_four.write("#Recorded Time is Local Time (MDT) \n")
outputFile_zero_four.write('The nLat is: '+str(nLat)+', The nLon is: '+str(nLon)+'\n')
outputFile_zero_four.write('The Latmax is: '+str(Latmax)+', The Latmin is: '+str(Latmin)+',
    The Lonmax is: '
                           +str(Lonmax)+', The Lonmin is: '+str(Lonmin)+'\n')
outputFile_zero_four.write('The max mine lat is: '+str(max_mine_lat)+', The min mine lat is:
    '+str(min_mine_lat)+
                           ', The max mine lon is: '+str(max_mine_lon)+', The min mine lon
                               is: '
                           +str(min_mine_lon)+'\n')
outputFile_zero_four.write('The max pond lat is: '+str(max_pond_lat)+', The min pond lat is:
    '+str(min_pond_lat)+
                           ', The max pond lon is: '+str(max_pond_lon)+', The min pond lon
                               is: '
                           +str(min_pond_lon)+'\n')
outputFile_zero_four.write("#0:LatAxis_zero_four \t #1:LonAxis_zero_four \t "
                           "#2:MedianTemp_zero_four(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_zero_four)-1):
    for j in range(0, len(LonAxis_zero_four)-1):
        print(TMatrix_zero_four_median[i][j])
        if numpy.isnan(TMatrix_zero_four_median[i][j]) == False:
            outputFile_zero_four.write("%f \t %f \t %f \n" % (LatAxis_median_zero_four[i],
                                                             LonAxis_median_zero_four[j],
                                                             TMatrix_zero_four_median[i][j
                                                                 ]))
outputFile_zero_four.close()

#
    ##########################################################################################################################

# At 04:00 to 08:00
Lataxis_four_eight = numpy.linspace(Latmin,Latmax,nLat+1)
```

```
Lonaxis_four_eight = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_four_eight,LatAxis_four_eight = numpy.meshgrid(Lonaxis_four_eight,Lataxis_four_eight
    )

fig_four_eight, ax = plt.subplots(figsize=figuresize)
Tpcolor_four_eight=plt.pcolor(LonAxis_four_eight,LatAxis_four_eight,
    TMatrix_four_eight_median,
                              vmin=color_bar_min_four_eight, vmax=color_bar_max_four_eight)
cbar_four_eight = plt.colorbar(Tpcolor_four_eight)
cbar_four_eight.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_four_eight.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array,xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_four_eight.show()
plt.savefig(direct_save+'0400_0800_map.png')
plt.show()

# Filter Nan for boxplot
Boxplot_mine_temps_four_eight_filtered = Boxplot_mine_temps_four_eight[~numpy.isnan(
    Boxplot_mine_temps_four_eight)]
Boxplot_pond_temps_four_eight_filtered = Boxplot_pond_temps_four_eight[~numpy.isnan(
    Boxplot_pond_temps_four_eight)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_four_eight_filtered != [] or Boxplot_pond_temps_four_eight_filtered !=
    []:
    fig_bp_four_eight, ax = plt.subplots(figsize=figuresize)
    # Plot Boxplot with filtered Nan data
    plt.boxplot([Boxplot_pond_temps_four_eight_filtered,
        Boxplot_mine_temps_four_eight_filtered],
                labels=['Tailings␣Pond', 'Mine'])
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.xticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.tight_layout()
    plt.savefig(direct_save+'0400_0800_boxplot.png')
    plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_four_eight = numpy.zeros((nLat, 1))
LonAxis_median_four_eight = numpy.zeros((nLon, 1))
```

```
LatAxisIndex_four_eight = numpy.empty((nLat+1,1))
LatAxisIndex_four_eight[:] = numpy.nan

LonAxisIndex_four_eight = numpy.empty((nLon+1,1))
LonAxisIndex_four_eight[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_four_eight[a] = ((Lataxis_four_eight[a])+(Lataxis_four_eight[a+1]))/2

for j in range(0, nLon):
    LonAxis_median_four_eight[j] = (Lonaxis_four_eight[j]+Lonaxis_four_eight[j+1])/2

# Save latitude/longitude indices and median temperatures
output_four_eight_filename = direct_save+'Figure_Data/Four_Eight_MedianTemp'+filename_res+'.
    txt'
outputFile_four_eight = open(output_four_eight_filename, 'w')
outputFile_four_eight.write("# Latitude, Longitude indices, median temperature and latitude/
    longitude bounds for"
                            " the mining facility, pond and mine \n")
outputFile_four_eight.write("#By: Ryan Byerlay \n")
outputFile_four_eight.write("#Recorded Time is Local Time (MDT) \n")
outputFile_four_eight.write('The nLat is: '+str(nLat)+', The nLon is: '+str(nLon)+'\n')
outputFile_four_eight.write('The Latmax is: '+str(Latmax)+', The Latmin is: '+str(Latmin)+',
    The Lonmax is: '
                            +str(Lonmax)+', The Lonmin is: '+str(Lonmin)+'\n')
outputFile_four_eight.write('The max mine lat is: '+str(max_mine_lat)+', The min mine lat is
    : '
                            +str(min_mine_lat)+', The max mine lon is: '+str(max_mine_lon)+'
                                , The min mine lon is: '
                            +str(min_mine_lon)+'\n')
outputFile_four_eight.write('The max pond lat is: '+str(max_pond_lat)+', The min pond lat is
    : '+str(min_pond_lat)+
                            ', The max pond lon is: '+str(max_pond_lon)+', The min pond lon
                                is: '+str(min_pond_lon)+
                            '\n')
outputFile_four_eight.write("#0:LatAxis_four_eight \t #1:LonAxis_four_eight"
                            " \t #2:MedianTemp_four_eight(K) {lat, lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_four_eight)-1):
    for j in range(0, len(LonAxis_four_eight)-1):
        print(TMatrix_four_eight_median[i][j])
        if numpy.isnan(TMatrix_four_eight_median[i][j]) == False:
            outputFile_four_eight.write("%f \t %f \t %f \n" % (LatAxis_median_four_eight[i],
                                                               LonAxis_median_four_eight[j],
                                                               TMatrix_four_eight_median[i][
                                                                   j]))
outputFile_four_eight.close()

#
    ##########################################################################################################################
```

```
# At 08:00 to 12:00
Lataxis_eight_twelve = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_eight_twelve = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_eight_twelve,LatAxis_eight_twelve = numpy.meshgrid(Lonaxis_eight_twelve,
    Lataxis_eight_twelve)

fig_eight_twelve, ax = plt.subplots(figsize=figuresize)
Tpcolor_eight_twelve=plt.pcolor(LonAxis_eight_twelve,LatAxis_eight_twelve,
    TMatrix_eight_twelve_median,
                                vmin=color_bar_min_eight_twelve, vmax=
                                    color_bar_max_eight_twelve)
cbar_eight_twelve = plt.colorbar(Tpcolor_eight_twelve)
cbar_eight_twelve.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_eight_twelve.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array,xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label,fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_eight_twelve.show()
plt.savefig(direct_save+'0800_1200_map.png')
plt.show()

# Filter Nan for boxplot
Boxplot_mine_temps_eight_twelve_filtered =\
    Boxplot_mine_temps_eight_twelve[~numpy.isnan(Boxplot_mine_temps_eight_twelve)]
Boxplot_pond_temps_eight_twelve_filtered =\
    Boxplot_pond_temps_eight_twelve[~numpy.isnan(Boxplot_pond_temps_eight_twelve)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_eight_twelve_filtered != [] or
    Boxplot_pond_temps_eight_twelve_filtered != []:
    fig_bp_eight_twelve, ax = plt.subplots(figsize=figuresize)
    # Plot boxplot with filtered Nan data
    plt.boxplot([Boxplot_pond_temps_eight_twelve_filtered,
        Boxplot_mine_temps_eight_twelve_filtered],
                labels=['Tailings␣Pond', 'Mine'])
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.xticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.tight_layout()
    plt.savefig(direct_save+'0800_1200_boxplot.png')
    plt.show()
```

```
# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_eight_twelve = numpy.zeros((nLat, 1))
LonAxis_median_eight_twelve = numpy.zeros((nLon, 1))

LatAxisIndex_eight_twelve = numpy.empty((nLat+1,1))
LatAxisIndex_eight_twelve[:] = numpy.nan

LonAxisIndex_eight_twelve = numpy.empty((nLon+1,1))
LonAxisIndex_eight_twelve[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_eight_twelve[a] = ((Lataxis_eight_twelve[a])+(Lataxis_eight_twelve[a+1]))
        /2

for j in range(0, nLon):
    LonAxis_median_eight_twelve[j] = (Lonaxis_eight_twelve[j]+Lonaxis_eight_twelve[j+1])/2

# Save latitude/longitude indices and median temperatures
output_eight_twelve_filename = direct_save+'Figure_Data/Eight_Twelve_MedianTemp'+
    filename_res+'.txt'
outputFile_eight_twelve = open(output_eight_twelve_filename, 'w')
outputFile_eight_twelve.write("# Latitude, Longitude indices, median temperature and
    latitude/longitude"
                              " bounds for the mining facility, pond and mine\n")
outputFile_eight_twelve.write("#By: Ryan Byerlay \n")
outputFile_eight_twelve.write("#Recorded Time is Local Time (MDT) \n")
outputFile_eight_twelve.write('The nLat is: '+str(nLat)+', The nLon is: '+str(nLon)+'\n')
outputFile_eight_twelve.write('The Latmax is: '+str(Latmax)+', The Latmin is: '+str(Latmin)+
    ', The Lonmax is: '
                              +str(Lonmax)+', The Lonmin is: '+str(Lonmin)+'\n')
outputFile_eight_twelve.write('The max mine lat is: '+str(max_mine_lat)+', The min mine lat
    is: '+str(min_mine_lat)+
                              ', The max mine lon is: '+str(max_mine_lon)+', The min mine
                                  lon is: '
                              +str(min_mine_lon)+'\n')
outputFile_eight_twelve.write('The max pond lat is: '+str(max_pond_lat)+', The min pond lat
    is: '+str(min_pond_lat)+
                              ', The max pond lon is: '+str(max_pond_lon)+', The mib pond
                                  lon is: '+str(min_pond_lon)+
                              '\n')
outputFile_eight_twelve.write("#0:LatAxis_eight_twelve \t #1:LonAxis_eight_twelve \t"
                              " #2:MedianTemp_eight_twelve(K) {lat, lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_eight_twelve)-1):
    for j in range(0, len(LonAxis_eight_twelve)-1):
        print(TMatrix_eight_twelve_median[i][j])
        if numpy.isnan(TMatrix_eight_twelve_median[i][j]) == False:
            outputFile_eight_twelve.write("%f \t %f \t %f \n" % (LatAxis_median_eight_twelve
                [i],
```

```
                                                    LonAxis_median_eight_twelve
                                                        [j],
                                                    TMatrix_eight_twelve_median
                                                        [i][j]))
outputFile_eight_twelve.close()

#
    ##########################################################################################################################

# At 12:00 to 16:00
Lataxis_twelve_sixteen = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_twelve_sixteen = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_twelve_sixteen,LatAxis_twelve_sixteen = numpy.meshgrid(Lonaxis_twelve_sixteen,
    Lataxis_twelve_sixteen)

fig_twelve_sixteen,ax = plt.subplots(figsize=figuresize)
Tpcolor_twelve_sixteen=plt.pcolor(LonAxis_twelve_sixteen,LatAxis_twelve_sixteen,
    TMatrix_twelve_sixteen_median,
                                  vmin=color_bar_min_twelve_sixteen, vmax=
                                      color_bar_max_twelve_sixteen)
cbar_twelve_sixteen = plt.colorbar(Tpcolor_twelve_sixteen)
cbar_twelve_sixteen.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_twelve_sixteen.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_twelve_sixteen.show()
plt.savefig(direct_save+'1200_1600_map.png')
plt.show()


# Filter Nan for boxplot
Boxplot_mine_temps_twelve_sixteen_filtered =\
    Boxplot_mine_temps_twelve_sixteen[~numpy.isnan(Boxplot_mine_temps_twelve_sixteen)]
Boxplot_pond_temps_twelve_sixteen_filtered =\
    Boxplot_pond_temps_twelve_sixteen[~numpy.isnan(Boxplot_pond_temps_twelve_sixteen)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_twelve_sixteen_filtered != [] or
    Boxplot_pond_temps_twelve_sixteen_filtered != []:
    fig_bp_twelve_sixteen, ax = plt.subplots(figsize=figuresize)
    # Plot Boxplot with filtered Nan data
    plt.boxplot([Boxplot_pond_temps_twelve_sixteen_filtered,
```

```
        Boxplot_mine_temps_twelve_sixteen_filtered],
                labels=['Tailings Pond', 'Mine'])
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.xticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.tight_layout()
    plt.savefig(direct_save+'1200_1600_boxplot.png')
    plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_twelve_sixteen = numpy.zeros((nLat, 1))
LonAxis_median_twelve_sixteen = numpy.zeros((nLon, 1))

LatAxisIndex_twelve_sixteen = numpy.empty((nLat+1,1))
LatAxisIndex_twelve_sixteen[:] = numpy.nan

LonAxisIndex_twelve_sixteen = numpy.empty((nLon+1,1))
LonAxisIndex_twelve_sixteen[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_twelve_sixteen[a] = ((Lataxis_twelve_sixteen[a])+(Lataxis_twelve_sixteen[
        a+1]))/2

for j in range(0, nLon):
    LonAxis_median_twelve_sixteen[j] = (Lonaxis_twelve_sixteen[j]+Lonaxis_twelve_sixteen[j
        +1])/2

# Save latitude/longitude indices and median temperatures
output_twelve_sixteen_filename = direct_save+'Figure_Data/Twelve_Sixteen_MedianTemp'+
    filename_res+'.txt'
outputFile_twelve_sixteen = open(output_twelve_sixteen_filename, 'w')
outputFile_twelve_sixteen.write("# Latitude, Longitude indices, median temperature and
    latitude/longitude"
                                "bounds for mining facility, pond and mine\n")
outputFile_twelve_sixteen.write("#By: Ryan Byerlay\n")
outputFile_twelve_sixteen.write("#Recorded Time is Local Time (MDT)\n")
outputFile_twelve_sixteen.write('The nLat is: '+str(nLat)+', The nLon is: '+str(nLon)+'\n')
outputFile_twelve_sixteen.write('The Latmax is: '+str(Latmax)+', The Latmin is: '+str(Latmin
    )+', The Lonmax is: '
                                +str(Lonmax)+', The Lonmin is: '+str(Lonmin)+'\n')
outputFile_twelve_sixteen.write('The max mine lat is: '+str(max_mine_lat)+', The min mine
    lat is: '+str(min_mine_lat)+
                                ', The max mine lon is: '+str(max_mine_lon)+', The min mine
                                    lon is: '
                                +str(min_mine_lon)+'\n')
outputFile_twelve_sixteen.write('The max pond lat is: '+str(max_pond_lat)+', The min pond
    lat is: '+str(min_pond_lat)+
                                ', The max pond lon is: '+str(max_pond_lon)+', The min pond
                                    lon is: '
                                +str(min_pond_lon)+'\n')
outputFile_twelve_sixteen.write("#0:LatAxis_twelve_sixteen \t #1:LonAxis_twelve_sixteen \t"
```

```
                                "␣#2:MedianTemp_twelve_sixteen(K)␣{lat,lon}␣␣\n")

# Save data to file
for i in range(0, len(LatAxis_twelve_sixteen)-1):
    for j in range(0, len(LonAxis_twelve_sixteen)-1):
        print(TMatrix_twelve_sixteen_median[i][j])
        if numpy.isnan(TMatrix_twelve_sixteen_median[i][j]) == False:
            outputFile_twelve_sixteen.write("%f␣\t␣%f␣\t␣%f␣\n" % (
                LatAxis_median_twelve_sixteen[i],
                                                                    LonAxis_median_twelve_sixteen
                                                                        [j],
                                                                    TMatrix_twelve_sixteen_median
                                                                        [i][j]))
outputFile_twelve_sixteen.close()

#
    ##########################################################################################################################

# At 16:00 to 20:00
Lataxis_sixteen_twenty = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_sixteen_twenty = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_sixteen_twenty,LatAxis_sixteen_twenty = numpy.meshgrid(Lonaxis_sixteen_twenty,
    Lataxis_sixteen_twenty)

fig_sixteen_twenty, ax = plt.subplots(figsize=figuresize)
Tpcolor_sixteen_twenty=plt.pcolor(LonAxis_sixteen_twenty,LatAxis_sixteen_twenty,
    TMatrix_sixteen_twenty_median,
                                  vmin=color_bar_min_sixteen_twenty, vmax=
                                      color_bar_max_sixteen_twenty)
cbar_sixteen_twenty = plt.colorbar(Tpcolor_sixteen_twenty)
cbar_sixteen_twenty.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_sixteen_twenty.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array,xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label,fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_sixteen_twenty.show()
plt.savefig(direct_save+'1600_2000_map.png')
plt.show()

# Filter Nan for boxplot
Boxplot_mine_temps_sixteen_twenty_filtered =\
    Boxplot_mine_temps_sixteen_twenty[~numpy.isnan(Boxplot_mine_temps_sixteen_twenty)]
Boxplot_pond_temps_sixteen_twenty_filtered =\
```

```
    Boxplot_pond_temps_sixteen_twenty[~numpy.isnan(Boxplot_pond_temps_sixteen_twenty)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_sixteen_twenty_filtered != [] or
    Boxplot_pond_temps_sixteen_twenty_filtered != []:
    fig_bp_sixteen_twenty, ax = plt.subplots(figsize=figuresize)
    # Plot boxplot with filtered Nan data
    plt.boxplot([Boxplot_pond_temps_sixteen_twenty_filtered,
        Boxplot_mine_temps_sixteen_twenty_filtered],
                labels=['Tailings␣Pond', 'Mine'])
    plt.xticks(fontsize=axes_bxplt_label_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.tight_layout()
    plt.savefig(direct_save+'1600_2000_boxplot.png')
    plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_sixteen_twenty = numpy.zeros((nLat, 1))
LonAxis_median_sixteen_twenty = numpy.zeros((nLon, 1))

LatAxisIndex_sixteen_twenty = numpy.empty((nLat+1,1))
LatAxisIndex_sixteen_twenty[:] = numpy.nan

LonAxisIndex_sixteen_twenty = numpy.empty((nLon+1,1))
LonAxisIndex_sixteen_twenty[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_sixteen_twenty[a] = ((Lataxis_sixteen_twenty[a])+(Lataxis_sixteen_twenty[
        a+1]))/2

for j in range(0, nLon):
    LonAxis_median_sixteen_twenty[j] = (Lonaxis_sixteen_twenty[j]+Lonaxis_sixteen_twenty[j
        +1])/2

# Save Latitude/Longitude indices and median temperatures
output_sixteen_twenty_filename = direct_save+'Figure_Data/Sixteen_Twenty_MedianTemp'+
    filename_res+'.txt'
outputFile_sixteen_twenty = open(output_sixteen_twenty_filename, 'w')
outputFile_sixteen_twenty.write("#␣Latitude,␣Longitude␣indices,␣median␣temperature␣and␣
    latitude/longitude␣bounds␣for"
                                "␣mining␣facility,␣pond␣and␣mine␣\n")
outputFile_sixteen_twenty.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_sixteen_twenty.write("#Recorded␣Time␣is␣Local␣Time␣(MDT)␣\n")
outputFile_sixteen_twenty.write('The␣nLat␣is:␣'+str(nLat)+',␣The␣nLon␣is:␣'+str(nLon)+'\n')
outputFile_sixteen_twenty.write('The␣Latmax␣is:␣'+str(Latmax)+',␣The␣Latmin␣is:␣'+str(Latmin
    )+',␣The␣Lonmax␣is:␣'
                                +str(Lonmax)+',␣The␣Lonmin␣is:␣'+str(Lonmin)+'\n')
outputFile_sixteen_twenty.write('The␣max␣mine␣lat␣is:␣'+str(max_mine_lat)+',␣The␣min␣mine␣
    lat␣is:␣'
                                +str(min_mine_lat)+',␣The␣max␣mine␣lon␣is:␣'+str(
```

```
                                        max_mine_lon)+
                                    ', The min mine lon is: '+str(min_mine_lon)+'\n')
outputFile_sixteen_twenty.write('The max pond lat is: '+str(max_pond_lat)+', The min pond
    lat is: '+str(min_pond_lat)+
                                    ', The max pond lon is: '+str(max_pond_lon)+', The min pond
                                        lon is: '
                                    +str(min_pond_lon)+'\n')
outputFile_sixteen_twenty.write("#0:LatAxis_sixteen_twenty \t #1:LonAxis_sixteen_twenty \t"
                                    " #2:MedianTemp_sixteen_twenty(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_sixteen_twenty)-1):
    for j in range(0, len(LonAxis_sixteen_twenty)-1):
        print(TMatrix_sixteen_twenty_median[i][j])
        if numpy.isnan(TMatrix_sixteen_twenty_median[i][j]) == False:
            outputFile_sixteen_twenty.write("%f \t %f \t %f \n" % (
                LatAxis_median_sixteen_twenty[i],
                                                                    LonAxis_median_sixteen_twenty
                                                                        [j],
                                                                    TMatrix_sixteen_twenty_median
                                                                        [i][j]))
outputFile_sixteen_twenty.close()

#
    ##########################################################################################################################################

# At 20:00 to 24:00
Lataxis_twenty_twentyfour = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_twenty_twentyfour = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_twenty_twentyfour,LatAxis_twenty_twentyfour = numpy.meshgrid(
    Lonaxis_twenty_twentyfour,
                                                                Lataxis_twenty_twentyfour
                                                                    )

fig_twenty_twentyfour,ax = plt.subplots(figsize=figuresize)
Tpcolor_twenty_twentyfour=plt.pcolor(LonAxis_twenty_twentyfour,LatAxis_twenty_twentyfour,
                                     TMatrix_twenty_twentyfour_median,vmin=
                                         color_bar_min_twenty_twentyfour,
                                     vmax=color_bar_max_twenty_twentyfour)
cbar_twenty_twentyfour = plt.colorbar(Tpcolor_twenty_twentyfour)
cbar_twenty_twentyfour.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_twenty_twentyfour.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon,property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon,pond_bounds_lat, c='c',s=3)
plt.scatter(base_lon, base_lat, c='w',s=launch_size)
plt.scatter(mine_bounds_lon,mine_bounds_lat, c='r',s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label,fontsize=axes_ticks_fontsize)
plt.xlabel(xaxis_label,fontsize=axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label,fontsize=axes_label_fontsize, labelpad=y_labelpad)
```

```
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_twenty_twentyfour.show()
plt.savefig(direct_save+'2000_2400_map.png')
plt.show()

# Filter Nan for boxplot
Boxplot_mine_temps_twenty_twentyfour_filtered =\
    Boxplot_mine_temps_twenty_twentyfour[~numpy.isnan(Boxplot_mine_temps_twenty_twentyfour)]
Boxplot_pond_temps_twenty_twentyfour_filtered =\
    Boxplot_pond_temps_twenty_twentyfour[~numpy.isnan(Boxplot_pond_temps_twenty_twentyfour)]

# Plot boxplot if data for either the pond OR the mine exist
if Boxplot_mine_temps_twenty_twentyfour_filtered != [] or
    Boxplot_pond_temps_twenty_twentyfour_filtered != []:
    fig_bp_twenty_twentyfour, ax = plt.subplots(figsize=figuresize)
    # Plot Boxplot with filtered Nan data
    plt.boxplot([Boxplot_pond_temps_twenty_twentyfour_filtered,
        Boxplot_mine_temps_twenty_twentyfour_filtered],
                labels=['Tailings␣Pond', 'Mine'])
    plt.ylabel(color_bar_label, fontsize=axes_bxplt_label_fontsize)
    plt.xticks(fontsize=axes_bxplt_label_fontsize)
    plt.yticks(fontsize=axes_bxplt_ticks_fontsize)
    plt.tight_layout()
    plt.savefig(direct_save+'2000_2400_boxplot.png')
    plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_twenty_twentyfour = numpy.zeros((nLat, 1))
LonAxis_median_twenty_twentyfour = numpy.zeros((nLon, 1))

LatAxisIndex_twenty_twentyfour = numpy.empty((nLat+1,1))
LatAxisIndex_twenty_twentyfour[:] = numpy.nan

LonAxisIndex_twenty_twentyfour = numpy.empty((nLon+1,1))
LonAxisIndex_twenty_twentyfour[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_twenty_twentyfour[a] = ((Lataxis_twenty_twentyfour[a])+(
        Lataxis_twenty_twentyfour[a+1]))/2

for j in range(0, nLon):
    LonAxis_median_twenty_twentyfour[j] = (Lonaxis_twenty_twentyfour[j]+
        Lonaxis_twenty_twentyfour[j+1])/2

# Save Latitude/Longitude indices and median temperatures
output_twenty_twentyfour_filename = direct_save+'Figure_Data/Twenty_Twentyfour_MedianTemp'+
    filename_res+'.txt'
outputFile_twenty_twentyfour = open(output_twenty_twentyfour_filename, 'w')
outputFile_twenty_twentyfour.write("#␣Latitude,␣Longitude␣indices,␣median␣temperature␣and"
                                   "␣latitude/longitude␣bounds␣for␣mining␣facility,␣pond␣and
```

```
                                        mine \n")
outputFile_twenty_twentyfour.write("#By: Ryan Byerlay \n")
outputFile_twenty_twentyfour.write("#Recorded Time is Local Time (MDT) \n")
outputFile_twenty_twentyfour.write('The nLat is: '+str(nLat)+', The nLon is: '+str(nLon)+'\n
    ')
outputFile_twenty_twentyfour.write('The Latmax is: '+str(Latmax)+', The Latmin is: '+str(
    Latmin)+', The Lonmax is: '
                                   +str(Lonmax)+', The Lonmin is: '+str(Lonmin)+'\n')
outputFile_twenty_twentyfour.write('The max mine lat is: '+str(max_mine_lat)+', The min mine
    lat is: '
                                   +str(min_mine_lat)+', The max mine lon is: '+str(
                                       max_mine_lon)+
                                   ', The min mine lon is: '+str(min_mine_lon)+'\n')
outputFile_twenty_twentyfour.write('The max pond lat is: '+str(max_pond_lat)+', The min pond
    lat is: '
                                   +str(min_pond_lat)+', The max pond lon is: '+str(
                                       max_pond_lon)+
                                   ', The min pond lon is: '+str(min_pond_lon)+'\n')
outputFile_twenty_twentyfour.write("#0:LatAxis_twenty_twentyfour \t #1:
    LonAxis_twenty_twentyfour \t"
                                   " #2:MedianTemp_twenty_twentyfour(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_twenty_twentyfour)-1):
    for j in range(0, len(LonAxis_twenty_twentyfour)-1):
        print(TMatrix_twenty_twentyfour_median[i][j])
        if numpy.isnan(TMatrix_twenty_twentyfour_median[i][j]) == False:
            outputFile_twenty_twentyfour.write("%f \t %f \t %f \n" % (
                LatAxis_median_twenty_twentyfour[i],
                                                                      LonAxis_median_twenty_twentyfour
                                                                          [j],
                                                                      TMatrix_twenty_twentyfour_median
                                                                          [i][j]))
outputFile_twenty_twentyfour.close()

#
    ##########################################################################################################################

# May 24 FLIR ST Plot
Lataxis_May_24_SA_FLIR = numpy.linspace(Latmin, Latmax, nLat+1)
Lonaxis_May_24_SA_FLIR = numpy.linspace(Lonmin, Lonmax, nLon+1)
LonAxis_May_24_SA_FLIR, LatAxis_May_24_SA_FLIR = numpy.meshgrid(Lonaxis_May_24_SA_FLIR,
    Lataxis_May_24_SA_FLIR)

fig_May_24_SA_FLIR, ax = plt.subplots(figsize=figuresize)
Tpcolor_May_24_SA_FLIR = plt.pcolor(LonAxis_May_24_SA_FLIR, LatAxis_May_24_SA_FLIR,
                                    TMatrix_May_24_SA_FLIR_Median, vmin=May_24_cbar_FLIR_min
                                        ,
                                    vmax=May_24_cbar_FLIR_max)
cbar_May_24_SA_FLIR = plt.colorbar(Tpcolor_May_24_SA_FLIR)
cbar_May_24_SA_FLIR.set_label(May_24_cbar_FLIR_label, labelpad=-75, y=1.1, rotation=0,
                              fontsize=May_24_cbar_FLIR_fontsize)
```

```
cbar_May_24_SA_FLIR.ax.tick_params(labelsize=May_24_axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon, property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon, pond_bounds_lat, c='c', s=3)
plt.scatter(base_lon, base_lat, c='w', s=launch_size)
plt.scatter(mine_bounds_lon, mine_bounds_lat, c='r', s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.xlabel(xaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_May_24_SA_FLIR.show()
plt.savefig(direct_save+'May_24_FLIR_ST_map.png')
plt.show()

#
    ##########################################################################################################################

# May 24 MODIS ST Plot
Lataxis_May_24_SA_MODIS = numpy.linspace(Latmin, Latmax, nLat+1)
Lonaxis_May_24_SA_MODIS = numpy.linspace(Lonmin, Lonmax, nLon+1)
LonAxis_May_24_SA_MODIS, LatAxis_May_24_SA_MODIS = numpy.meshgrid(Lonaxis_May_24_SA_MODIS,
    Lataxis_May_24_SA_MODIS)

fig_May_24_SA_MODIS, ax = plt.subplots(figsize=figuresize)
Tpcolor_May_24_SA_MODIS = plt.pcolor(LonAxis_May_24_SA_MODIS, LatAxis_May_24_SA_MODIS,
                                     TMatrix_May_24_SA_MODIS_Median, vmin=
                                         May_24_cbar_MODIS_min,
                                     vmax=May_24_cbar_MODIS_max)
cbar_May_24_SA_MODIS = plt.colorbar(Tpcolor_May_24_SA_MODIS)
cbar_May_24_SA_MODIS.set_label(May_24_cbar_MODIS_label, labelpad=-75, y=1.1, rotation=0,
                               fontsize=May_24_cbar_MODIS_fontsize)
cbar_May_24_SA_MODIS.ax.tick_params(labelsize=May_24_axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon, property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon, pond_bounds_lat, c='c', s=3)
plt.scatter(base_lon, base_lat, c='w', s=launch_size)
plt.scatter(mine_bounds_lon, mine_bounds_lat, c='r', s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.xlabel(xaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_May_24_SA_MODIS.show()
plt.savefig(direct_save+'May_24_MODIS_ST_map.png')
plt.show()

#
    ##########################################################################################################################
```

```
# May 24 Percentage Error/Absolute Error Plot
Lataxis_May_24_SA_PE = numpy.linspace(Latmin, Latmax, nLat+1)
Lonaxis_May_24_SA_PE = numpy.linspace(Lonmin, Lonmax, nLon+1)
LonAxis_May_24_SA_PE, LatAxis_May_24_SA_PE = numpy.meshgrid(Lonaxis_May_24_SA_PE,
    Lataxis_May_24_SA_PE)

print('The Median Error is: '+str(numpy.nanmedian(PE_Matrix_May_24_SA_Median)))
print('The Max Error is: '+str(numpy.nanmax(PE_Matrix_May_24_SA_Median)))
print('The Minimum Error is: '+str(numpy.nanmin(PE_Matrix_May_24_SA_Median)))

print('The Bias is: '+str(numpy.nanmean(PE_Matrix_May_24_SA_Median)))
print('The RMSE is: '+str(numpy.sqrt(numpy.nanmean(PE_Matrix_May_24_SA_Median**2))))

fig_May_24_SA_PE, ax = plt.subplots(figsize=figuresize)
Tpcolor_May_24_SA_PE = plt.pcolor(LonAxis_May_24_SA_PE, LatAxis_May_24_SA_PE,
    PE_Matrix_May_24_SA_Median,
                                  vmin=May_24_cbar_PE_min, vmax=May_24_cbar_PE_max)
cbar_May_24_SA_PE = plt.colorbar(Tpcolor_May_24_SA_PE)
cbar_May_24_SA_PE.set_label(May_24_cbar_PE_label, labelpad=-75, y=1.1, rotation=0, fontsize=
    May_24_cbar_PE_fontsize)
cbar_May_24_SA_PE.ax.tick_params(labelsize=May_24_axes_clrbar_ticks_fontsize)
plt.scatter(property_bounds_lon, property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon, pond_bounds_lat, c='c', s=3)
plt.scatter(base_lon, base_lat, c='w', s=launch_size)
plt.scatter(mine_bounds_lon, mine_bounds_lat, c='r', s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.xlabel(xaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=x_labelpad)
plt.ylabel(yaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_May_24_SA_PE.show()
# For Percentage Error
# plt.savefig(direct_save+'May_24_PE_map.png')
# For Absolute Error
plt.savefig(direct_save+'May_24_Absolute_Error_map.png')
plt.show()

#
    ##########################################################################################################################

# Mining facility, pond, and mine outline plot
fig_site_outline, ax = plt.subplots(figsize=figuresize)
plt.scatter(property_bounds_lon, property_bounds_lat, c='k', s=4.5)
plt.scatter(pond_bounds_lon, pond_bounds_lat, c='c', s=3)
plt.scatter(base_lon, base_lat, c='b', s=launch_size)
plt.scatter(mine_bounds_lon, mine_bounds_lat, c='r', s=3)
# Verifed distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xticks(xticks_array, xticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.yticks(yticks_array, yticks_label, fontsize=May_24_axes_ticks_fontsize)
plt.xlabel(xaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=x_labelpad)
```

```
plt.ylabel(yaxis_label, fontsize=May_24_axes_label_fontsize, labelpad=y_labelpad)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_site_outline.show()
plt.savefig(direct_save+'Facility_Mine_Pond_Outlines.png')
plt.show()
```

A.2.8 Principal Component Analysis (PCA)

```
import numpy
import matplotlib.pyplot as plt
import pandas as pd
from pandas import DataFrame
from math import sqrt

# Current as of October 16, 2019
# This script is to complete the PCA analysis for temperatures considering land use type
#  as a function of geographic position.
# For zero to four
filename = '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/
    Processed_Data/Separated_Hours/' \
            'Campus_Calibrated/PCA_Data/Zero_four_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])

# Keep original x
x_original = x

# Get the mean of each column
mean_lat = numpy.nanmean(x['#0:Latitude'])
mean_lon = numpy.nanmean(x['#1:Longitude'])
mean_tempK = numpy.nanmean(x['#2:Temperature(K)'])

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is: \n'+str(Cx))
print('The eigenvalues are: \n'+str(eig_vals))
print('The eigenvectors are: \n'+str(eig_vecs))
```

```
# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

# Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])

# Add the removed mean to the plotted lines
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],
                         [0+mean_lon,(eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])


# Save the line with the greatest variance, will be plotted at the end of this script
eigVal_zero_four_Line = eigVal_1_Line

#
    ##########################################################################################################################################

# For four to eight
filename = '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/
    Processed_Data/Separated_Hours/' \
           'Campus_Calibrated/PCA_Data/Four_eight_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])

# Keep original x
x_original = x

# Get mean of each column
mean_lat = numpy.nanmean(x['#0:Latitude'])
print(mean_lat)
mean_lon = numpy.nanmean(x['#1:Longitude'])
print(mean_lon)
mean_tempK = numpy.nanmean(x['#2:Temperature(K)'])
print(mean_tempK)

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
```

```
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is: \n'+str(Cx))
print('The eigenvalues are: \n'+str(eig_vals))
print('The eigenvectors are: \n'+str(eig_vecs))

# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

# Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])

# Add the removed mean to the line with the highest variance
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],
                         [0+mean_lon,(eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])

# If necessary, Reverse direction of four_eight Line to ensure that all lines are in the
    same quadrant
eigVal_four_eight_reversed = ([0,(eigVal_1_Line[0][1])],[0,(eigVal_1_Line[1][1])],[0,(
    eigVal_1_Line[2][1])])

# Save the reversed line with the greatest variance, will be plotted at the end of this
    script
eigVal_four_eight_Line = eigVal_four_eight_reversed

#
    ##########################################################################################################################

# For Eight to Twelve
filename = '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/
    Processed_Data/Separated_Hours/' \
           'Campus_Calibrated/PCA_Data/Eight_twelve_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])
```

```
# Keep original x
x_original = x

# Get mean of each column
mean_lat = numpy.mean(x['#0:Latitude'])
mean_lon = numpy.mean(x['#1:Longitude'])
mean_tempK = numpy.mean(x['#2:Temperature(K)'])

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is:\n'+str(Cx))
print('The eigenvalues are:\n'+str(eig_vals))
print('The eigenvectors are:\n'+str(eig_vecs))

# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

# Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])

# Add the removed mean to the line with the highest variance
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],[0+mean_lon,(
    eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])

# Save line with the greatest variance, will be plotted at the end of this script
eigVal_eight_twelve_Line = eigVal_1_Line
#
    ##########################################################################################################################

# For Twelve to Sixteen
filename = '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/
    Processed_Data/Separated_Hours/' \
           'Campus_Calibrated/PCA_Data/Twelve_sixteen_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
```

```
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])

# Keep original x
x_original = x

# Get mean of each column
mean_lat = numpy.mean(x['#0:Latitude'])
mean_lon = numpy.mean(x['#1:Longitude'])
mean_tempK = numpy.mean(x['#2:Temperature(K)'])

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is:\n'+str(Cx))
print('The eigenvalues are:\n'+str(eig_vals))
print('The eigenvectors are:\n'+str(eig_vecs))

# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

# Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])

# Add the removed mean to the line with the highest variance
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],[0+mean_lon,(
    eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])

# Save line with the greatest variance, will be plotted at the end of this script
eigVal_twelve_sixteen_Line = eigVal_1_Line

#
    ##########################################################################################################################
```

```
# For Sixteen to Twenty
filename = '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/
    Processed_Data/Separated_Hours/' \
           'Campus_Calibrated/PCA_Data/Sixteen_twenty_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])

# Keep original x
x_original = x

# Get mean of each column
mean_lat = numpy.mean(x['#0:Latitude'])
mean_lon = numpy.mean(x['#1:Longitude'])
mean_tempK = numpy.mean(x['#2:Temperature(K)'])

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is: \n'+str(Cx))
print('The eigenvalues are: \n'+str(eig_vals))
print('The eigenvectors are: \n'+str(eig_vecs))

# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

# Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])

# Add the removed mean to the line with the highest variance
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],[0+mean_lon,(
    eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])
```

```
# Save line with the greatest variance, will be plotted at the end of this script
eigVal_sixteen_twenty_Line = eigVal_1_Line

#
    ##########################################################################################################################

# For Twenty to Twenty four hour interval
filename= '/export/home/users/username/Documents/DG_temp/Mining_Facility_2018/Processed_Data
    /Separated_Hours/' \
          'Campus_Calibrated/PCA_Data/Twenty_twentyfour_data_calibrated_PCA.csv'

# Read in data (skip the first row with the header) and declare column names
x = pd.read_csv(filename, names=['#0:Latitude', '#1:Longitude', '#2:Temperature(K)'],
    skiprows=1)

# Get length of data
N = len(x['#0:Latitude'])

# Keep original x
x_original = x

# Get mean of each column
mean_lat = numpy.mean(x['#0:Latitude'])
mean_lon = numpy.mean(x['#1:Longitude'])
mean_tempK = numpy.mean(x['#2:Temperature(K)'])

# Create array and subtract appropriate mean from each column index
mean_x = numpy.ones((int(N),3))
mean_x = mean_x[:,0]*mean_lat, mean_x[:,1]*mean_lon, mean_x[:,2]*mean_tempK
mean_x_array = x-mean_x

# Calculate the covariance
Cx = mean_x_array.cov()

# Calculate eigenvalues and eigenvectors
eig_vals, eig_vecs = numpy.linalg.eig(Cx)
print('The covariance matrix is: \n'+str(Cx))
print('The eigenvalues are: \n'+str(eig_vals))
print('The eigenvectors are: \n'+str(eig_vecs))

# Separate Eigenvectors
eigvec1 = eig_vecs[:,0]
eigvec2 = eig_vecs[:,1]
eigvec3 = eig_vecs[:,2]

## Line coordinates to represent variances for the three most variable directions
eigVal_1_Line = ([0,(eig_vals[0]*eigvec1[0])],[0,(eig_vals[0]*eigvec1[1])], [0,(eig_vals[0]*
    eigvec1[2])])
eigVal_2_Line = ([0,(eig_vals[1]*eigvec2[0])],[0,(eig_vals[1]*eigvec2[1])], [0,(eig_vals[1]*
    eigvec2[2])])
eigVal_3_Line = ([0,(eig_vals[2]*eigvec3[0])],[0,(eig_vals[2]*eigvec3[1])], [0,(eig_vals[2]*
    eigvec3[2])])
```

```
# Add the removed mean to the line with the highest variance
eigVal_1_Line_no_mean = ([0+mean_lat,(eig_vals[0]*eigvec1[0])+mean_lat],[0+mean_lon,(
    eig_vals[0]*eigvec1[1])+mean_lon],
                         [0+mean_tempK,(eig_vals[0]*eigvec1[2])+mean_tempK])

# Reverse direction of twenty_twentyfour Line to ensure that all lines are in the same
    quadrant
eigVal_twenty_twentyfour_reversed = ([0,(eigVal_1_Line[0][1]*-1)],[0,(eigVal_1_Line
    [1][1]*-1)],
                                     [0,(eigVal_1_Line[2][1]*-1)])

# Save the reversed line with the greatest variance, will be plotted at the end of this
    script
eigVal_twenty_twentyfour_Line = eigVal_twenty_twentyfour_reversed

#
    ##########################################################################################################################

# Calculate the magnitude of each vector with the mean removed
sqrt_eigVal_zero_four_Line = sqrt((eigVal_zero_four_Line[0][1]**2)+(eigVal_zero_four_Line
    [1][1]**2))
sqrt_eigVal_four_eight_Line = sqrt((eigVal_four_eight_Line[0][1]**2)+(eigVal_four_eight_Line
    [1][1]**2))
sqrt_eigVal_eight_twelve_Line = sqrt((eigVal_eight_twelve_Line[0][1]**2)+(
    eigVal_eight_twelve_Line[1][1]**2))
sqrt_eigVal_twelve_sixteen_Line = sqrt((eigVal_twelve_sixteen_Line[0][1]**2)+(
    eigVal_twelve_sixteen_Line[1][1]**2))
sqrt_eigVal_sixteen_twenty_Line = sqrt((eigVal_sixteen_twenty_Line[0][1]**2)+(
    eigVal_sixteen_twenty_Line[1][1]**2))
sqrt_eigVal_twenty_twentyfour_Line = sqrt((eigVal_twenty_twentyfour_Line[0][1]**2)+
                                          (eigVal_twenty_twentyfour_Line[1][1]**2))

# Create normalized vectors with the mean removed from the latitude and longitude
norm_eigVal_Line_zero_four = ([0, (eigVal_zero_four_Line[0][1]/sqrt_eigVal_zero_four_Line)],
                              [0, (eigVal_zero_four_Line[1][1]/sqrt_eigVal_zero_four_Line)])
norm_eigVal_Line_four_eight = ([0, (eigVal_four_eight_Line[0][1]/sqrt_eigVal_four_eight_Line
    )],
                               [0,(eigVal_four_eight_Line[1][1]/sqrt_eigVal_four_eight_Line)
                                   ])
norm_eigVal_Line_eight_twelve = ([0, (eigVal_eight_twelve_Line[0][1]/
    sqrt_eigVal_eight_twelve_Line)],
                                 [0, (eigVal_eight_twelve_Line[1][1]/
                                     sqrt_eigVal_eight_twelve_Line)])
norm_eigVal_Line_twelve_sixteen = ([0, (eigVal_twelve_sixteen_Line[0][1]/
    sqrt_eigVal_twelve_sixteen_Line)],
                                   [0, (eigVal_twelve_sixteen_Line[1][1]/
                                       sqrt_eigVal_twelve_sixteen_Line)])
norm_eigVal_Line_sixteen_twenty = ([0, (eigVal_sixteen_twenty_Line[0][1]/
    sqrt_eigVal_sixteen_twenty_Line)],
                                   [0, (eigVal_sixteen_twenty_Line[1][1]/
                                       sqrt_eigVal_sixteen_twenty_Line)])
```

```
norm_eigVal_Line_twenty_twentyfour = ([0, (eigVal_twenty_twentyfour_Line[0][1]/
    sqrt_eigVal_twenty_twentyfour_Line)],
                                      [0, (eigVal_twenty_twentyfour_Line[1][1]/
                                          sqrt_eigVal_twenty_twentyfour_Line)])

# Create 2D plot of PCA lines with highest variances for each time interval
fig = plt.figure()
# 0000-0400
plt.plot(norm_eigVal_Line_zero_four[1], norm_eigVal_Line_zero_four[0],
         c='r', linestyle=':', marker='x', markersize=5,  linewidth=2)
# 0400-0800
plt.plot(norm_eigVal_Line_four_eight[1], norm_eigVal_Line_four_eight[0],
         c='k', marker='s', markersize=3, linewidth=2)
# 0800-1200
plt.plot(norm_eigVal_Line_eight_twelve[1], norm_eigVal_Line_eight_twelve[0],
         c='k', linestyle=':', marker='+', markersize=5, linewidth=2)
# 1200-1600
plt.plot(norm_eigVal_Line_twelve_sixteen[1], norm_eigVal_Line_twelve_sixteen[0],
         c='r', linestyle='-.', marker='d', markersize=5, linewidth=2)
# 1600-2000
plt.plot(norm_eigVal_Line_sixteen_twenty[1], norm_eigVal_Line_sixteen_twenty[0],
         c='b', linestyle='--', marker='*', markersize=5, linewidth=2)
# 2000-2400
plt.plot(norm_eigVal_Line_twenty_twentyfour[1], norm_eigVal_Line_twenty_twentyfour[0],
         c='b', marker='v', markersize=5, linewidth=2)

# Create PCA Plot
plt.plot([0, 0], [0, 1],  c='k', linewidth=0.4, zorder=1)
plt.plot([-1, 0], [0, 0], c='k', linewidth=0.4, zorder=1)
plt.ylim([0,1])
plt.xlim([-1,0])
ax = plt.gca()
ax.set_xticks([-1,0])
ax.set_xticklabels(['West', 'East'], fontsize=14)
ax.set_yticks([0,1])
ax.set_yticklabels(['South', 'North'], fontsize=14)
plt.legend(['0000-0400', '0400-0800', '0800-1200', '1200-1600', '1600-2000', '2000-2400'],
    fontsize=14)
plt.show()
```

A.3 Guelph Campaign

A.3.1 Identify TANAB2 Ascending and Descending Times

```
import sys
import numpy
import datetime

# Current as of October 16, 2019
# Use this script to create a text file of known TANAB2 starting altitides with the
```

```
#  corresponding geographic location and time stamps. This data will be used to identify the
#  base altitude for images and pixels

# From another analysis (via linear fit to the hypsometric equation)
#  define 1st order polyfit coefficients to convert pressure to altitude alt=a*p+b
a=-8.51514286e+00
b=8.62680905e+03

#
    ##########################################################################################################

# Open file with indices indicating start/end of profiles for July 28, 2018 Launch (This
    file was manually
#  concatenated/compiled together)
indices_28_07_2018 = numpy.loadtxt('/export/home/users/username/Documents/DG_Temp/'
                                   'Guelph_2018/TriSonica/Unaveraged/Cleaned_Data/18-07-28-
                                       Indices.txt')

# Separate Columns
profile_Start_28_07_2018 = indices_28_07_2018[:,0]
profile_End_28_07_2018 = indices_28_07_2018[:,1]
profile_Ascending_28_07_2018 = indices_28_07_2018[:,2]

# Initialize the starting indices from ground level
start_28_07_2018 = numpy.zeros(17)

# For the total length of the start profile array
for i in range(0, numpy.size(profile_Start_28_07_2018)):
    if (profile_Ascending_28_07_2018[i] == 1):
        for j in range(0, numpy.size(start_28_07_2018)):
            if start_28_07_2018[j] == 0:
                start_28_07_2018[j] = profile_Start_28_07_2018[i]
                break

# Import TriSonica Data for July 28, 2018 (This file was manually concatenated/compiled
    together)
fileName_28_07_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/TriSonica/
    Unaveraged/' \
                      'Cleaned_Data/18-07-28-Data.csv'
data_28_07_2018 = numpy.genfromtxt(fileName_28_07_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 13), missing_values='',
                                       filling_values=numpy.nan,
                                   delimiter=',')

# Separate Columns
year_28_07_2018 = data_28_07_2018[:,0]
month_28_07_2018 = data_28_07_2018[:,1]
day_28_07_2018 = data_28_07_2018[:,2]
hour_28_07_2018 = data_28_07_2018[:,3]
minute_28_07_2018 = data_28_07_2018[:,4]
seconds_28_07_2018 = data_28_07_2018[:,5]
P_28_07_2018 = data_28_07_2018[:,6]
```

```
# Get starting pressures from start of profile index
start_P_28_07_2018 = numpy.zeros(len(start_28_07_2018))
BaseAlt_28_07_2018 = numpy.zeros(len(start_28_07_2018))
year_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
month_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
day_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
hour_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
minute_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
seconds_base_28_07_2018 = numpy.zeros(len(start_28_07_2018))
index_28_07_2018 = numpy.zeros(len(start_28_07_2018))

# Write indices data to new variables for the start and end of each profile as per the
    imported indices file
for i in range(0,len(data_28_07_2018)):
    for j in range(0, len(start_28_07_2018)):
        if start_28_07_2018[j] == i:
            start_P_28_07_2018[j] = P_28_07_2018[i]
            year_base_28_07_2018[j] = year_28_07_2018[i]
            month_base_28_07_2018[j] = month_28_07_2018[i]
            day_base_28_07_2018[j] = day_28_07_2018[i]
            hour_base_28_07_2018[j] = hour_28_07_2018[i]
            minute_base_28_07_2018[j] = minute_28_07_2018[i]
            seconds_base_28_07_2018[j] = seconds_28_07_2018[i]
            index_28_07_2018[j] = i

# Calculate the base altitude in meters
for i in range(0, len(BaseAlt_28_07_2018)):
    BaseAlt_28_07_2018[i] =(a*start_P_28_07_2018[i])+b

# Get today's date
today_date = datetime.date.today().strftime("%B_%d_%Y")

# Save Base Altitudes file for July 28, 2018 Launch
outputFileName_28_07_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    TriSonica/Unaveraged/' \
                            'Cleaned_Data/BaseAltitudes_18-07-28.txt'
outputFile_28_07_2018 = open(outputFileName_28_07_2018, 'w')
outputFile_28_07_2018.write("# Base Altitudes for start of each profile as determined by
    location\n")
outputFile_28_07_2018.write("#By: Ryan Byerlay \n")
outputFile_28_07_2018.write("#Created on "+today_date+" \n")
outputFile_28_07_2018.write("#0:Year \t #1:Month \t #2:Day \t #3:Hour \t #4:Minute \t "
                            "#5:Seconds \t #6:Base Pressure \t #7:Base Altitude \t #8:Index
                                \n")
for i in range(0, len(BaseAlt_28_07_2018)):
    outputFile_28_07_2018.write("%i \t %i \t %i \t %i \t %i "
                                "\t %i \t %f \t %f \t %f \n" % (year_base_28_07_2018[i],
                                    month_base_28_07_2018[i],
                                                                day_base_28_07_2018[i],
                                                                    hour_base_28_07_2018[i],
                                                                minute_base_28_07_2018[i],
```

```
                                                        seconds_base_28_07_2018[i],
                                                        start_P_28_07_2018[i],
                                                            BaseAlt_28_07_2018[i],
                                                        index_28_07_2018[i]))
outputFile_28_07_2018.close()

#
    ##########################################################################################################################

# Open file with indices indicating start/end of profiles for August 13, 2018 Launch
indices_13_08_2018 = numpy.loadtxt('/export/home/users/username/Documents/DG_Temp/
    Guelph_2018/TriSonica/Unaveraged/' \
                               'Cleaned_Data/18-08-13-Indices.txt')

# Separate Columns
profile_Start_13_08_2018 = indices_13_08_2018[:,0]
profile_End_13_08_2018 = indices_13_08_2018[:,1]
profile_Ascending_13_08_2018 = indices_13_08_2018[:,2]

# Initialize the starting indices from ground level
start_13_08_2018 = numpy.zeros(11)

# For the total length of the start profile array
for i in range(0, numpy.size(profile_Start_13_08_2018)):
    if (profile_Ascending_13_08_2018[i] == 1):
        for j in range(0, numpy.size(start_13_08_2018)):
            if start_13_08_2018[j] == 0:
                start_13_08_2018[j] = profile_Start_13_08_2018[i]
                break

print(start_13_08_2018)

# Import Cleaned TriSonica Data from August 13, 2018
fileName_13_08_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/TriSonica/
    Unaveraged/' \
                              'Cleaned_Data/18-08-13_Data.csv'
data_13_08_2018 = numpy.genfromtxt(fileName_13_08_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 13), missing_values='',
                                       filling_values=numpy.nan,
                                   delimiter=',')

# Separate Columns
year_13_08_2018 = data_13_08_2018[:,0]
month_13_08_2018 = data_13_08_2018[:,1]
day_13_08_2018 = data_13_08_2018[:,2]
hour_13_08_2018 = data_13_08_2018[:,3]
minute_13_08_2018 = data_13_08_2018[:,4]
seconds_13_08_2018 = data_13_08_2018[:,5]
P_13_08_2018 = data_13_08_2018[:,6]

# Get starting pressures from start of profile index
start_P_13_08_2018 = numpy.zeros(len(start_13_08_2018))
```

```
BaseAlt_13_08_2018 = numpy.zeros(len(start_13_08_2018))
year_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
month_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
day_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
hour_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
minute_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
seconds_base_13_08_2018 = numpy.zeros(len(start_13_08_2018))
index_13_08_2018 = numpy.zeros(len(start_13_08_2018))

# Write indices data to new variables for the start and end of each profile as per the
    imported indices file
for i in range(0,len(data_13_08_2018)):
    for j in range(0, len(start_13_08_2018)):
        if start_13_08_2018[j] == i:
            start_P_13_08_2018[j] = P_13_08_2018[i]
            year_base_13_08_2018[j] = year_13_08_2018[i]
            month_base_13_08_2018[j] = month_13_08_2018[i]
            day_base_13_08_2018[j] = day_13_08_2018[i]
            hour_base_13_08_2018[j] = hour_13_08_2018[i]
            minute_base_13_08_2018[j] = minute_13_08_2018[i]
            seconds_base_13_08_2018[j] = seconds_13_08_2018[i]
            index_13_08_2018[j] = i

# Calculate the base altitude in meters
for i in range(0, len(BaseAlt_13_08_2018)):
    BaseAlt_13_08_2018[i] =(a*start_P_13_08_2018[i])+b

# Save Base Altitudes file for August 13, 2018 Launch
outputFileName_13_08_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    TriSonica/Unaveraged/' \
                            'Cleaned_Data/BaseAltitudes_18-08-13.txt'
outputFile_13_08_2018 = open(outputFileName_13_08_2018, 'w')
outputFile_13_08_2018.write("# Base Altitudes for start of each profile as determined by the
    TriSonica \n")
outputFile_13_08_2018.write("#By: Ryan Byerlay \n")
outputFile_13_08_2018.write("#Created on "+today_date+" \n")
outputFile_13_08_2018.write("#0:Year \t #1:Month \t #2:Day \t #3:Hour \t #4:Minute \t "
                            "#5:Seconds \t #6:Base Pressure \t #7:Base Altitude \t #8:Index
                                \n")
for i in range(0, len(BaseAlt_13_08_2018)):
    outputFile_13_08_2018.write("%i \t %i \t %i \t %i \t %i \t %i "
                                "\t %f \t %f \t %f \n" % (year_base_13_08_2018[i],
                                    month_base_13_08_2018[i],
                                                         day_base_13_08_2018[i],
                                                             hour_base_13_08_2018[i],
                                                         minute_base_13_08_2018[i],
                                                             seconds_base_13_08_2018[i],
                                                         start_P_13_08_2018[i],
                                                             BaseAlt_13_08_2018[i],
                                                         index_13_08_2018[i]))
outputFile_13_08_2018.close()
```

```
#
    ##########################################################################################################

# Calculate day of year in seconds and save Unaveraged concatenated Base Altitude values for
     urban data set
# Concatenate columns
year = numpy.concatenate([year_base_28_07_2018, year_base_13_08_2018])
month = numpy.concatenate([month_base_28_07_2018, month_base_13_08_2018])
day = numpy.concatenate([day_base_28_07_2018, day_base_13_08_2018])
hour = numpy.concatenate([hour_base_28_07_2018, hour_base_13_08_2018])
minute = numpy.concatenate([minute_base_28_07_2018, minute_base_13_08_2018])
seconds = numpy.concatenate([seconds_base_28_07_2018, seconds_base_13_08_2018])
BaseAltitude = numpy.concatenate([BaseAlt_28_07_2018, BaseAlt_13_08_2018])
index = numpy.concatenate([index_28_07_2018, index_13_08_2018])
BasePressure = numpy.concatenate([start_P_28_07_2018, start_P_13_08_2018])

# Initialize variable for day of year in seconds
doy_sec = numpy.zeros(len(day))

for i in range(0, len(day)):
    # For July 28, 2018
    if day[i] == 28:
        # Day of year * Hours * Minutes * Seconds
        doy_sec[i] = (209 * 24 * 60 * 60) + (hour[i] * 60 * 60) + (minute[i]*60) + seconds[i
            ]

    # For August 13, 2018
    elif day[i] == 13:
        # DOY * Hours * Minutes * Seconds
        doy_sec[i] = (225 * 24 * 60 * 60) + (hour[i] * 60 * 60) + (minute[i] * 60) + seconds
            [i]

    else:
        print('More Dates need to be included above')

# Save Unaveraged concatenated variables and altitudes to file
outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                 'TriSonica/Unaveraged/TANAB2_Ascending_Indices.txt'
outputFile = open(outputFileName, 'w')
outputFile.write("# Data collected by TriSonica for July/August Guelph Urban Field Campaign.
    "
                 " Includes Altitude calculation \n")
outputFile.write("#By: Ryan Byerlay \n")
outputFile.write("#Created on "+today_date+" \n")
outputFile.write("#Recorded Time is Local Time (EDT) \n")
outputFile.write("#0:Year \t #1:Month \t #2:Day \t #3:Hour \t #4:Minute \t #5:Seconds "
                 "\t #6:DOY In Minutes \t #7:Base Pressure \t #8:Base Altitude \t #9:Index \
                     n")
for i in range(0, len(year)):
    outputFile.write("%i \t %i \t %i \t %i \t %i "
                     "\t %f \t %f \t %f \t %f \t %i \n" % (year[i], month[i], day[i], hour[i
                         ], minute[i],
```

```
                                                    seconds[i], doy_sec[i],
                                                        BasePressure[i],
                                                    BaseAltitude[i], index[i]))
outputFile.close()
```

A.3.2 TriSonica Atmospheric Pressure to Altitude

```
import numpy
import datetime

# Current as of October 18, 2019
# Code to load in TriSonica data (from the two UofG TANAB2 launches),
#  concatenate data for the two days and calculate the 1 second averaged data

# July 28/2018
fileName_28_07_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                      'TriSonica/Unaveraged/Cleaned_Data/18-07-28-Data.csv'

data_28_07_2018 = numpy.genfromtxt(fileName_28_07_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                   filling_values=numpy.nan, delimiter=',')

# Separate columns
year_28_07_2018 = data_28_07_2018[:,0]
month_28_07_2018 = data_28_07_2018[:,1]
day_28_07_2018 = data_28_07_2018[:,2]
hour_28_07_2018 = data_28_07_2018[:,3]
minute_28_07_2018 = data_28_07_2018[:,4]
seconds_28_07_2018 = data_28_07_2018[:,5]
temp_28_07_2018 = data_28_07_2018[:,6]
P_28_07_2018 = data_28_07_2018[:,7]

# August 13/2018
fileName_13_08_2018 = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                      'TriSonica/Unaveraged/Cleaned_Data/18-08-13_Data.csv'

data_13_08_2018 = numpy.genfromtxt(fileName_13_08_2018, skip_header=5, invalid_raise=False,
                                   usecols=(0, 1, 2, 3, 4, 5, 11, 13), missing_values='',
                                   filling_values=numpy.nan, delimiter=',')

# Separate columns
year_13_08_2018 = data_13_08_2018[:,0]
month_13_08_2018 = data_13_08_2018[:,1]
day_13_08_2018 = data_13_08_2018[:,2]
hour_13_08_2018 = data_13_08_2018[:,3]
minute_13_08_2018 = data_13_08_2018[:,4]
seconds_13_08_2018 = data_13_08_2018[:,5]
temp_13_08_2018 = data_13_08_2018[:,6]
P_13_08_2018 = data_13_08_2018[:,7]

# Concatenate columns for all days
```

```
year = numpy.concatenate([year_28_07_2018, year_13_08_2018])
month = numpy.concatenate([month_28_07_2018, month_13_08_2018])
day = numpy.concatenate([day_28_07_2018, day_13_08_2018])
hour = numpy.concatenate([hour_28_07_2018, hour_13_08_2018])
minute = numpy.concatenate([minute_28_07_2018, minute_13_08_2018])
seconds = numpy.concatenate([seconds_28_07_2018, seconds_13_08_2018])
temp = numpy.concatenate([temp_28_07_2018, temp_13_08_2018])
P = numpy.concatenate([P_28_07_2018, P_13_08_2018])

#
    ##########################################################################################################################

# Calculate Altitudes from pressure
# From another analysis  (via first order fit of hypsometric equation) define 1st order
    polyfit
#  coefficients to convert pressure to altitude alt=a*p+b
a=-8.51514286e+00
b=8.62680905e+03

# Initialize altitude array
altitude = numpy.zeros(len(year))

# Calculate Altitude
for i in range(0, len(altitude)):
    altitude[i] =(a*P[i])+b

# Calculate the number of seconds passed since the beginning of 2018 based on day of year
doy_sec = numpy.zeros(len(year))
for i in range(0, len(year)):
    # For July 28, 2018
    if day[i] == 28:
        doy_sec[i] = (209*24*60*60)+(hour[i]*60*60)+(minute[i]*60)+seconds[i]

    # For August 13, 2018
    elif day[i] == 13:
        doy_sec[i] = (225*24*60*60)+(hour[i]*60*60)+(minute[i]*60)+seconds[i]

    else:
        print('More dates need to be included above')

# Save Unaveraged concatenated variables and altitude to file
today_date = datetime.date.today().strftime("%B_%d_%Y")

outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/TriSonica' \
                 '/Unaveraged/TriSonica_Unaveraged_2018_Urban_Campaign_Altitudes.txt'
outputFile = open(outputFileName, 'w')
outputFile.write("# Data collected by TriSonica for UofG Campus Field Campaigns. Includes
    Altitude calculation \n")
outputFile.write("#By: Ryan Byerlay \n")
outputFile.write("#Created on "+today_date+" \n")
outputFile.write("#Recorded Time is Local Time (EDT) \n")
outputFile.write("#0:Year \t #1:Month \t #2:Day \t #3:Hour \t #4:Minute \t #5:Seconds [sec]
```

```
    \t #6:DOY In Seconds [sec]"
                "\t\t #7:Pressure [hPa]\t #8:Altitude [m]\t #9: Temperature [degC]\n")
for i in range(0, len(year)):
    # Check for Pressure values of 0 and skip
    if P[i] == 0:
        continue
    else:
        outputFile.write("%i \t %i \t %i \t %i \t %i \t %f \t %f \t %f "
                         "\t %f \t %f \n" % (year[i], month[i], day[i], hour[i], minute[i],
                             seconds[i], doy_sec[i],
                                          P[i], altitude[i], temp[i]))

outputFile.close()

#
    ##########################################################################################################################

# Complete 1 second averaging
# Second Averaging (10 Hz Frequency)
AverageSample = 10

# Total number of samples
Ntotal = numpy.size(altitude)
NSample = int(Ntotal/AverageSample)

# Calculate 1 second averaged altitudes and year, month, day, hour, minute, second, pressure
    , doy second, and
#  temperature averages for each sample
yearavg = numpy.zeros((NSample,1))
monthavg = numpy.zeros((NSample,1))
houravg = numpy.zeros((NSample,1))
dayavg = numpy.zeros((NSample,1))
minuteavg = numpy.zeros((NSample,1))
secondavg = numpy.zeros((NSample,1))
pressureavg = numpy.zeros((NSample,1))
altitudeavg = numpy.zeros((NSample,1))
doy_sec_avg = numpy.zeros((NSample,1))
temp_avg = numpy.zeros((NSample,1))

for i in range(0, NSample):
    # Calculate Averages
    yearavg[i] = numpy.mean(year[i*AverageSample:(i+1)*AverageSample])
    monthavg[i] = numpy.mean(month[i*AverageSample:(i+1)*AverageSample])
    houravg[i] = numpy.mean(hour[i*AverageSample:(i+1)*AverageSample])
    dayavg[i] = numpy.mean(day[i*AverageSample:(i+1)*AverageSample])
    minuteavg[i] = numpy.mean(minute[i*AverageSample:(i+1)*AverageSample])
    secondavg[i] = numpy.mean(seconds[i*AverageSample:(i+1)*AverageSample])
    pressureavg[i] = numpy.mean(P[i*AverageSample:(i+1)*AverageSample])
    altitudeavg[i] = numpy.mean(altitude[i*AverageSample:(i+1)*AverageSample])
    doy_sec_avg[i] = numpy.mean(doy_sec[i*AverageSample:(i+1)*AverageSample])
    temp_avg[i] = numpy.mean(temp[i*AverageSample:(i+1)*AverageSample])
```

```
# The following is to the fix the issue where in the averaged second column, the 59th
    averaged
#  second does not equal 59, instead it is a lower value as I think this is whats
#  happening: 58, 59, 0 are being averaged so the value will be < 59
# Note: As of May 2/2019 This problem occurs when the U-99.99 etc are manually deleted (from
     raw TriSonica file),
# within 1 second of data, 10 data points do not always exist
# Put condition in to omit Nan values in the base altitude section too
# When saving, put in a check to see if the values are Nan, if index has Nan values, then
    skip
for i in range(0, len(secondavg)):
    print(secondavg[i])
    if numpy.isnan(secondavg[i]) == True:
        continue
    else:
        if int(secondavg[i]) == 58 and int(secondavg[i+1]) != 59:
            yearavg[i+1] = numpy.nan
            monthavg[i+1] = numpy.nan
            houravg[i+1] = numpy.nan
            dayavg[i+1] = numpy.nan
            minuteavg[i+1] = numpy.nan
            secondavg[i+1] = numpy.nan
            pressureavg[i+1] = numpy.nan
            altitudeavg[i+1] = numpy.nan
            doy_sec_avg[i+1] = numpy.nan
            temp_avg[i+1] = numpy.nan

# Call in TriSonica Ascending TANAB2 launch base altitude data
fileName_BaseAlt = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/TriSonica/
    Unaveraged/' \
                   'TANAB2_Ascending_Indices.txt'

data_BaseAlt = numpy.genfromtxt(fileName_BaseAlt, skip_header=5)

# Separate columns
day_BaseAlt = data_BaseAlt[:,2]
hour_BaseAlt = data_BaseAlt[:,3]
minute_BaseAlt = data_BaseAlt[:,4]
seconds_BaseAlt = data_BaseAlt[:,5]
doy_sec_BaseAlt = data_BaseAlt[:,6]
BasePressure_trisonica = data_BaseAlt[:,7]
BaseAlt_trisonica = data_BaseAlt[:,8]

# Initialize variable that stores the difference between day of year in seconds for the
    TriSonica TANAB2 ascending
#  indices and for the second averaged file
delta_doy = numpy.zeros((len(data_BaseAlt),1))

# Initialize variables to calculate altitude above the ground
BaseAltitude = numpy.zeros((NSample,1))
BasePressure = numpy.zeros((NSample,1))
Location = numpy.empty(NSample, dtype=str)
```

```
index_Base = numpy.zeros((NSample,1))

# Match day of year in seconds indices from TANAB2 ascending indices file & second averaged
    data
# loop through averaged dy of year in seconds indices
for i in range(0, len(yearavg)):
    # Skip Nan values
    if numpy.isnan(yearavg[i]) == True:
        continue
    else:
        # loop through Base Altitude indices
        for j in range(0, len(day_BaseAlt)):
            # Calculate difference between times
            delta_doy[j] = abs(doy_sec_BaseAlt[j]-doy_sec_avg[i])

            # When at the last value of the TriSonica TANAB2 Ascending Indices file,
            #  find index of minimum value and write corresponding Altitude to file
            # Also record index of the Base Altitude file
            if j == (len(BaseAlt_trisonica)-1):
                BaseAltitude[i] = BaseAlt_trisonica[numpy.argmin(delta_doy)]
                BasePressure[i] = BasePressure_trisonica[numpy.argmin(delta_doy)]
                index_Base[i] = numpy.argmin(delta_doy[j])

# Initialize variables for deltaPressure and deltaAltitude
deltaPressure = numpy.zeros((NSample,1))
deltaAltitude  = numpy.zeros((NSample,1))

# Calculate deltaPressure (change in pressure with respect to the start of the TANAB2 launch
    ) and deltaAltitude
#  (change in altitude) with respect to the ground
for i in range(0, NSample):
    # skip Nan values
    if numpy.isnan(pressureavg[i]) == True:
        continue
    else:
        deltaPressure[i] = abs(BasePressure[i]-pressureavg[i])
        deltaAltitude[i] = abs(BaseAltitude[i]-altitudeavg[i])

# Save Averaged data to file
outputFileName_avg = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/TriSonica/'
    \
                     'TriSonica_Averaged_Guelph_Urban_Altitudes.txt'

outputFile_avg = open(outputFileName_avg, 'w')
outputFile_avg.write("# Second Averaged Data collected by TriSonica for 2018 Guelph Urban
    field campaign."
                     " Includes Altitude calculation \n")
outputFile_avg.write("#By: Ryan Byerlay \n")
outputFile_avg.write("#Created on "+today_date+" \n")
outputFile_avg.write("#Recorded Time is Local Time, EDT \n")
outputFile_avg.write("#0:AvgYear \t #1:AvgMonth \t #2:AvgDay \t #3:AvgHour \t #4:AvgMinute \
    t #5:AvgSeconds \t"
```

```
                    "␣#6:Day␣of␣Year␣in␣Seconds␣\t␣#7:AvgPressure␣\t␣#8:BasePressure␣\t␣#9:
                        DeltaPressure␣\t"
                    "␣#10:AvgAltitude␣\t␣#11:BaseAltitude␣\t␣#12:DeltaAltitude␣\t␣#13:
                        Temperature(degC)␣\t"
                    "␣#14:Index␣in␣BaseAlt␣File␣\n")
for i in range(0, NSample):
    # If index has Nan value, skip and do not write
    if numpy.isnan(yearavg[i]) == True:
        continue
    else:
        outputFile_avg.write("%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%f␣\t␣%f␣\t␣%f␣\t␣%f␣\t␣%f
            ␣\t␣%f␣"
                             "\t␣%f␣\t␣%f␣\t␣%i␣\n" % (yearavg[i], monthavg[i], dayavg[i],
                                 houravg[i], minuteavg[i],
                                                 secondavg[i], doy_sec_avg[i],
                                                     pressureavg[i], BasePressure[i],
                                                 deltaPressure[i], altitudeavg[i],
                                                     BaseAltitude[i],
                                                 deltaAltitude[i], temp_avg[i],
                                                     index_Base[i]))
outputFile_avg.close()
```

A.3.3 Spatial Coordinate Grid Overlaid on University of Guelph Campus

```
import numpy
import math
from math import radians
from math import degrees
from math import asin
from math import tan
from math import atan2
from math import sin
from math import cos
from math import acos
from math import sqrt
from math import atan

def SiteCoordinatesCalc(res_x, y_iterator, TLeft_lat_rads, R, TLeft_lon_rads, GPS_matrix):
    for a in range(0, res_x):
        for b in range(0, res_y):
            if a == 0 and b == 0:
                continue
            elif a == 0 and b != 0:
                d_km = y_iterator[b-1]
                Yaw = 180
                Yaw_rads = math.radians(Yaw)

                lat2 = math.asin(math.sin(TLeft_lat_rads) * math.cos(d_km / R) + math.cos(
                    TLeft_lat_rads)
```

```
                    * math.sin(d_km / R) * math.cos(Yaw_rads))
        lon2 = TLeft_lon_rads + math.atan2(math.sin(Yaw_rads) * math.sin(d_km / R) *
            math.cos(TLeft_lat_rads),
                                           math.cos(d_km / R) - math.sin(
                                               TLeft_lat_rads) * math.sin(lat2))

        # Convert back to decimal degrees
        lat2 = math.degrees(lat2)
        lon2 = math.degrees(lon2)

        # Save to GPS Matrix
        GPS_matrix[a][b][0] = lat2
        GPS_matrix[a][b][1] = lon2
        continue

    elif b == 0 and a != 0:
        d_km = x_iterator[a-1]
        Yaw = 90
        Yaw_rads = math.radians(Yaw)

        lat2 = math.asin(math.sin(TLeft_lat_rads) * math.cos(d_km / R) + math.cos(
            TLeft_lat_rads) *
                         math.sin(d_km / R) * math.cos(Yaw_rads))
        lon2 = TLeft_lon_rads + math.atan2(math.sin(Yaw_rads) * math.sin(d_km / R) *
            math.cos(TLeft_lat_rads),
                                           math.cos(d_km / R) - math.sin(
                                               TLeft_lat_rads) * math.sin(lat2))

        # Convert back to decimal degrees
        lat2 = math.degrees(lat2)
        lon2 = math.degrees(lon2)

        # Save to GPS Matrix
        GPS_matrix[a][b][0] = lat2
        GPS_matrix[a][b][1] = lon2
        continue

    else:
        d_km = y_iterator[b]
        Yaw = 180
        Yaw_rads = math.radians(Yaw)

        lat2 = math.asin(math.sin(math.radians(GPS_matrix[a][0][0]))*math.cos(d_km /
            R) +
                         math.cos(math.radians(GPS_matrix[a][0][0]))*math.sin(d_km/R
                             )*math.cos(Yaw_rads))

        lon2 = math.radians(GPS_matrix[a][0][1]) + \
               math.atan2(math.sin(Yaw_rads) * math.sin(d_km/R)*math.cos(math.
                   radians(GPS_matrix[a][0][0])),
                          math.cos(d_km/R)-math.sin(math.radians(GPS_matrix[a
                              ][0][0]))*math.sin(lat2))
```

```
                # Convert back to decimal degrees
                lat2 = math.degrees(lat2)
                lon2 = math.degrees(lon2)

                # Save to GPS Matrix
                GPS_matrix[a][b][0] = lat2
                GPS_matrix[a][b][1] = lon2

    return GPS_matrix

# Get GPS coordinates for an approximately 15 km by 13 km rectangle around the University of
    Guelph TANAB2 launch
#  with a spatial resolution of 500m

# Identify the Top Left latitude and longitude
# Use the top left as a reference point
TLeft_lat = 43.594589
TLeft_lon = -80.331025

# Convert degrees to rads
TLeft_lat_rads = radians(TLeft_lat)
TLeft_lon_rads = radians(TLeft_lon)

# Top right latitude/longitude
TRight_lat = TLeft_lat
TRight_lon = -80.145714

# Bottom right latitude/longitude
# Starting latitude/longitude
BRight_lat = 43.472844
BRight_lon = -80.145714

# Bottom left latitude/longitude
BLeft_lat = BRight_lat
BLeft_lon = -80.331025

# Equatorial radius of earth in km as per: https://nssdc.gsfc.nasa.gov/planetary/factsheet/
    earthfact.html
R = 6378.1

# Number of bins per resolution
# 500m resolution
res500_x = 30
res500_y = 28

# 100m resolution
res100_x = 150
res100_y = 140

# 1km resolution
res1000_x = 15
```

```
res1000_y = 14

# For Code below use the following resolution
res_x = res500_x
res_y = res500_y

x_iterator = numpy.zeros((res_x,1))
y_iterator = numpy.zeros((res_y,1))

# Initialize array for latitude and longitude calculation
GPS_matrix = numpy.zeros((res_x, res_y, 2))

# Assign Known GPS latitude/longitude
GPS_matrix[0][0][0] = TLeft_lat
GPS_matrix[0][0][1] = TLeft_lon

# The number of GPS coordinates to save to file
len_save = res_x*res_y
save_GPS_matrix = numpy.zeros((len_save,2))
save_GPS_matrix[:] = numpy.nan

# Set up x_iterator vector for left to right GPS coordinates
if res_x == res100_x:
    d_iterator = 0.1
elif res_x == res500_x:
    d_iterator = 0.5
elif res_x == res1000_x:
    d_iterator = 1

for i in range(0, res_x):
    if i == 0:
        x_iterator[i] = d_iterator
    else:
        x_iterator[i] = d_iterator*(i+1)

# Set up y_iterator vector for top to bottom GPS coordinates
for i in range(0, res_y):
    if i == 0:
        y_iterator[i] = d_iterator
    else:
        y_iterator[i] = d_iterator*(i+1)


# Calculate new latitude and longitude coordinates 500m apart from each other and save to
    text file
SiteCoordinatesCalc(res_x, y_iterator, TLeft_lat_rads, R, TLeft_lon_rads, GPS_matrix)

for i in range(0,res_x):
    for j in range(0,res_y):
        for k in range(0, len_save):
            if numpy.isnan(save_GPS_matrix[k][0]) == True:
                save_GPS_matrix[k][0] = GPS_matrix[i][j][0]
```

```
                save_GPS_matrix[k][1] = GPS_matrix[i][j][1]
                break

# Save Data to file, name file based on spatial resolution
if res_x == res100_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                     'MODIS/Emissivity/100m_Resolution_GPS_Data.csv'
elif res_x == res500_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                     'MODIS/Emissivity/500m_Resolution_GPS_Data.csv'
elif res_x == res1000_x:
    outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                     'MODIS/Emissivity/1000m_Resolution_GPS_Data.csv'

outputFile = open(outputFileName, 'w')
numpy.savetxt(outputFileName, save_GPS_matrix, delimiter=',', fmt='%f', header='#0:Lat,#1:
    Lon')
```

A.3.4 Direct Georeferencing and Temperature Calculation

```
# Extract temperatures (degC and K) from individual pixels within each image
# Extract longitude and latitude for each image
# Created By: Ryan Byerlay On: April 26, 2018, Current as of: October 18, 2019

# Successfully works on ImageMagick (IM) 7.0.7 and ExifTool 10.94 on a linux OS (both Ubuntu
    16.04 and Ubuntu 18.04)
#  with Python 3.5
# NOTE: Syntax for IM before version 7 is different
# NOTE: The Raw Thermal Image Type must be TIFF, to check put image in same folder as IM7,
    ExifTool and this script
#  and type the following into the command line: "ExifTool filename.jpg"
# NOTE: May need to install Ubuntu/Linux developer tools for TIFF, PNG, JPEG etc as IM 7 may
#  not be able to process images
# NOTE: Updated versions of ExifTool may have more functionality for FLIR Images, may result
    in improved
#  quantitative image analysis
# Check here for the latest on ExifTool: https://www.sno.phy.queensu.ca/~phil/exiftool/
#
    ##################################################################################################################


import os
import subprocess
import numpy
import time
import math
from math import tan
from math import sin
from math import cos
from math import asin
from math import sqrt
```

```
from math import pi
from math import fabs
from math import atan2
from math import radians
from math import atan
import datetime
import simplekml
from numba import jit
import pytemperature

# Data derived from the Advanced Land Observing Satellite (ALOS) Digital Surface Model
#  (DSM) Version 2.1 file with a spatial resolution of 30 m
def LandSlopeEquations (BaseAltitude, heading):
    # Convert the heading from a float to an int
    heading_int = int(heading)

    # Degree of poly fit. Use degree of 3 for Earth surface elevation as
    #  per: https://doi.org/10.1080/13658810310001596058
    poly_deg = 3
    # For the TANAB2 launch location at Reek Walk, University of Guelph, Guelph, Ontario,
        Canada
    # For the North direction
    # Between N (0 deg) and NNE (22.5 deg) or NNW (337.5 deg) and N (360 deg)
    if heading_int >= 0 and heading_int < 22.5 or heading_int > 337.5 and heading_int <=360:
        # Elevation data for 10 km due North of the Reek Walk launch location
        Guelph_N_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                            'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_N.csv'

        # Load data from file
        Guelph_N_data = numpy.genfromtxt(Guelph_N_filename, delimiter=',')

        # Distance away from the TANAB2 launch location in meters
        distance_Guelph_N = Guelph_N_data[:, 0]

        # Elevation above sea level in meters for each data point away from the TANAB2
            launch location
        elevation_Guelph_N = Guelph_N_data[:, 3]

        # Returns coefficients for the polyfit equation between the distance away from the
        #  TANAB2 launch location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_N, elevation_Guelph_N, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
        #  TANAB2 Launch location in the North direction
        # Elevation above sea level
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_N)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial
        #  with respect to the elevation data derived from the ALOS DSM file
        # Elevation above ground
        ground_elev_AGL = elevation_Guelph_N - ground_elev_ASL_fitted
```

```
    return ground_elev_ASL_fitted, ground_elev_AGL

# For the North East direction
# Between NNE (22.5 deg) and ENE (67.5 deg)
elif heading_int > 22.5 and heading_int < 67.5:
    # Elevation data for 10 km due North East of the Reek Walk launch location
    Guelph_NE_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                         'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_NE.csv'

    # Load data from file
    Guelph_NE_data = numpy.genfromtxt(Guelph_NE_filename, delimiter=',')

    # Distance away from the TANAB2 launch location in meters
    distance_Guelph_NE = Guelph_NE_data[:, 0]

    # Elevation above sea level in meters for each data point away from the TANAB2
        launch location in meters
    elevation_Guelph_NE = Guelph_NE_data[:, 3]

    # Returns coefficients for the polyfit equation between the distance away from the
        TANAB2
    #  launch location and the corresponding surface elevation above sea level in meters
    Land_Poly_Coeff = numpy.polyfit(distance_Guelph_NE, elevation_Guelph_NE, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from the
        TANAB2
    #  Launch location in the North East direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_NE)

    # Detrend the resulting land surface elevations from the derived polynomial with
        respect to the
    #  elevation data derived from the ALOS DSM file
    ground_elev_AGL = elevation_Guelph_NE - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the East direction
# Between ENE (67.5 deg) and ESE (112.5 deg)
elif heading_int > 67.5 and heading_int < 112.5:
    # Elevation data for 10 km due East of the Reek Walk launch location
    Guelph_E_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                        'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_E.csv'

    # Load data from file
    Guelph_E_data = numpy.genfromtxt(Guelph_E_filename, delimiter=',')

    # Distance away from the TANAB2 launch location in meters
    distance_Guelph_E = Guelph_E_data[:, 0]

    # Elevation above sea level in meters for each data point away from the TANAB2
        launch location
```

```
        elevation_Guelph_E = Guelph_E_data[:, 3]

        # Returns coefficients for the polyfit equation between the distance away from the
            TANAB2 launch
        #  location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_E, elevation_Guelph_E, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
            TANAB2
        #  Launch location in the East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_E)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial with respect to the
        #  elevation data derived from the ALOS DSM file
        ground_elev_AGL = elevation_Guelph_E - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the South East direction
    # Between ESE (112.5 deg) and SSE (157.5 deg)
    elif heading_int > 112.5 and heading_int < 157.5:
        # Elevation data for 10 km due South East of the Reek Walk launch location
        Guelph_SE_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                             'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_SE.csv'

        # Load data from file
        Guelph_SE_data = numpy.genfromtxt(Guelph_SE_filename, delimiter=',')

        # Distance away from TANAB2 launch location in meters
        distance_Guelph_SE = Guelph_SE_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2 launch
            location
        elevation_Guelph_SE = Guelph_SE_data[:, 3]

        # Returns coefficients for the polyfit equation between the distance away from the
            TANAB2 launch
        #  location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_SE, elevation_Guelph_SE, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
            TANAB2
        #  Launch location in the South East direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_SE)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial with respect to the
        #  elevation data derived from the ALOS DSM file
        ground_elev_AGL = elevation_Guelph_SE - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL
```

```
# For the South direction
# Between SSE (157.5 deg) and SSW (202.5 deg)
elif heading_int > 157.5 and heading_int < 202.5:
    # Elevation data for 10 km due South of the Reek Walk launch location
    Guelph_S_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                        'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_S.csv'

    # Load data from file
    Guelph_S_data = numpy.genfromtxt(Guelph_S_filename, delimiter=',')

    # Distance away from TANAB2 launch location in meters
    distance_Guelph_S = Guelph_S_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2 launch
        location
    elevation_Guelph_S = Guelph_S_data[:, 3]

    # Returns coefficients for the polyfit equation between the distance away from the
        TANAB2
    #  launch location and the corresponding elevation above sea level in meters
    Land_Poly_Coeff = numpy.polyfit(distance_Guelph_S, elevation_Guelph_S, poly_deg)

    # Evaluate the polynomial at specific values as given by the distance away from the
        TANAB2
    #  Launch location in the South direction
    ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_S)

    # Detrend the resulting land surface elevations calculated from the derived
        polynomial with respect
    #  to the elevation data derived from the ALOS DSM file
    ground_elev_AGL = elevation_Guelph_S - ground_elev_ASL_fitted

    return ground_elev_ASL_fitted, ground_elev_AGL

# For the South West direction
# Between SSW (202.5 deg) and WSW (247.5 deg)
elif heading_int > 202.5 and heading_int < 247.5:
    # Elevation data for 10 km due South West of the Reek Walk launch location
    Guelph_SW_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                        'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_SW.csv'

    # Load data from file
    Guelph_SW_data = numpy.genfromtxt(Guelph_SW_filename, delimiter=',')

    # Distance away from TANAB2 launch location in meters
    distance_Guelph_SW = Guelph_SW_data[:, 0]

    # Elevation above sea level in meters for each data point away from TANAB2 launch
        location
    elevation_Guelph_SW = Guelph_SW_data[:, 3]
```

```
        # Returns coefficients for the polyfit equation between the distance away from the
            TANAB2
        #  launch location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_SW, elevation_Guelph_SW, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
            TANAB2
        #  Launch location in the South West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_SW)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial with respect
        #  to the elevation data derived from the ALOS DSM file
        ground_elev_AGL = elevation_Guelph_SW - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the West direction
    # Between WSW (247.5 deg) and WNW (292.5 deg)
    elif heading_int > 247.5 and heading_int < 292.5:
        # Elevation data for 10 km due West of the Reek Walk launch location
        Guelph_W_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                            'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_W.csv'

        # Load data from file
        Guelph_W_data = numpy.genfromtxt(Guelph_W_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Guelph_W = Guelph_W_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2 launch
            location
        elevation_Guelph_W = Guelph_W_data[:, 3]

        # Returns coefficients for the polyfit equation between the distance away from the
            TANAB2
        #  launch location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_W, elevation_Guelph_W, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
            TANAB2
        #  Launch location in the West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_W)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial with respect
        #  to the elevation data derived from the ALOS DSM file
        ground_elev_AGL = elevation_Guelph_W - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

    # For the North West direction
```

```
    # Between WNW (292.5 deg) and NNW (337.5 deg)
    elif heading_int > 292.5 and heading_int < 337.5:
        # Elevation data for 10 km due North West of the Reek Walk launch location
        Guelph_NW_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                             'Elevation_Data/Cardinal_Direction_Profiles/ALOS_DSM_NW.csv'

        # Load data from file
        Guelph_NW_data = numpy.genfromtxt(Guelph_NW_filename, delimiter=',')

        # Distance from TANAB2 launch location in meters
        distance_Guelph_NW = Guelph_NW_data[:, 0]

        # Elevation above sea level in meters for each data point away from TANAB2 launch
            location
        elevation_Guelph_NW = Guelph_NW_data[:, 3]

        # Returns coefficients for the polyfit equation between the distance away from the
            TANAB2
        #  launch location and the corresponding elevation above sea level in meters
        Land_Poly_Coeff = numpy.polyfit(distance_Guelph_NW, elevation_Guelph_NW, poly_deg)

        # Evaluate the polynomial at specific values as given by the distance away from the
            TANAB2
        #  Launch location in the North West direction
        ground_elev_ASL_fitted = numpy.polyval(Land_Poly_Coeff, distance_Guelph_NW)

        # Detrend the resulting land surface elevations calculated from the derived
            polynomial with respect
        #  to the elevation data derived from the ALOS DSM file
        ground_elev_AGL = elevation_Guelph_NW - ground_elev_ASL_fitted

        return ground_elev_ASL_fitted, ground_elev_AGL

# Note: Using the Numba library and the @jit (Just In Time compiler), these functions are
    sped up with parallel
#  processing as this function is executed in another compiler after the code is transformed
     to machine code
#
# Depending on the application of this software, this library supports CUDA/GPU processing
    within Python
#
# This library is continually being updated and future versions should have increased
    functionality with
#  respect to parallel processing and GPU/CUDA processing from a Python script

# Calculate Pixel distance to assign emissivity value. For selected pixels
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with
#  the Python code
# This compiler reduced the run time of the code by 90%
# The following formulas are based off of:
#  https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on
```

```
    -latitude-longitude
@jit(nopython=True, parallel=True)
def HaversinePixelCalc(emis_lat_Jul, lat2_pixel, emis_lon_Jul, lon2_pixel, Radius_Earth,
    haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_pixel)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_pixel)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_pixel))
            \
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d

# Calculate Pixel distance to assign emissivity value. For top left pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with
#  the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_left(emis_lat_Jul, lat2_top_left, emis_lon_Jul, lon2_top_left,
    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_top_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_top_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_top_left))\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d
```

```
# Calculate pixel distance to assign emissivity value. For top center pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with
#  the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_center(emis_lat_Jul, lat2_top, emis_lon_Jul, lon2_top,
    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_top)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_top)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_top))\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c

    return haversine_d

# Calculate pixel distance to assign emissivity value. For top right pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with
#  the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_top_right(emis_lat_Jul, lat2_top_right, emis_lon_Jul, lon2_top_right,
     Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_top_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_top_right)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(radians(lat2_top_right))\
                    * math.cos(radians(emis_lat_Jul[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d
```

```
# Calculate pixel distance to assign emissivity value. For center left pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with
#  the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center_left(emis_lat_Jul, lat2_center_left, emis_lon_Jul,
    lon2_center_left, Radius_Earth,
                                   haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_center_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_center_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_center_left))\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value. For center pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with the
#  Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center(emis_lat_Jul, lat2_center, emis_lon_Jul, lon2_center,
    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_center)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_center)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_center)
            )\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))
```

```
        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value. For center right pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with the
#  Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_center_right(emis_lat_Jul, lat2_center_right, emis_lon_Jul,
    lon2_center_right, Radius_Earth,
                                    haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_center_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_center_right)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_center_right))\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value. For bottom left pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with the
#  Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom_left(emis_lat_Jul, lat2_bottom_left, emis_lon_Jul,
    lon2_bottom_left, Radius_Earth,
                                   haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_bottom_left)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_bottom_left)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_bottom_left))\
```

```
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value. For bottom center pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel with the
#  Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom(emis_lat_Jul, lat2_bottom, emis_lon_Jul, lon2_bottom,
    Radius_Earth, haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_bottom)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_bottom)

        # Separate parts of the haversine formula into different variables for calculation
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(lat2_bottom)
            )\
                    * math.cos(math.radians(emis_lat_Jul[k])) * math.sin(haversine_dlon /
                        2) ** 2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Calculate pixel distance to assign emissivity value. For bottom right pixel
# Use the JIT compiler to translate Python/numpy code into machine code that is executed in
    parallel
#  with the Python code
# This compiler reduced the run time of the code by 90%
@jit(nopython=True, parallel=True)
def HaversinePixelCalc_bottom_right(emis_lat_Jul, lat2_bottom_right, emis_lon_Jul,
    lon2_bottom_right, Radius_Earth,
                                    haversine_d):
    # Calculate the haversine distance between each coordinate pair
    for k in range(0, len(emis_lat_Jul)):
        # Calculate the difference between the two latitude locations
        haversine_dlat = math.radians(emis_lat_Jul[k] - lat2_bottom_right)
        # Calculate the difference between the two longitude locations
        haversine_dlon = math.radians(emis_lon_Jul[k] - lon2_bottom_right)

        # Separate parts of the haversine formula into different variables for calculation
```

```
            simplicity
        haversine_a = math.sin(haversine_dlat / 2) ** 2 + math.cos(math.radians(
            lat2_bottom_right))\
                    * math.cos(radians(emis_lat_Jul[k])) * math.sin(haversine_dlon / 2) **
                        2
        haversine_c = 2 * math.atan2(math.sqrt(haversine_a), math.sqrt(1 - haversine_a))

        # Solve for the geographic distance between the two coordinate pairs
        haversine_d[k] = Radius_Earth * haversine_c
    return haversine_d

# Save picture data to master file (master file saves to text file with all image data from
    the 'RawImages' folder
@jit(nopython=True, parallel=True)
def SaveMasterMatrix(x_pixel_range, v_pixel_top, y_pixel_range, image_matrix,
    all_pixel_data_multi_image,
                     filename_image, filenames_total):
    for i in range(0, x_pixel_range):
        for j in range(v_pixel_top, y_pixel_range):
            # If a real value exists with latitude/longitude etc, save to master matrix
            if numpy.isnan(image_matrix[i][j][5]) == False:
                for k in range(0, len(all_pixel_data_multi_image)):
                    if numpy.isnan(all_pixel_data_multi_image[k][0]) == True:
                        # Save name of file to master array
                        filenames_total[k][0] = filename_image[i][j][0]
                        # Save year image was taken to master array
                        all_pixel_data_multi_image[k][0] = image_matrix[i][j][0]
                        # Save month image was taken to master array
                        all_pixel_data_multi_image[k][1] = image_matrix[i][j][1]
                        # Save day image was taken to master array
                        all_pixel_data_multi_image[k][2] = image_matrix[i][j][2]
                        # Save hour image was taken to master array
                        all_pixel_data_multi_image[k][3] = image_matrix[i][j][3]
                        # Save minute image was taken to master array
                        all_pixel_data_multi_image[k][4] = image_matrix[i][j][4]
                        # Save calculated geographic latitude to array
                        all_pixel_data_multi_image[k][5] = image_matrix[i][j][5]
                        # Save calculated geographic longitude to array
                        all_pixel_data_multi_image[k][6] = image_matrix[i][j][6]
                        # Save the horizontal pixel value of the image where ST was
                            calculated
                        all_pixel_data_multi_image[k][7] = image_matrix[i][j][7]
                        # Save the vertical pixel value of the image where ST was calculated
                        all_pixel_data_multi_image[k][8] = image_matrix[i][j][8]
                        # Save the ST in kelvin considering the MODIS emissivity of the land
                             surface
                        all_pixel_data_multi_image[k][9] = image_matrix[i][j][9]
                        # Save the ST in degC considering the MODIS emissivity of the land
                            surface
                        all_pixel_data_multi_image[k][10] = image_matrix[i][j][10]
                        break
    return all_pixel_data_multi_image
```

```
# Time the following process
start = time.time()

# Directory where RAW DJI_XXX.jpg Images are located
directory = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/Elevation_Data/
    RawImages'

# Return the Number of images in RawImages Directory
numFiles = sum([len(files) for r, d, files in os.walk(directory)])

# Loop through each thermal image in the RawImages directory
for file in os.listdir(directory):
    # Read the file name that would be shown in the Linux Terminal
    filename = os.fsdecode(file)
    print('The Image being processed now is: '+str(filename))

    # Extract variables used for georeferencing calculations
    # Extract GPS Latitude from image via the Linux Terminal with ExifTool (ExifTool
        converts to decimal degrees)
    Latitude = subprocess.Popen(["exiftool -b -GPSLatitude " + directory + "/" + filename],
        shell=True,
                                stdout=subprocess.PIPE).communicate()[0]
    # Decode Latitude to string, a readable format
    Latitude = Latitude.decode("utf-8")
    # Convert from string to float
    Latitude = float(Latitude)

    # Extract GPS Longitude from image via the Linux Terminal with ExifTool (ExifTool
        converts to decimal degrees)
    Longitude = subprocess.Popen(["exiftool -b -GPSLongitude " + directory + "/" + filename
        ], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]
    # Decode Longitude to string, a readable format
    Longitude = Longitude.decode("utf-8")
    # Convert from string to float
    Longitude = float(Longitude)

    # Extract camera Gimbal Roll Degree from image via the Linux terminal with ExifTool
    gRollDeg = subprocess.Popen(["exiftool -b -GimbalRollDegree " + directory + "/" +
        filename], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]
    # Decode Gimbal Roll Degree to string, a readable format
    gRollDeg = gRollDeg.decode("utf-8")
    # Convert from string to float
    gRollDeg = float(gRollDeg)

    # Extract camera Gimbal Yaw Degree from image via the Linux terminal with ExifTool
    gYawDeg = subprocess.Popen(["exiftool -b -GimbalYawDegree " + directory + "/" + filename
        ], shell=True,
                               stdout=subprocess.PIPE).communicate()[0]
    # Decode Gimbal Yaw Degree to string
```

```
gYawDeg = gYawDeg.decode("utf-8")
# Convert from string to float
gYawDeg = float(gYawDeg)

# Extract camera Gimbal Pitch Degree from image via the Linux terminal with ExifTool
gPitchDeg = subprocess.Popen(["exiftool -b -GimbalPitchDegree " + directory + "/" +
    filename], shell=True,
                             stdout=subprocess.PIPE).communicate()[0]
# Decode Gimbal Pitch Degree to string
gPitchDeg = gPitchDeg.decode("utf-8")
# Convert from string to float
gPitchDeg = float(gPitchDeg)

# Extract Flight (Gondola) Roll Degree from image as recorded by N3 via the Linux
    terminal with ExifTool
fRollDeg = subprocess.Popen(["exiftool -b -FlightRollDegree " + directory + "/" +
    filename], shell=True,
                            stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Roll Degree to string
fRollDeg = fRollDeg.decode("utf-8")
# Convert from string to float
fRollDeg = float(fRollDeg)

# Extract Flight (Gondola) Yaw Degree from image as recorded by N3 via the Linux
    Terminal with ExifTool
fYawDeg = subprocess.Popen(["exiftool -b -FlightYawDegree " + directory + "/" + filename
    ], shell=True,
                           stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Yaw Degree to string
fYawDeg = fYawDeg.decode("utf-8")
# Convert from string to float
fYawDeg = float(fYawDeg)

# Extract Flight (Gondola) Pitch Degree from image as recorded by N3 via the Linux
    terminal with ExifTool
fPitchDeg = subprocess.Popen(["exiftool -b -FlightPitchDegree " + directory + "/" +
    filename], shell=True,
                             stdout=subprocess.PIPE).communicate()[0]
# Decode Flight Pitch Degree to string
fPitchDeg = fPitchDeg.decode("utf-8")
# Convert from string to float
fPitchDeg = float(fPitchDeg)

#
    ##################################################################################################################

# Filtering Parameters for GPS georeferencing:
# If Gondola Roll > +/- 45 degrees (since camera is self stabilized, roll should be
    minimal)
# If Gondola tilt (fPitchDeg) is > +45 degrees or < -135 degrees (as per mechanical
    range of
#  Zenmuse XT: https://www.dji.com/zenmuse-xt/info) This can affect the self
```

```
    stabilization of the camera
# If latitude or longitude = 0 degrees, Longitude > 180 degrees or
#  Longitude < 180 degrees, Latitude > 90 degrees or Latitude < 90 degrees
# If Camera Gimbal pitch is >= to 0 degrees (center of the image), GPS georeferencing
    will not work
#  as the camera line of sight will extend to the sky
if fRollDeg > 45 or fRollDeg < -45 or fPitchDeg > 45 or fPitchDeg < -135 or gPitchDeg >=
     0:
    continue
elif Latitude <= 0 or Latitude > 90 or Latitude < -90:
    continue
elif Longitude == 0 or Longitude > 180 or Longitude < -180:
    continue

# If the gimbal pitch plus half of the vertical field of view is <= -76 deg, then skip
    the image
# If this was not included, the bottom of the image could theoretically be positioned
    behind the camera
#  which would complicate calculations
if gPitchDeg <= -76:
    continue

# If gimbal pitch is greater than 2 deg, skip image
if gPitchDeg > -2:
    continue

#
    ##########################################################################################################

# Parameters for temperature calculation
# For all Planck Constants below, reference Martiny et al. 1996, "In Situ Calibration
    for Quantitative Infrared
#  Thermography": http://qirt.gel.ulaval.ca/archives/qirt1996/papers/001.pdf
# Also reference FLIR Systems, Installation manual: FLIR A3XX and FLIR A6XX series,
#  2010: http://91.143.108.245/Downloads/Flir/Dokumentation/T559498$a461_Manual.pdf

# Get Planck R1 constant from image metadata with ExifTool via Linux terminal
R1 = subprocess.Popen(["exiftool -b -PlanckR1 " + directory + "/" + filename], shell=
    True,
                      stdout=subprocess.PIPE).communicate()[0]
# Decode Planck R1 constant to string
R1 = R1.decode("utf-8")
# Convert from string to float
R1 = float(R1)

# Get Planck R2 constant from image metadata with ExifTool via Linux terminal
R2 = subprocess.Popen(["exiftool -b -PlanckR2 " + directory + "/" + filename], shell=
    True,
                      stdout=subprocess.PIPE).communicate()[0]
# Decode Planck R2 constant to string
R2 = R2.decode("utf-8")
# Convert from string to float
```

```
R2 = float(R2)

# Get Planck B constant from image metadata with ExifTool via Linux terminal
B = subprocess.Popen(["exiftool -b -PlanckB " + directory + "/" + filename], shell=True,
                     stdout=subprocess.PIPE).communicate()[0]
# Decode Planck B constant to string
B = B.decode("utf-8")
# Convert from string to float
B = float(B)

# Get Planck O constant from image metadata with ExifTool via Linux terminal
planck_O = subprocess.Popen(["exiftool -b -PlanckO " + directory + "/" + filename],
    shell=True,
                            stdout=subprocess.PIPE).communicate()[0]
# Decode Planck O constant to string
planck_O = planck_O.decode("utf-8")
# Convert from string to float
planck_O = float(planck_O)

# Get Planck F constant from image metadata with ExifTool via Linux terminal
F = subprocess.Popen(["exiftool -b -PlanckF " + directory + "/" + filename], shell=True,
                     stdout=subprocess.PIPE).communicate()[0]
# Decode Planck F constant to string
F = F.decode("utf-8")
# Convert from string to float
F = float(F)

#
    ##########################################################################################################################

# The next few lines is for TriSonica Altitude calculations to derive TANAB2 altitude
    above ground level
# Need to call in the date from each picture and convert to day of year in seconds (doy)
     (add the hours, minutes,
#  and the seconds)
# Next, find the closest doy in TriSonica doy, return the index with the closest value
    to identify the
#  altitude of balloon

# Get date and time when pictures were taken
# If the following variables do not exist as a local variable, then initialize them
if 'Year' not in locals():
    Year = numpy.empty(numFiles, dtype=object)
    Month = numpy.empty(numFiles, dtype=object)
    Days = numpy.empty(numFiles, dtype=object)
    Hour = numpy.empty(numFiles, dtype=object)
    Minutes = numpy.empty(numFiles, dtype=object)

# Get date and time from images via the Linux terminal with ExifTool
dates = subprocess.Popen(["exiftool -b -DateTimeOriginal " + directory + "/" + filename
    ], shell=True,
                         stdout=subprocess.PIPE).communicate()[0]
```

```
# convert dates to string
dates = str(dates)
# Slice string to only include date and time
print('The Image Date and Time is: '+str(dates[2:21]))
# Date & time as YYYY:MM:DD HH:MM:SS
dates = dates[2:21]
# Convert date format to date time from string
dates = datetime.datetime.strptime(dates, "%Y:%m:%d %H:%M:%S")

# Separate image date and time into variables and change data type from datetime to
    string
yr = str(dates.year)
mnth = str(dates.month)
day = str(dates.day)
hr = str(dates.hour)
minute = str(dates.minute)

#
    ##########################################################################################################

# Assign the Base Altitude (elevation above sea level) for the TANAB2 launch location
    for the Urban Field Campaign
# Estimated elevations from Google Earth,
BaseLat = 43.532381
BaseLon = -80.225408
BaseAltitude = 334 # meters above sea level
#
    ##########################################################################################################

# Call in averaged data extracted from TriSonica
# This file includes data from the July 28, 2018 and August 13, 2018 TANAB2 Launches
trisonica_avg_fileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
                         'TriSonica/TriSonica_Averaged_Guelph_Urban_Altitudes.txt'

# Call in TriSonica averaged data
trisonica_avg = numpy.genfromtxt(trisonica_avg_fileName, usecols=[6,10])
# Call in second averaged data column
# TriSonica day of year (since January 1, 2018) in seconds
trisonica_soy = trisonica_avg[:,0]

# Call in the TriSonica derived altitude from pressures. The altitude is relative to the
    land surface
trisonica_altitude = trisonica_avg[:,1]

# Day/hour/minute in July/August when pictures were taken
month_picture = dates.month
day_picture = dates.day
hour_picture = dates.hour
minute_picture = dates.minute
seconds_picture = dates.second

# Initialize doy in seconds for pictures
```

```
doy_seconds_picture = 0

# Convert day/hour/minute into doy seconds based on the day the picture was recorded
# Where 209 DOY = July 28, 2018 and 225 DOY - August 13, 2018
if day_picture == 28:
    doy_seconds_picture = (209*24*60*60)+(hour_picture*60*60)+(minute_picture*60)+
        seconds_picture

elif day_picture == 13:
    doy_seconds_picture = (225*24*60*60)+(hour_picture*60*60)+(minute_picture*60)+
        seconds_picture

else:
    print('More Dates need to be included above')

# Initialize delta doy in seconds (soy) variable (difference between image capture time
    and TriSonica data)
delta_soy = numpy.zeros(len(trisonica_soy))

# Match doy second indices from second averaged file & each individual image
# Loop through averaged doy indices from TriSonica data
for i in range(0, len(trisonica_soy)):
    # Calculate difference between doy times (each individual image and each second
        averaged TriSonica index)
    delta_soy[i] = abs(doy_seconds_picture - trisonica_soy[i])

    # When at the last value of the TriSonica averaged file, find index of minimum value
        and write corresponding
    #  altitude value to the Altitude variable
    if i == (len(trisonica_soy) - 1):
        Altitude_AGL = trisonica_altitude[numpy.argmin(delta_soy)]

print('The Altitude above the TANAB launch location is: '+str(Altitude_AGL)+' [m]')

#
    ##########################################################################################################

# Set up variables that are used within the georeferencing calculations below
# Yaw, Roll and Pitch frame of references correspond to:
#  http://blog-gh4-france.over-blog.com/2015/12/test-du-dji-ronin-m-sur-le-gh4.html
# Yaw of gimbal is calibrated to the True North, therefore it is not necessary to add
    the flight yaw angle
Yaw = gYawDeg

# Assume the gimbal roll is zero (camera is self-stabilized)
Roll = fRollDeg

# Pitch and Roll angles are independent of each other, ie if one changes (ex. the flight
    parameter),
#  it will directly affect the gimbal but it will not be accounted for by the gimbal
    pitch/roll variable
# Positive upward from horizontal; usually negative for lines of sight below horizontal
```

```
Pitch = gPitchDeg

# Field of View angles in degrees for the 19 mm Zenmuse XT thermal camera
#  (Specifications: https://www.dji.com/ca/zenmuse-xt/specs)
# Vertical Field of View [degree]
FoVV = 26
# Horizontal Field of View [degree]
FoVH = 32

# When the camera lens is exactly perpendicular to the ground (pointed towards the
    horizon),
#  the resulting pitch angle for the center of an image is 0 degrees
# If the camera is tilted towards the ground, the pitch angle is negative, if camera
    tiled upwards,
#  the pitch angle is positive

# References for mechanical rotational limits of the Zenmuse XT: https://www.dji.com/
    zenmuse-xt/info
# Find GPS coordinates for middle (tilt_center), top middle (tilt_top) and
#  bottom middle (tilt_bottom) of each image
# These three values are all related to the gimbal pitch angle from the image metadata
    and the
#  physical vertical field of view for the 19mm lens camera
# Tilt angles [degree]
tilt_center = Pitch
tilt_bottom = Pitch-(FoVV/2)
tilt_top = Pitch+(FoVV/2)

# Need to check if the sky will be in the image. Images with a top tilt angle
#  above 0 degrees will have their tilt angle adjusted to include pixels below an
    assumed
#  pixel row which is a direct function of an assumed tilt angle
# The center tilt angle does not need to be considered as images with a gimbal pitch
    angle
#  >= 0 degrees were already skipped over in the code near the top of the script

# if top of image has a tilt angle >= -1 degrees then, assign an assumed tilt angle to
    omit pixels that may
#  contain the sky (i.e. above the horizon/horizontal)
if tilt_top >= -1:
    print('The tilt angle for the top of the image is: ' + str(tilt_top))
    # Change tilt angle so it is equal to an arbitrary tilt angle. I assumed -1 degrees
        (can be changed)
    tilt_top = -1

print('The top tilt angle (deg) for the image is: ' + str(tilt_top))
print('The center tilt angle (deg) for the image is: ' + str(tilt_center))
print('The bottom tilt angle (deg) for the image is: ' + str(tilt_bottom))

#
    ##########################################################################################################################
```

```
# Used this section of code to implement a sensitivity analysis. Can be revisited if
    required.

# Apply an offset to each Pitch angle used during the sensitivity analysis as of Dec.
    5/2018.
# December 17, 2018: After completing a few tests, the developed method does not need
    the tilt angle adjusted,
#  therefore set the theta offset as 0.
thetaOffset = 0

theta_top_degrees = tilt_top-thetaOffset
theta_center_degrees = tilt_center-thetaOffset
theta_bottom_degrees = tilt_bottom-thetaOffset

#
    ##########################################################################################################################

# Convert theta angles to radians. All trigonometric functions in Python assume that
    angles are in radians
theta_top_rads = math.radians(theta_top_degrees)
theta_center_rads = math.radians(theta_center_degrees)
theta_bottom_rads = math.radians(theta_bottom_degrees)

# Note: If heading is 0 deg or 360 deg = north exactly, if 90 deg =  east, 180 deg =
    south, 270 deg = west
# Set Heading variable for georeferencing calculations
heading = Yaw

# Adjust heading if less than 0 degrees, add 360 degrees, so the angles will always be
    positive
if heading < 0:
    heading = heading + 360

# Convert Latitude/Longitude/Yaw to Rads for georeferencing calculations
Yaw_rads = math.radians(heading)
Latitude_rads = math.radians(Latitude)
Longitude_rads = math.radians(Longitude)

# Call LandSlopeEquations Function
# Function fits a polynomial relating the elevation above sea level for the land in the
    8 cardinal
#  directions at the Reek Walk TANAB2 launch site to distance away from the launch site
#  (up to 10km away from the origin of each launch site)
ground_elev_ASL_fitted, ground_elev_AGL = LandSlopeEquations(BaseAltitude,heading)

# Calculate latitude/longitude of top/virtual top of image
# Find coefficients for line of sight from camera that intersects with the ground
line_of_sight = numpy.zeros(4)
# Find slope of the line associated with line of sight which is negative because
    theta_top is negative
line_of_sight[2] = math.tan(theta_top_rads)
# Find altitude of the TANAB2 from the ground. this is the y-intercept of the line of
```

```
    sight with respect to
#  TANAB2 launch origin located at ground level
line_of_sight[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000, the number of indices to
    use is equal to the
#  length of ground_elev_AGL (as calculated in LandSlopeEquations function) (given
    information,
#  the numerical difference between each value is calculated by the function
# x_test represents distance in meters away from the TANAB2 launch site
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with respect to the
    detrended
#  land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended surface elevation
    with respect to distance
#  away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test,ground_elev_AGL,3)

# Find the roots and select the smallest positive value to be the horizontal distance
    away from the
#  TANAB2 launch location
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Top = numpy.roots(coefficients_of_intersection)
print('The roots are: ' + str(Roots_Top))
print('The altitude of the balloon above grade level is: '+str(line_of_sight[3]))
print('The Tilt angle is: '+str(math.degrees(theta_top_degrees)))

# Create an array for the roots that are real numbers (not complex)
real_roots_top = numpy.empty(3)
real_roots_top[:] = numpy.nan

# Check for roots that are not complex numbers and write real roots to array
for i in range(0, len(Roots_Top)):
    if numpy.iscomplex(Roots_Top[i]) == False:
        real_roots_top[i] = Roots_Top[i]

print('The Real Roots for the Top Center are: '+str(real_roots_top))

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Top)):
    if real_roots_top[non_real_root] < 0 or numpy.isnan(real_roots_top[non_real_root])
        == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Top):
```

```
        continue

    # Choose the root that is the smallest real positive solution to be the distance from
        the TANAB2 launch location
    #  to the top of the projected image on the land surface
    d_top = min(i for i in real_roots_top if i > 0)
    # Convert d_top to km
    d_top_km = d_top/1000

    # Volumetric Mean Radius of earth in km: https://nssdc.gsfc.nasa.gov/planetary/factsheet
        /earthfact.html
    Radius_Earth = 6378.1

    # If the distance is less than 5km, then calculate coordinates. This is included as the
        urban environment
    #  likely impacts the camera line of sight. The 5 km value is assumed and can be changed
        .
    if d_top_km < 5:
        # Get the latitude/longitude for the top, center, and bottom of each image via the
        #  variation of the Haversine Formula
        # Formulas for latitude/longitude are from: https://www.movable-type.co.uk/scripts/
            latlong.html,
        #  as of August 17/2018, I verified that these formulas are correct
        # Calculate lat2/lon2 for top of image

        lat2_top = asin(sin(Latitude_rads)*cos(d_top_km/Radius_Earth)+cos(Latitude_rads)*sin
            (d_top_km/Radius_Earth)
                        *cos(Yaw_rads))

        lon2_top = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_top_km/Radius_Earth)*cos(
            Latitude_rads),
             cos(d_top_km/Radius_Earth)-sin(Latitude_rads)*sin(lat2_top))

        # Convert coordinates to decimal degrees
        lat2_top = math.degrees(lat2_top)
        lon2_top = math.degrees(lon2_top)

    #
        ##########################################################################################################################

    # Calculate latitude/longitude for the center of the image
    # Find coefficients for line of sight from camera that intersects with the ground
    line_of_sight = numpy.zeros(4)
    line_of_sight[2] = math.tan(theta_center_rads)
    line_of_sight[3] = Altitude_AGL

    # Create equations and detrend the data
    # Create a numeric sequence starting at 0, ending at 30,000, the number of indices to
        use is equal to the
    #  length of ground_elev_AGL (as calculated in LandSlopeEquations function)
    #  (given information, the numerical difference between each value is calculated by the
        function
```

```
# x_test represents distance in meters away from the TANAB2 launch site
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with respect to the
    detrended
#  land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended surface elevation
    with respect
#  to distance away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test, ground_elev_AGL, 3)

# Find the roots and select the smallest positive value to be the horizontal distance
    away from the
#  TANAB2 launch location
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Center = numpy.roots(coefficients_of_intersection)
print('The roots are: ' + str(Roots_Center))
print('The altitude of the balloon is: ' + str(line_of_sight[3]))
print('The Tilt angle is: ' + str(math.degrees(theta_center_rads)))

# Create an array for the roots that are real numbers (not complex)
real_roots_center = numpy.empty(3)
real_roots_center[:] = numpy.nan
for i in range(0, len(Roots_Center)):
    if numpy.iscomplex(Roots_Center[i]) == False:
        real_roots_center[i] = Roots_Center[i]

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Center)):
    if real_roots_center[non_real_root] <= 0 or numpy.isnan(real_roots_center[
        non_real_root]) == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Center):
    continue

# Choose the root that is the smallest real positive solution to be the distance from
    the TANAB2 launch location
#  to the center of the projected image on the land surface
d_center = min(i for i in real_roots_center if i > 0)

# Convert d_center to km
d_center_km = d_center / 1000

# If the center horizontal distance is less than 5km, calculate the corresponding pixel
    coordinates.
#  This is included as the urban environment likely impacts the camera line of sight.
#  The 5 km value is assumed and can be changed.
```

```
if d_center_km < 5:
    # Formulas for latitude/longitude are from: https://www.movable-type.co.uk/scripts/
        latlong.html,
    #  as of Aug.17/2018, I verified that these formulas are correct
    # Calculate lat2/lon2 for the center of the image
    lat2_center = asin(sin(Latitude_rads)*cos(d_center_km/Radius_Earth)+cos(
        Latitude_rads)
                       *sin(d_center_km/Radius_Earth)*cos(Yaw_rads))

    lon2_center = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_center_km/Radius_Earth)*cos
        (Latitude_rads),
                                         cos(d_center_km/Radius_Earth)-sin(Latitude_rads)*
                                             sin(lat2_center))

    # Convert back to decimal degrees
    lat2_center = math.degrees(lat2_center)
    lon2_center = math.degrees(lon2_center)

#
    ##########################################################################################################################

# Calculate latitude/longitude for the center of the image
# Find coefficients for line of sight from camera that intersects with the ground
line_of_sight = numpy.zeros(4)
line_of_sight[2] = math.tan(theta_bottom_rads)
line_of_sight[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000, the number of indices to
    use is equal
#  to the length of ground_elev_AGL (as calculated in LandSlopeEquations function)
#  (given information, the numerical difference between each value is calculated by the
    function
# x_test represents distance in meters away from the TANAB2 launch site along the
x_test = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with respect to the
#  detrended land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended surface elevation
    with respect
#  to distance away from the TANAB2 launch site
land_elevation_equation_AGL = numpy.polyfit(x_test, ground_elev_AGL, 3)

# Find the roots and select the smallest positive value to be the horizontal distance
    away from the
#  TANAB2 launch location
coefficients_of_intersection = land_elevation_equation_AGL - line_of_sight
Roots_Bottom = numpy.roots(coefficients_of_intersection)

print('The roots are: ' + str(Roots_Bottom))
print('The altitude of the balloon is: ' + str(line_of_sight[3]))
print('The Tilt angle is: ' + str(math.degrees(theta_bottom_rads)))
```

```
# Create an array for the roots that are real numbers (not complex)
real_roots_bottom = numpy.empty(3)
real_roots_bottom[:] = numpy.nan
for i in range(0, len(Roots_Bottom)):
    if numpy.iscomplex(Roots_Bottom[i]) == False:
        real_roots_bottom[i] = Roots_Bottom[i]

# Initialize a variable that counts the number of non real roots
root_counter = 0

# Identify the number of non real roots
for non_real_root in range(0, len(Roots_Bottom)):
    if real_roots_bottom[non_real_root] < 0 or numpy.isnan(real_roots_bottom[
        non_real_root]) == True:
        root_counter += 1

# If no real roots exist, then continue to the next image
if root_counter == len(Roots_Bottom):
    continue

# Choose the root that is the smallest real positive solution to be the distance from
    the TANAB2 launch location to
#  the bottom of the projected image on the land surface
d_bottom = min(i for i in real_roots_bottom if i > 0)
# Convert d_bottom to km
d_bottom_km = d_bottom / 1000

# If the horizontal distance for the bottom of the image is greater than 5 km,
#  skip the image as the line of sight likely intersects a structure
if d_bottom_km > 5:
    continue

# Formulas for latitude/longitude are from: https://www.movable-type.co.uk/scripts/
    latlong.html,
#  as of Aug.17/2018, I verified that these formulas are correct
# Calculate lat2/lon2 for the bottom of the image
lat2_bottom = asin(sin(Latitude_rads)*cos(d_bottom_km/Radius_Earth)+cos(Latitude_rads)
                   *sin(d_bottom_km/Radius_Earth)*cos(Yaw_rads))

lon2_bottom = Longitude_rads + atan2(sin(Yaw_rads)*sin(d_bottom_km/Radius_Earth)*cos(
    Latitude_rads),
                                     cos(d_bottom_km/Radius_Earth)-sin(Latitude_rads)*sin(
                                         lat2_bottom))

# Convert back to decimal degrees
lat2_bottom = math.degrees(lat2_bottom)
lon2_bottom = math.degrees(lon2_bottom)

# Print Results
print('The Altitude of the balloon with respect to grade level is: '+str(Altitude_AGL)+'
    \n')
```

```
print('The Origin lat is: '+str(Latitude))
print('The Origin lon is: '+str(Longitude)+'\n')
if 'lat2_top' in locals():
    print('The lat_top is: ' + str(lat2_top))
    print('The lon_top is: ' + str(lon2_top) + '\n')
    print('The d_top is: ' + str(d_top))
if 'lat2_center' in locals():
    print('The lat2_center is: ' + str(lat2_center))
    print('The lon2_center is: ' + str(lon2_center) + '\n')
    print('The d_center is: ' + str(d_center))
if 'lat2_bottom' in locals():
    print('The lat2_bottom is: ' + str(lat2_bottom))
    print('The lon2_bottom is: ' + str(lon2_bottom) + '\n')
    print('The d_bottom is: '+str(d_bottom))

#
    ##########################################################################################################################

# Calculate GPS coordinates for pixels on the edge/corners of the image
# Find the top right and top left latitude/longitude for each image
# Find the geographic distance in km for both the top right and left of each image
d_geographic_top_km = d_top_km/cos(math.radians(FoVH / 2))

# For pixels on the left edge of the image
Yaw_left_rads = math.radians(heading - (FoVH / 2))

# Ensure this angle is strictly positive
if Yaw_left_rads < 0:
    Yaw_left_rads = Yaw_left_rads + 2 * numpy.pi

# For pixels on the right edge of the image
Yaw_right_rads = math.radians(heading + (FoVH / 2))

# Ensure this angle is strictly less than 360 degrees
if Yaw_right_rads > 2 * numpy.pi:
    Yaw_right_rads = Yaw_right_rads - 2 * numpy.pi

# Find the latitude/longitude for the top left of the image
lat2_top_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_top_km / Radius_Earth) +
    cos(Latitude_rads)
                          * sin(d_geographic_top_km / Radius_Earth) * cos(Yaw_left_rads)
                              )

lon2_top_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads) * sin(d_geographic_top_km
    / Radius_Earth)
                                            * cos(Latitude_rads), cos(
                                                d_geographic_top_km / Radius_Earth)
                                            - sin(Latitude_rads) * sin(
                                                lat2_top_left_rads))

# Convert back to decimal degrees
lat2_top_left = math.degrees(lat2_top_left_rads)
```

```
lon2_top_left = math.degrees(lon2_top_left_rads)

# Find the latitude/longitude for the top right of the image
lat2_top_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_top_km / Radius_Earth)
    + cos(Latitude_rads)
                           * sin(d_geographic_top_km / Radius_Earth) * cos(
    Yaw_right_rads))

lon2_top_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads) * sin(
    d_geographic_top_km / Radius_Earth)
                                             * cos(Latitude_rads), cos(
                                                 d_geographic_top_km / Radius_Earth)
                                             - sin(Latitude_rads) * sin(
                                                 lat2_top_right_rads))

# Convert back to decimal degrees
lat2_top_right = math.degrees(lat2_top_right_rads)
lon2_top_right = math.degrees(lon2_top_right_rads)

#
    ##########################################################################################################################

# Find the geographic distance in km for both the center right and left of each image
d_geographic_center_km = d_center_km / cos(math.radians(FoVH / 2))

# Find the latitude/longitude for the center left of the image
lat2_center_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_center_km /
    Radius_Earth) + cos(Latitude_rads)
                             * sin(d_geographic_center_km / Radius_Earth) * cos(
                                 Yaw_left_rads))

lon2_center_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads) * sin(
    d_geographic_center_km / Radius_Earth)
                                               * cos(Latitude_rads), cos(
                                                   d_geographic_center_km / Radius_Earth
                                                   )
                                               - sin(Latitude_rads) * sin(
                                                   lat2_center_left_rads))

# Convert back to decimal degrees
lat2_center_left = math.degrees(lat2_center_left_rads)
lon2_center_left = math.degrees(lon2_center_left_rads)

# Find the latitude/longitude for the center right of the image
lat2_center_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_center_km /
    Radius_Earth) + cos(Latitude_rads)
                              * sin(d_geographic_center_km / Radius_Earth) * cos(
                                  Yaw_right_rads))

lon2_center_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads) * sin(
    d_geographic_center_km / Radius_Earth)
                                                * cos(Latitude_rads), cos(
```

```
                                                    d_geographic_center_km /
                                                    Radius_Earth)
                                                - sin(Latitude_rads) * sin(
                                                    lat2_center_right_rads))

# Convert back to decimal degrees
lat2_center_right = math.degrees(lat2_center_right_rads)
lon2_center_right = math.degrees(lon2_center_right_rads)

#
    ##########################################################################################################

# Find the geographic distance in km for both the bottom right and left of each image
d_geographic_bottom_km = d_bottom_km / cos(math.radians(FoVH / 2))

# Find the latitude/longitude for the bottom left of the image
lat2_bottom_left_rads = asin(sin(Latitude_rads) * cos(d_geographic_bottom_km /
    Radius_Earth) + cos(Latitude_rads)
                             * sin(d_geographic_bottom_km / Radius_Earth) * cos(
                                 Yaw_left_rads))

lon2_bottom_left_rads = Longitude_rads + atan2(sin(Yaw_left_rads) * sin(
    d_geographic_bottom_km / Radius_Earth)
                                               * cos(Latitude_rads), cos(
                                                   d_geographic_bottom_km / Radius_Earth
                                                   )
                                               - sin(Latitude_rads) * sin(
                                                   lat2_bottom_left_rads))

# Convert back to decimal degrees
lat2_bottom_left = math.degrees(lat2_bottom_left_rads)
lon2_bottom_left = math.degrees(lon2_bottom_left_rads)

# Find the latitude/longitude for the bottom right of the image
lat2_bottom_right_rads = asin(sin(Latitude_rads) * cos(d_geographic_bottom_km /
    Radius_Earth) + cos(Latitude_rads)
                              * sin(d_geographic_bottom_km / Radius_Earth) * cos(
                                  Yaw_right_rads))

lon2_bottom_right_rads = Longitude_rads + atan2(sin(Yaw_right_rads) * sin(
    d_geographic_bottom_km / Radius_Earth)
                                                * cos(Latitude_rads), cos(
                                                    d_geographic_bottom_km /
                                                    Radius_Earth)
                                                - sin(Latitude_rads) * sin(
                                                    lat2_bottom_right_rads))

# Convert back to decimal degrees
lat2_bottom_right = math.degrees(lat2_bottom_right_rads)
lon2_bottom_right = math.degrees(lon2_bottom_right_rads)

print('The lat2_top_left is: ' + str(lat2_top_left))
```

```
print('The lon2_top_left is: ' + str(lon2_top_left) + '\n')
print('The lat2_top_right is: ' + str(lat2_top_right))
print('The lon2_top_right is: ' + str(lon2_top_right) + '\n')

print('The lat2_center_left is: ' + str(lat2_center_left))
print('The lon2_center_left is: ' + str(lon2_center_left) + '\n')
print('The lat2_center_right is: ' + str(lat2_center_right))
print('The lon2_center_right is: ' + str(lon2_center_right) + '\n')

print('The lat2_bottom_left is: ' + str(lat2_bottom_left))
print('The lon2_bottom_left is: ' + str(lon2_bottom_left) + '\n')
print('The lat2_bottom_right is: ' + str(lat2_bottom_right))
print('The lon2_bottom_right is: ' + str(lon2_bottom_right) + '\n')

#
    ##########################################################################################################################

# Maximum pixel width and height of each image based on the FLIR 19 mm lens Zenmuse XT
x_pixel_range = 640
y_pixel_range = 512


# If the tilt angle for the top of the image is greater than zero degrees and an assumed
    top
#  tilt angle was assigned, Calculate the new pixel top row, only if the top tilt angle
    is > or = to -1 degrees
if tilt_top >= -1:
    # Get relation between Trigonometric angles and pixels
    # See created figures in thesis for visual reference
    # The following variables are all in degrees
    # Note: d_top was already calculated by assuming a new top which looked down -1
        degrees below horizon
    gamma = math.degrees(math.atan(d_top/Altitude_AGL))
    beta = 90 - abs(Pitch) + (FoVV/2) - gamma
    kappa = 90-(FoVV/2)
    eta = 90-(FoVV/2)+beta

    y_pixel_top = int(((y_pixel_range/2)*sin(radians(beta)))/(sin(radians(FoVV/2))*sin(
        radians(180-eta))))

    print('The top of the image is located at: '+str(y_pixel_top))

# The value to divide the horizontal and vertical pixel resolution by
# Used to calculate the maximum pixel step for the horizontal direction
delta_x_pixel = 10
delta_y_pixel = 8

# Returns a maximum pixel step of 64 columns/rows based on the 19mm-Zenmuse XT image
#  horizontal and vertical resolution
x_max_step = int(x_pixel_range/delta_x_pixel)
y_max_step = int(y_pixel_range/delta_y_pixel)
```

```
# Initialize matrix for singular image for pixels include: x/y pixel coordinates,
    latitude, longitude, temperature,
#  and set values to Nan

# Note: The amount of data retrieved per image should be less than the full image as a
    result the size of
#  image_matrix could likely be optimized
image_matrix = numpy.zeros((x_pixel_range, y_pixel_range, 11))
image_matrix[:] = numpy.nan

# Create an array for the filename for the corresponding pixel retrieved from each image
# This is required as this script does not process images in chronological order
filename_image = numpy.chararray((x_pixel_range, y_pixel_range,1), itemsize=12)

# Initialize matrix to save data from all pictures (and a separate variable for
    filenames)
#  check if variable exists in local (do only for the first image)
# This section is only run for the 1st image processed
if 'all_pixel_data_multi_image' not in locals():
    all_pixel_data_multi_image = numpy.zeros((x_max_step*y_max_step*numFiles, 11))
    all_pixel_data_multi_image[:] = numpy.nan
    filenames_total = numpy.chararray((x_max_step*y_max_step*numFiles, 1), itemsize=12)

# Create variables to save known coordinates to
# Size will depend on the number of files in the folder (numFiles)
# These variables are to contain the data for the center and edges of each image
# This section of code is only executed for the 1st image processed
if 'file_names_array' not in globals():
    # file names
    file_names_array = numpy.chararray(numFiles*1, itemsize=12)
    file_names_array[:] = b''

    # TANAB2 launch location
    lat_TANAB_array = numpy.zeros((numFiles, 1))
    lon_TANAB_array = numpy.zeros((numFiles, 1))

    # Top left of image
    # Create arrays for the latitude, longitude, and pixel locations for the top left
        corner of the image
    tLeft_lat_array = numpy.zeros((numFiles, 1))
    tLeft_lon_array = numpy.zeros((numFiles, 1))
    tLeft_x_pixel_array = numpy.zeros((numFiles, 1))
    tLeft_y_pixel_array = numpy.zeros((numFiles, 1))

    # Top center of image
    # Create arrays for the latitude, longitude, and pixel locations for the top middle
        of the image
    tCenter_lat_array = numpy.zeros((numFiles, 1))
    tCenter_lon_array = numpy.zeros((numFiles, 1))
    tCenter_x_pixel_array = numpy.zeros((numFiles, 1))
    tCenter_y_pixel_array = numpy.zeros((numFiles, 1))
```

```
# Top right of image
# Create arrays for the latitude, longitude, and pixel locations for the top right
    corner of the image
tRight_lat_array = numpy.zeros((numFiles, 1))
tRight_lon_array = numpy.zeros((numFiles, 1))
tRight_x_pixel_array = numpy.zeros((numFiles, 1))
tRight_y_pixel_array = numpy.zeros((numFiles, 1))

# Center left of image
# Create arrays for the latitude, longitude, and pixel locations for the center left
     edge of the image
cLeft_lat_array = numpy.zeros((numFiles, 1))
cLeft_lon_array = numpy.zeros((numFiles, 1))
cLeft_x_pixel_array = numpy.zeros((numFiles, 1))
cLeft_y_pixel_array = numpy.zeros((numFiles, 1))

# Center of image
# Create arrays for the latitude, longitude, and pixel locations for the middle (
    center) of the image
center_lat_array = numpy.zeros((numFiles, 1))
center_lon_array = numpy.zeros((numFiles, 1))
center_x_pixel_array = numpy.zeros((numFiles, 1))
center_y_pixel_array = numpy.zeros((numFiles, 1))

# Center right of image
# Create arrays for the latitude, longitude, and pixel locations for the center
    right edge of the image
cRight_lat_array = numpy.zeros((numFiles, 1))
cRight_lon_array = numpy.zeros((numFiles, 1))
cRight_x_pixel_array = numpy.zeros((numFiles, 1))
cRight_y_pixel_array = numpy.zeros((numFiles, 1))

# Bottom left of image
# Create arrays for the latitude, longitude, and pixel locations for the bottom left
     corner of the image
bLeft_lat_array = numpy.zeros((numFiles, 1))
bLeft_lon_array = numpy.zeros((numFiles, 1))
bLeft_x_pixel_array = numpy.zeros((numFiles, 1))
bLeft_y_pixel_array = numpy.zeros((numFiles, 1))

# Bottom center image
# Create arrays for the latitude, longitude, and pixel locations for the bottom
    center of the image
bCenter_lat_array = numpy.zeros((numFiles, 1))
bCenter_lon_array = numpy.zeros((numFiles, 1))
bCenter_x_pixel_array = numpy.zeros((numFiles, 1))
bCenter_y_pixel_array = numpy.zeros((numFiles, 1))

# Bottom right of image
# Create arrays for the latitude, longitude, and pixel locations for the bottom
    right corner of the image
bRight_lat_array = numpy.zeros((numFiles, 1))
```

```
    bRight_lon_array = numpy.zeros((numFiles, 1))
    bRight_x_pixel_array = numpy.zeros((numFiles, 1))
    bRight_y_pixel_array = numpy.zeros((numFiles, 1))

# Write filenames to array. Used in kml (google earth) save
for j in range(0, numFiles):
    if file_names_array[j] == '':
        file_names_array[j] = filename
        break

#
    ##########################################################################################################################

# Check if the top pixel is not at the top of the image
# If the gimbal pitch angle for the top of the image based on the recorded pitch plus
    half of
#  the vertical field of view is > 0 degrees, use the calculated new top pixel row as
    the "top" of the image
if tilt_top >= -1:
    v_pixel_top = y_pixel_top
else:
    v_pixel_top = 0

# Calculate and implement geometric step to determine how many vertical pixels to skip
    over when calculating
#  pixel temperature
# The goal is to have higher resolution for steps at the top of the image as pixel rows
    at the top of the image
#  would result in a larger geographic distance away from the TANAB2 as compared to
    pixel rows near
#  the bottom of the image.
# Data associated with pixels at the top of the image should return surface temperatures
     further away from the
#  TANAB2 launch site and result in a more even and possibly consistent spatial surface
    temperature map

# Initialize the pixel step array
y_pixel_step = numpy.zeros((10, 1))
# Identify the coefficient to use in the geometric pixel step formula
aStepGeometric = 18
# Identify the constant to use in the geometric pixel step calculation
rStepGeometric = 1.41

# Start at the top of the image (Row 0), if a new "top" is chosen, a filtering loop
    below skips over any pixels
#  in y_pixel_step that are out of the calculated vertical pixel range.
y_pixel_step[0] = 0

# The second index in the geometric step function was selected to be pixel row 18.
y_pixel_step[1] = aStepGeometric
for GeometricStep in range(2, 10):
    # Calculate the geometric pixel step for the 8 remaining pixels and save to the
```

```
        appropriate array
    y_pixel_step[GeometricStep] = int(aStepGeometric * ((rStepGeometric) **
        GeometricStep))

print('The virtual pixel top is: '+str(v_pixel_top))
print('The y_pixel_step is as follows: '+str(y_pixel_step))

#
    ##########################################################################################################################

# This nested loop chooses pixels based on the predetermined horizontal pixel step and
    the
#  calculated geometric pixel step
# Within the loop, each pixel is georefereced with the derived mathematical formulas
    between pixels and
#  geographic distance. Surface tempertaures are calculated based on recorded pixel
    signal values
# The outer loop represents the horizontal (column) pixel step
for i in range(0, x_pixel_range, x_max_step):
    print('The pixel column number being processed now is: '+str(i))
    # Initialize a counter variable to be used to correspond to the geometric step
        matrix index
    # Count must be -1 as y_pixel_step[0] = 0 (if count = 0, y_pixel_step[1] = 18 and if
        a new "top" is NOT used,
    #  j must equal 0!
    count = -1
    # The inner loop represents the vertical (row) pixel step
    for j in y_pixel_step:
        # Add one to the counter variable
        count += 1
        # Check to see if the chosen geometric pixel step value is less than the virtual
            pixel top, if it is,
        #  continue to next y_pixel_step value
        if y_pixel_step[count] < v_pixel_top:
            continue

        print('The pixel row number being processed now is: '+str(j))

        # Need to convert data type of pixel step to int, same as i
        j = int(j)

        print('x Pixel Location: '+str(i))
        print('y Pixel Location: '+str(j)+'\n')

        # Pixel to Geographic distance relationship
        # Find Slope for line of sight for each specific pixel coordinate from the
            camera to the ground
        # Need Beta_new (tilt angle of camera given known y pixel coordinate)
        # Must correlate pixels to latitude/longitude... need to calculate new beta
            given pixel coordinates...
        #  From Sine Law solve for beta, Where kappa = 90-FoVV/2
        # Refer to diagrams of TANAB2 camera with respect to image projection on the
```

```
#  Earth's surface in thesis for further clarification
kappa = 90-FoVV/2

# Go from pixels to distance, Using the sine law, rearrange for Beta
beta_new_rads = -1*atan(((0.5*y_pixel_range-j)*sin(math.radians(0.5*FoVV)))/
                        (0.5*y_pixel_range*sin(math.radians(kappa))))+(math.
                            radians(0.5*FoVV))

# Convert beta_new to degrees
beta_new = math.degrees(beta_new_rads)

# Next get gamma_new. This is the angle away from the horizontal axis (zero
    degrees)
# corresponding to the current pixel
gamma_new = 90-abs(Pitch)+(FoVV/2)-beta_new
gamma_new_rads = math.radians(gamma_new)

# Calculate the slope for line of sight through the current pixel from camera.
slope = (1/tan(gamma_new_rads))*-1

# Call LandSlopeEquations Function
ground_elev_ASL_fitted, ground_elev_AGL = LandSlopeEquations(BaseAltitude,
    heading)

print('The slope for the line of sight from the camera to the ground on the
    current pixel is: '+str(slope))

# Find coefficients for line of sight from camera that intersects with the
    ground
line_of_sight_pixel = numpy.zeros(4)
line_of_sight_pixel[2] = slope
line_of_sight_pixel[3] = Altitude_AGL

# Create equations and detrend the data
# Create a numeric sequence starting at 0, ending at 30,000, the number of
    indices to use is equal to
#  the length of ground_elev_AGL (as calculated in LandSlopeEquations function)
    (given information,
#  the numerical difference between each value is calculated by the function
# x_test represents distance in meters away from the TANAB2 launch site
x_test_pixel = numpy.linspace(0, 30000, len(ground_elev_AGL))

# Make a polyfit of the x_test distance away from TANAB2 launch site with
    respect to
#  the detrended land surface elevation
# Return the coefficients for a 3rd order polynomial of the detrended surface
    elevation with
#  respect to distance away from the TANAB2 launch site
land_elevation_equation_pixel_AGL = numpy.polyfit(x_test_pixel, ground_elev_AGL,
     3)

# Find the roots and select the smallest positive value to be the horizontal
```

```
    distance away from the
#  TANAB2 launch location
intersections_pixel = land_elevation_equation_pixel_AGL - line_of_sight_pixel
Roots_new = numpy.roots(intersections_pixel)
print('The roots are: ' + str(Roots_new))

# Create an array for the roots that are real numbers (not complex)
real_roots_new = numpy.empty(3)
real_roots_new[:] = numpy.nan
for m in range(0, len(Roots_new)):
    if numpy.iscomplex(Roots_new[m]) == False:
        real_roots_new[m] = Roots_new[m]

# Initialize a variable that counts the number of non real roots
root_counter_new = 0

# Identify the number of non real roots
for non_real_root_new in range(0, len(Roots_new)):
    if real_roots_new[non_real_root_new] < 0 or numpy.isnan(real_roots_new[
        non_real_root_new]) == True:
        root_counter_new += 1

# If no real roots exist, then continue to the next image
if root_counter_new == len(Roots_new):
    continue
else:
    # Choose the root that is the smallest real positive solution to be the
        distance from the TANAB2
    #  launch location
    d_pixel_proj_ctr = min(n for n in real_roots_new if n > 0)
    # Convert d_center to km
    d_pixel_proj_ctr_km = d_pixel_proj_ctr / 1000

    # Put check in for d_pixel. If greater than 5 km (too far), continue on to
        next y pixel
    #  (This condition can be changed)
    if d_pixel_proj_ctr_km > 5:
        continue
    else:
        print('The horizontal geographic pixel distance as projected on the
            center of the image is: '
              +str(d_pixel_proj_ctr_km))

        # Get the alpha angle. The angle away from the geographic distance away
            from the
        #  TANAB2 and parallel to the camera line of sight
        # The alpha value is used to calculate the geographic distance away from
            the
        #  TANAB2 for pixels that are not parallel to the camera line of sight

        # Find the angle from the center line of the image given index i for the
            current pixel
```

```
# This will change the "effective" yaw angle
if i == 0:
    alpha = - FoVH/2
    alpha_rads = math.radians(alpha)
elif (i > 0) and (i < x_pixel_range/2):
    alpha = - (x_pixel_range/2-i) * FoVH / (x_pixel_range)
    alpha_rads = math.radians(alpha)
elif i == x_pixel_range/2:
    alpha_rads = 0
elif (i > x_pixel_range/2) and (i < x_pixel_range):
    alpha = (i - x_pixel_range / 2) * FoVH / (x_pixel_range)
    alpha_rads = math.radians(alpha)
elif i == x_pixel_range:
    alpha = FoVH/2
    alpha_rads = math.radians(alpha)

print ("alpha_rads is equal to: "+ str(alpha_rads))
print ("Yaw_rads is equal to: " + str(Yaw_rads))

# Find the d_hyp distance in km for each respective pixel
d_pixel_km = d_pixel_proj_ctr_km/(cos(alpha_rads))

# Ensure this angle is strictly positive and less than 2*pi
if Yaw_rads + alpha_rads < 0:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads + 2 * numpy.pi

elif Yaw_right_rads + alpha_rads > 2 * numpy.pi:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads - 2 * numpy.pi
else:
    Yaw_rads_adjusted = Yaw_rads + alpha_rads

print("Yaw_rads_adjusted is equal to: " + str(Yaw_rads_adjusted))

# Find the geographic coordinates for each specific pixel coordinate,
#  must add the calculated alpha to the Yaw value so we use
    Yaw_rads_adjusted
lat2_pixel = asin(sin(Latitude_rads) * cos(d_pixel_km / Radius_Earth) +
    cos(Latitude_rads)
                  * sin(d_pixel_km / Radius_Earth) * cos(
                      Yaw_rads_adjusted))
lon2_pixel = Longitude_rads + atan2(sin(Yaw_rads_adjusted) * sin(
    d_pixel_km / Radius_Earth)
                                    * cos(Latitude_rads), cos(d_pixel_km
                                        / Radius_Earth)
                                    - sin(Latitude_rads) * sin(
                                        lat2_pixel))

# Convert back to decimal degrees
lat2_pixel = math.degrees(lat2_pixel)
lon2_pixel = math.degrees(lon2_pixel)

print('The lat2_pixel is: '+str(lat2_pixel)+', given a x pixel of: '+
```

```
    str(i))
print('The lon2 pixel is: '+str(lon2_pixel)+', given a y pixel of: '+
    str(j)+'\n')

#
    ##############################################################################

# Temperature Calculation
# Some reference source on temperatures:
#  http://91.143.108.245/Downloads/Flir/Dokumentation/
    T559498$a461_Manual.pdf
# Temperature formula reference:
#  https://graftek.biz/system/files/137/original/
    FLIR_AX5_GenICam_ICD_Guide_052013.pdf?1376925336
# Radiance relation to A/D counts reference: http://flir.custhelp.com/ci
    /fattach/get/1667/
# Useful reference from FLIR for thermal imaging and A/D counts/Signals
    generated from
#  Thermal cameras: http://www.hoskin.ca/wp-content/uploads/2016/10/
    flir_thermal_camera_guide
# _for_research_professionals.pdf

# Extract the RAW total signal value contained within the specific pixel
     as denoted by i and j
# Return the value to a variable
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " +
    directory + "/" + filename +
                              " 2>/dev/zero | magick - -crop 1X1+" +
                                  str(i) + "+" + str(j) +
                              " -colorspace gray -format '%[mean]' info:
                                  "], shell=True,
                             stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

#
    ##############################################################################

# Calculate temperature for each specific pixel in in K and degC when
    emissivity < 1.0
# Call in Emis_29, Emis_31, Emis_32 and apply Wang et al. 2005
#  BroadBand Emissivity (BBE) formula

# Check if BBE variables are in local variables, if they are do not call
     in file
#  (do this for the first image only)
if 'emis_filename' not in locals():
```

```
# Emissivity values are derived from the MODIS11B3 monthly land
    surface emissivity file
# MODIS image and a grid of coordinates at 500m resolution were
    overlaid on each other in
#  QGIS (same procedure as the mining campaign)
# Using the point extract tool, emissivity data and the
    corresponding geographic
#  coordinates were extracted and saved to both a CSV and text file

# For July 2018 Images
emis_filename_Jul = '/export/home/users/username/Documents/DG_Temp/
    Guelph_2018/MODIS/' \
                    'Emissivity/LatLonEmisData_Jul18.csv'

# For August 2018 Images
emis_filename_Aug = '/export/home/users/username/Documents/DG_Temp/
    Guelph_2018/MODIS/' \
                    'Emissivity/LatLonEmisData_Aug18.csv'

# Call in emissivity data
emis_data_Jul = numpy.genfromtxt(emis_filename_Jul, delimiter=',',
    skip_header=1)

emis_data_Aug = numpy.genfromtxt(emis_filename_Aug, delimiter=',',
    skip_header=1)

# For July 2018
# Latitude of Emissivity values
emis_lat_Jul = emis_data_Jul[:,0]
# Longitude of Emissivity Values
emis_lon_Jul = emis_data_Jul[:,1]
# MODIS Band 32 Emissivity Values
emis_32_uncorrected_Jul = emis_data_Jul[:,2]
# MODIS Band 29 Emissivity Values
emis_29_uncorrected_Jul = emis_data_Jul[:,3]
# MODIS Band 31 Emissivity Values
emis_31_uncorrected_Jul = emis_data_Jul[:,4]

# For August 2018
# Latitude of Emissivity values
emis_lat_Aug = emis_data_Aug[:, 0]
# Longitude of Emissivity Values
emis_lon_Aug = emis_data_Aug[:, 1]
# MODIS Band 32 Emissivity Values
emis_32_uncorrected_Aug = emis_data_Aug[:, 2]
# MODIS Band 29 Emissivity Values
emis_29_uncorrected_Aug = emis_data_Aug[:, 3]
# MODIS Band 31 Emissivity Values
emis_31_uncorrected_Aug = emis_data_Aug[:, 4]

# Initialize corrected emissivity variables
# For July 2018
```

```
emis_32_corrected_Jul = numpy.zeros((len(emis_32_uncorrected_Jul),
    1))
emis_29_corrected_Jul = numpy.zeros((len(emis_32_uncorrected_Jul),
    1))
emis_31_corrected_Jul = numpy.zeros((len(emis_32_uncorrected_Jul),
    1))

# For August 2018
emis_32_corrected_Aug = numpy.zeros((len(emis_32_uncorrected_Aug),
    1))
emis_29_corrected_Aug = numpy.zeros((len(emis_32_uncorrected_Aug),
    1))
emis_31_corrected_Aug = numpy.zeros((len(emis_32_uncorrected_Aug),
    1))

# Convert all emissivity values  (multiply by scale factor and add
    offset) as per:
# MODIS documentation: https://lpdaac.usgs.gov/sites/default/files/
    public/
# product_documentation/mod11_user_guide.pdf
emis_scale = 0.002
emis_offset = 0.49

# Calculate the true emissivity values for each band by applying the
     appropriate scale
#  factor and additive offset for each index of each array
for k in range(0, len(emis_lat_Jul)):
    # For July 2018
    # Apply Emissivity scale/offset factors to Band 29
    emis_29_corrected_Jul[k] = (emis_29_uncorrected_Jul[k]*
        emis_scale)+emis_offset
    # Apply Emissivity scale/offset factors to Band 31
    emis_31_corrected_Jul[k] = (emis_31_uncorrected_Jul[k]*
        emis_scale)+emis_offset
    # Apply Emissivity scale/offset factors to Band 32
    emis_32_corrected_Jul[k] = (emis_32_uncorrected_Jul[k]*
        emis_scale)+emis_offset

    # For August 2018
    # Apply Emissivity scale/offset factors to Band 29
    emis_29_corrected_Aug[k] = (emis_29_uncorrected_Aug[k] *
        emis_scale) + emis_offset
    # Apply Emissivity scale/offset factors to Band 31
    emis_31_corrected_Aug[k] = (emis_31_uncorrected_Aug[k] *
        emis_scale) + emis_offset
    # Apply Emissivity scale/offset factors to Band 32
    emis_32_corrected_Aug[k] = (emis_32_uncorrected_Aug[k] *
        emis_scale) + emis_offset

# Create new array for BBE, is used to calculation ST
# For July 2018
BBEmissivity_Jul = numpy.zeros((len(emis_lat_Jul),1))
```

```
    # For August 2018
    BBEmissivity_Aug = numpy.zeros((len(emis_lat_Aug), 1))

    # Initialize coefficients for BBE formula as per Wang et al 2005 pg
        7 of 12 Table 2
    # See the following for more information: https://doi.org
        /10.1029/2004JD005566
    BBE_constant_29 = 0.2122
    BBE_constant_31 = 0.3859
    BBE_constant_32 = 0.4029

    # Calculate BBE for each index
    # Haversine Distance Formula from:
    #  https://stackoverflow.com/questions/19412462/getting-distance-
        between-two-points
    # -based-on-latitude-longitude
    for k in range(0, len(emis_lat_Jul)):
        BBEmissivity_Jul[k] = (BBE_constant_29*emis_29_corrected_Jul[k])
            +\
                              (BBE_constant_31*emis_31_corrected_Jul[k])
                                  +\
                              (BBE_constant_32*emis_32_corrected_Jul[k])
        BBEmissivity_Aug[k] = (BBE_constant_29*emis_29_corrected_Aug[k])
            +\
                              (BBE_constant_31*emis_31_corrected_Aug[k])
                                  +\
                              (BBE_constant_32*emis_32_corrected_Aug[k])
    # Initialize Haversine distance array
    # This array is used to calculate the geographic distance between
        the BBE values and the
    #  specific pixel location
    # The BBE index with smallest distance between the two geographic
        coordinates will be used as
    #  the emissivity value in the surface temperature calculation
    # Note: The length of emis_lat_Jul and emis_lat_Aug are the same.
        The latitude/longitude
    #  coordinates are the same for each case. Only emissivity changes
        between the two months
    haversine_d = numpy.empty((len(emis_lat_Jul), 1))

# Initialize haversine formula variables to be used in the
    HaversinePixelCalc function\
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

# Run following function with the @jit compiler in parallel
# Haversine Distance Formula from:
#  https://stackoverflow.com/questions/19412462/getting-distance-between
```

```
    -two-points
# -based-on-latitude-longitude
haversine_d = HaversinePixelCalc(emis_lat_Jul, lat2_pixel, emis_lon_Jul,
                                 lon2_pixel, Radius_Earth, haversine_d)

# Find minimum index of the output of the haversine formula with the
    smallest distance
# This will be the index that has the surface emissivity value that will
     be used
#  in the temperature calculation for the specific pixel
min_idx = numpy.argmin(haversine_d)

# For the temperature calculation assume that transmissivity is close to
     1
# Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

# Reflected Apparent Temperature as per FLIR manual
#  (http://www.cctvcentersl.es/upload/Manuales/A3xxx_A6xxx_manual_eng.
    pdf)
#  and image metadata (use ExifTool in Linux terminal)
# Note: Depending on the imaged surface, this paramater may change

# Return the reflected apparent temperature (degrees C) from the image
    metadata with
#  the Linux terminal and ExifTool
refl_temp_degC = subprocess.Popen(["exiftool -b -
    ReflectedApparentTemperature "
                                   + directory + "/" + filename], shell=
                                       True,
                                  stdout=subprocess.PIPE).communicate()
                                      [0]

# Decode the value and convert its type to a float
refl_temp_degC = refl_temp_degC.decode("utf-8")
refl_temp_degC = float(refl_temp_degC)

# Convert reflected apparent temp from degC to K
refl_temp_K = pytemperature.c2k(refl_temp_degC)

# Get Raw reflected apparent temperature signal value
# See the FLIR Manual: http://www.cctvcentersl.es/upload/Manuales/
    A3xxx_A6xxx_manual_eng.pdf and
RAWrefl = (R1/(R2*(math.exp(B/(refl_temp_K))-F))-planck_O)

# RAWrefl remains the same for all pixels as it is a function of the
    assumed
#  constant apparent reflected temperature

# Check to see if July or August BBE should be used
if mnth == str(7):
    RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
        BBEmissivity_Jul[min_idx]
```

```
elif mnth == str(8):
    RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
        BBEmissivity_Aug[min_idx]
else:
    print('Double check the value and data type of the mnth variable')

# Rearrange the RAW object signal value to calculate the object
    temperature of each pixel in
#  degC and Kelvin
# Consider the case for when emissivity is not equivalent to 1
LST_kelvin = (B/numpy.log(R1/(R2*(RAWobj+planck_O))+F))
LST_degree = (B/numpy.log(R1/(R2*(RAWobj+planck_O))+F)-273.15)

# Save data to matrix for the specific image
# The data in this matrix is then saved to a master matrix which will
    include all surface
#  temperature data for pixels from each file in the 'RawImages'
    directory
# Save the filename for each corresponding pixel location, not used in
    data analysis,
#  only used as a check as the files are not processed chronologically
filename_image[i][j][0] = filename

# Save the year for each corresponding pixel location for when the image
     was recorded
image_matrix[i][j][0] = yr
# Save the month for each corresponding pixel location for when the
    image was recorded
image_matrix[i][j][1] = mnth
# Save the day for each corresponding pixel location for when the image
    was recorded
image_matrix[i][j][2] = day
# Save the hour for each corresponding pixel location for when the image
     was recorded
image_matrix[i][j][3] = hr
# Save the minute for each corresponding pixel location for when the
    image was recorded
image_matrix[i][j][4] = minute
# Save the calculated latitude for each corresponding pixel location
image_matrix[i][j][5] = lat2_pixel
# Save the calculated longitude for each corresponding pixel location
image_matrix[i][j][6] = lon2_pixel
# Save the horizontal pixel coordinate that was processed to obtain ST
image_matrix[i][j][7] = i
# Save the vertical pixel coordinate that was processed to obtain ST
image_matrix[i][j][8] = j
# Save the ST in kelvin of each corresponding pixel location where
    emissivity does not equal 1
image_matrix[i][j][9] = LST_kelvin
# Save the ST in degC of each corresponding pixel location where
    emissivity does not equal 1
image_matrix[i][j][10] = LST_degree
```

```
#
    ##########################################################################################################################

# Save known latitude, longitude, and x/y pixels to arrays
# TANAB2 launch location
for origin in range(0, numFiles):
    if int(lat_TANAB_array[origin]) == 0:
        lat_TANAB_array[origin] = Latitude
        lon_TANAB_array[origin] = Longitude
        break

# Save known location for the top left pixel
# Initialize variables identifying top left pixel in terms of horizontal/vertical pixel
    row/column location
horiz_pixel = 0
vert_pixel = v_pixel_top

# Extract the RAW total value for the top left pixel through the Linux terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info: "],
                                shell=True, stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW total as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW total from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_top' in locals():
    # Calculate Haversine distance for the top left pixel to identify the emissivity
    #  value to use in the ST calculation
    haversine_d = HaversinePixelCalc_top_left(emis_lat_Jul, lat2_top_left, emis_lon_Jul,
                                              lon2_top_left, Radius_Earth, haversine_d)

    # Find the index with the lowest distance between the 2 geographic coordinates
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305
```

```
        # Check to see if July or August BBE should be used
        if mnth == str(7):
            RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
                BBEmissivity_Jul[min_idx]
        elif mnth == str(8):
            RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
                BBEmissivity_Aug[min_idx]
        else:
            print('Double␣check␣the␣value␣and␣data␣type␣of␣the␣mnth␣variable')

        # Calculate temperature of each pixel in kelvin and degC respectively
        LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_top_left
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_top_left
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(tLeft_lat_array[i]) == 0:
                tLeft_lat_array[i] = lat2_top_left
                tLeft_lon_array[i] = lon2_top_left
                tLeft_x_pixel_array[i] = horiz_pixel
                tLeft_y_pixel_array[i] = vert_pixel
                break

#
    ##########################################################################################################

# Save known location for the top center pixel
# Initialize variables identifying top center pixel in terms of horizontal/vertical
    pixel row/column location
horiz_pixel = int(x_pixel_range/2)
vert_pixel = v_pixel_top

# Extract the RAW total value for the top center pixel through the Linux terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool␣-b␣-RawThermalImage␣" + directory + "/" +
    filename +
                             "␣2>/dev/zero␣|␣magick␣-␣-crop␣1X1+" + str(horiz_pixel) +
```

```
                                    "+" + str(vert_pixel) +
                                "␣-colorspace␣gray␣-format␣'%[mean]'␣info:␣"], shell=True,
                               stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

# Call Haversine top center pixel function to identify the emissivity value to use in
    the
#  surface temperature calculation
if 'lat2_top' in locals():
    haversine_d = HaversinePixelCalc_top_center(emis_lat_Jul, lat2_top, emis_lon_Jul,
                                                lon2_top, Radius_Earth, haversine_d)

    # Find the index with the lowest distance between the 2 geographic coordinates
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used and calculate Raw object
        temperature
    if mnth == str(7):
        RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
            BBEmissivity_Jul[min_idx]
    elif mnth == str(8):
        RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
            BBEmissivity_Aug[min_idx]
    else:
        print('Double␣check␣the␣value␣and␣data␣type␣of␣the␣mnth␣variable')

    # Calculate temperature of each pixel in kelvin and degC respectively
    LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
    LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

    # Save data to matrix for specific image
    filename_image[horiz_pixel][vert_pixel][0] = filename

    image_matrix[horiz_pixel][vert_pixel][0] = yr
    image_matrix[horiz_pixel][vert_pixel][1] = mnth
    image_matrix[horiz_pixel][vert_pixel][2] = day
    image_matrix[horiz_pixel][vert_pixel][3] = hr
```

```
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_top
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_top
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(tCenter_lat_array[i]) == 0:
                tCenter_lat_array[i] = lat2_top
                tCenter_lon_array[i] = lon2_top
                tCenter_x_pixel_array[i] = horiz_pixel
                tCenter_y_pixel_array[i] = vert_pixel
                break

#
    ##########################################################################################################

# Save known coordinates for the top right pixel
# Initialize variables identifying top right pixel in terms of horizontal/vertical pixel
    row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = v_pixel_top

# Extract the RAW total value for the top right pixel through the terminal with ExifTool
    and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info: "], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_top' in locals():
    # Call Haversine top right pixel function to identify the emissivity value to use in
        the
    #  surface temperature calculation
```

```
        haversine_d = HaversinePixelCalc_top_right(emis_lat_Jul, lat2_top_right,
            emis_lon_Jul,
                                                   lon2_top_right, Radius_Earth, haversine_d
                                                       )

        # Find the index with the lowest distance between the 2 geographic coordinates
        min_idx = numpy.argmin(haversine_d)

        # For the temperature calculation assume that transmissivity is close to 1
        # Source: Usamentiaga et al: https://doi.org/10.3390/s140712305

        # Check to see if July or August BBE should be used
        if mnth == str(7):
            RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
                BBEmissivity_Jul[min_idx]
        elif mnth == str(8):
            RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
                BBEmissivity_Aug[min_idx]
        else:
            print('Double check the value and data type of the mnth variable')

        # Calculate temperature of each pixel in kelvin and degC respectively
        LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_top_right
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_top_right
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(tRight_lat_array[i]) == 0:
                tRight_lat_array[i] = lat2_top_right
                tRight_lon_array[i] = lon2_top_right
                tRight_x_pixel_array[i] = horiz_pixel
                tRight_y_pixel_array[i] = vert_pixel
                break
#
    ##########################################################################################################################

# For the middle left pixel
```

```
# Initialize variables identifying the middle left pixel in terms of horizontal/vertical
     pixel row/column location
horiz_pixel = 0
vert_pixel = int(y_pixel_range / 2)

# Extract the RAW total value for the middle left pixel through the Linux terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info:"], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_center' in locals():
# Call Haversine center left pixel function to identify the emissivity value to use in
    the surface
#  temperature calculation
    haversine_d = HaversinePixelCalc_center_left(emis_lat_Jul, lat2_center_left,
        emis_lon_Jul, lon2_center_left,
                                                 Radius_Earth, haversine_d)

    # Find the index with the lowest distance between the 2 geographic coordinates
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used
    if mnth == str(7):
        RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
            BBEmissivity_Jul[min_idx]
    elif mnth == str(8):
        RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
            BBEmissivity_Aug[min_idx]
    else:
        print('Double check the value and data type of the mnth variable')

    # Calculate temperature of each pixel in kelvin and degC respectively
```

```
        LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_center_left
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_center_left
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(cLeft_lat_array[i]) == 0:
                cLeft_lat_array[i] = lat2_center_left
                cLeft_lon_array[i] = lon2_center_left
                cLeft_x_pixel_array[i] = horiz_pixel
                cLeft_y_pixel_array[i] = vert_pixel
                break
#
        ##########################################################################################################################

    # For the middle center pixel
    # Initialize variables identifying the center pixel in terms of horizontal/vertical
        pixel
    #  row/column location
    horiz_pixel = int(x_pixel_range/2)
    vert_pixel = int(y_pixel_range/2)

    # Extract the RAW total value for the center pixel through the Linux terminal with
        ExifTool and ImageMagcick
    RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
        filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
                                  " -colorspace gray -format '%[mean]' info:"], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]

    # Need to decode RAW as its a bytes object to a string
    RAW_total = RAW_total.decode("utf-8")

    # Convert RAW from string to float
    RAW_total = float(RAW_total)

    # Initialize arrays used in haversine calculation
```

```
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_center' in locals():
    # Call Haversine center pixel function to identify the emissivity value to use in
        the surface
    #  temperature calculation
    haversine_d = HaversinePixelCalc_center(emis_lat_Jul, lat2_center, emis_lon_Jul,
                                            lon2_center, Radius_Earth, haversine_d)

    # Find minimum distance index
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used
    if mnth == str(7):
        RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
            BBEmissivity_Jul[min_idx]
    elif mnth == str(8):
        RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
            BBEmissivity_Aug[min_idx]
    else:
        print('Double␣check␣the␣value␣and␣data␣type␣of␣the␣mnth␣variable')

    # Calculate temperature of each pixel in degC
    LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
    LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

    # Save data to matrix for specific image
    filename_image[horiz_pixel][vert_pixel][0] = filename

    image_matrix[horiz_pixel][vert_pixel][0] = yr
    image_matrix[horiz_pixel][vert_pixel][1] = mnth
    image_matrix[horiz_pixel][vert_pixel][2] = day
    image_matrix[horiz_pixel][vert_pixel][3] = hr
    image_matrix[horiz_pixel][vert_pixel][4] = minute
    image_matrix[horiz_pixel][vert_pixel][5] = lat2_center
    image_matrix[horiz_pixel][vert_pixel][6] = lon2_center
    image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
    image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
    image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
    image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

    # Save latitude, longitude, x, and y pixel values to arrays
    for i in range(0, numFiles):
        if int(center_lat_array[i]) == 0:
            center_lat_array[i] = lat2_center
```

```
            center_lon_array[i] = lon2_center
            center_x_pixel_array[i] = horiz_pixel
            center_y_pixel_array[i] = vert_pixel
            break

#
    ##########################################################################################################################

# For the middle right pixel
# Initialize variables identifying the middle right pixel in terms of horizontal/
    vertical pixel row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = int(y_pixel_range/2)

# Extract the RAW total value for the middle right pixel through the terminal with
    Exiftool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b --RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info:"], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_center' in locals():
    # Call Haversine center right pixel function to identify the emissivity value to use
        in the surface
    #  temperature calculation
    haversine_d = HaversinePixelCalc_center_right(emis_lat_Jul, lat2_center_right,
        emis_lon_Jul,
                                                  lon2_center_right, Radius_Earth,
                                                      haversine_d)

    # Find minimum distance index
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used
```

```
        if mnth == str(7):
            RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
                BBEmissivity_Jul[min_idx]
        elif mnth == str(8):
            RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
                BBEmissivity_Aug[min_idx]
        else:
            print('Double check the value and data type of the mnth variable')

        # Calculate temperature of each pixel in Kelvin and degC respectively
        LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_center_right
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_center_right
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(cRight_lat_array[i]) == 0:
                cRight_lat_array[i] = lat2_center_right
                cRight_lon_array[i] = lon2_center_right
                cRight_x_pixel_array[i] = horiz_pixel
                cRight_y_pixel_array[i] = vert_pixel
                break

#
    ##########################################################################################################

# For the bottom left pixel
# Initialize variables identifying the bottom left pixel in terms of horizontal/vertical
    pixel row/column location
horiz_pixel = 0
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom left pixel through the Linux terminal with
    ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
```

```
                                    "␣-colorspace␣gray␣-format␣'%[mean]'␣info:␣"], shell=True,
                                   stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_bottom' in locals():
    # Call Haversine bottom left pixel function to identify the emissivity value to use
        in the
    #  surface temperature calculation
    haversine_d = HaversinePixelCalc_bottom_left(emis_lat_Jul, lat2_bottom_left,
        emis_lon_Jul,
                                                 lon2_bottom_left, Radius_Earth,
                                                     haversine_d)

    # Find minimum distance index
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used
    if mnth == str(7):
        RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
            BBEmissivity_Jul[min_idx]
    elif mnth == str(8):
        RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
            BBEmissivity_Aug[min_idx]
    else:
        print('Double␣check␣the␣value␣and␣data␣type␣of␣the␣mnth␣variable')

    # Calculate temperature of each pixel in kelvin and degC respectively
    LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
    LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

    # Save data to matrix for specific image
    filename_image[horiz_pixel][vert_pixel][0] = filename

    image_matrix[horiz_pixel][vert_pixel][0] = yr
    image_matrix[horiz_pixel][vert_pixel][1] = mnth
    image_matrix[horiz_pixel][vert_pixel][2] = day
    image_matrix[horiz_pixel][vert_pixel][3] = hr
```

```
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom_left
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom_left
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(bLeft_lat_array[i]) == 0:
                bLeft_lat_array[i] = lat2_bottom_left
                bLeft_lon_array[i] = lon2_bottom_left
                bLeft_x_pixel_array[i] = horiz_pixel
                bLeft_y_pixel_array[i] = vert_pixel
                break
#
    ##########################################################################################################

# For the bottom center pixel
# Initialize variables identifying the bottom center pixel in terms of horizontal/
    vertical
#  pixel row/column location
horiz_pixel = int(x_pixel_range/2)
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom center pixel through the Linux terminal
    with ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                  " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                      "+" + str(vert_pixel) +
                                  " -colorspace gray -format '%[mean]' info: "], shell=True,
                                 stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_bottom' in locals():
    # Call Haversine bottom pixel function to identify the emissivity value to use in
        the surface
    #  temperature calculation
```

```
        haversine_d = HaversinePixelCalc_bottom(emis_lat_Jul, lat2_bottom, emis_lon_Jul,
            lon2_bottom,
                                                Radius_Earth, haversine_d)

        # Find minimum distance index
        min_idx = numpy.argmin(haversine_d)

        # For the temperature calculation assume that transmissivity is close to 1
        # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

        # Check to see if July or August BBE should be used
        if mnth == str(7):
            RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
                BBEmissivity_Jul[min_idx]
        elif mnth == str(8):
            RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
                BBEmissivity_Aug[min_idx]
        else:
            print('Double check the value and data type of the mnth variable')

        # Calculate temperature of each pixel in kelvin and degC respectively
        LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(bCenter_lat_array[i]) == 0:
                bCenter_lat_array[i] = lat2_bottom
                bCenter_lon_array[i] = lon2_bottom
                bCenter_x_pixel_array[i] = horiz_pixel
                bCenter_y_pixel_array[i] = vert_pixel
                break

#
    ##########################################################################################################################

# For the bottom right pixel
```

```
# Initialize variables identifying the bottom right pixel in terms of horizontal/
    vertical pixel row/column location
horiz_pixel = x_pixel_range-1
vert_pixel = y_pixel_range-1

# Extract the RAW total value for the bottom right pixel through the Linux terminal with
     ExifTool and ImageMagcick
RAW_total = subprocess.Popen(["exiftool -b -RawThermalImage " + directory + "/" +
    filename +
                                 " 2>/dev/zero | magick - -crop 1X1+" + str(horiz_pixel) +
                                     "+" + str(vert_pixel) +
                                 " -colorspace gray -format '%[mean]' info:"], shell=True,
                                stdout=subprocess.PIPE).communicate()[0]

# Need to decode RAW as its a bytes object to a string
RAW_total = RAW_total.decode("utf-8")

# Convert RAW from string to float
RAW_total = float(RAW_total)

# Initialize arrays used in haversine calculation
haversine_d[:] = numpy.nan
haversine_a = 0
haversine_c = 0
haversine_dlat = 0
haversine_dlon = 0

if 'lat2_bottom' in locals():
    # Call Haversine bottom right pixel function
    haversine_d = HaversinePixelCalc_bottom_right(emis_lat_Jul, lat2_bottom_right,
        emis_lon_Jul,
                                                  lon2_bottom_right, Radius_Earth,
                                                      haversine_d)

    # Find minimum distance index
    min_idx = numpy.argmin(haversine_d)

    # For the temperature calculation assume that transmissivity is close to 1
    # Source: Usamentiaga et al.: https://doi.org/10.3390/s140712305

    # Check to see if July or August BBE should be used
    if mnth == str(7):
        RAWobj = (RAW_total - (1 - BBEmissivity_Jul[min_idx]) * RAWrefl) /
            BBEmissivity_Jul[min_idx]
    elif mnth == str(8):
        RAWobj = (RAW_total - (1 - BBEmissivity_Aug[min_idx]) * RAWrefl) /
            BBEmissivity_Aug[min_idx]
    else:
        print('Double check the value and data type of the mnth variable')

    # Calculate temperature of each pixel in kelvin and degC respectively
    LST_kelvin = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F))
```

```
        LST_degree = (B / numpy.log(R1 / (R2 * (RAWobj + planck_O)) + F) - 273.15)

        # Save data to matrix for specific image
        filename_image[horiz_pixel][vert_pixel][0] = filename

        image_matrix[horiz_pixel][vert_pixel][0] = yr
        image_matrix[horiz_pixel][vert_pixel][1] = mnth
        image_matrix[horiz_pixel][vert_pixel][2] = day
        image_matrix[horiz_pixel][vert_pixel][3] = hr
        image_matrix[horiz_pixel][vert_pixel][4] = minute
        image_matrix[horiz_pixel][vert_pixel][5] = lat2_bottom_right
        image_matrix[horiz_pixel][vert_pixel][6] = lon2_bottom_right
        image_matrix[horiz_pixel][vert_pixel][7] = horiz_pixel
        image_matrix[horiz_pixel][vert_pixel][8] = vert_pixel
        image_matrix[horiz_pixel][vert_pixel][9] = LST_kelvin
        image_matrix[horiz_pixel][vert_pixel][10] = LST_degree

        # Save latitude, longitude, x, and y pixel values to arrays
        for i in range(0, numFiles):
            if int(bRight_lat_array[i]) == 0:
                bRight_lat_array[i] = lat2_bottom_right
                bRight_lon_array[i] = lon2_bottom_right
                bRight_x_pixel_array[i] = horiz_pixel
                bRight_y_pixel_array[i] = vert_pixel
                break
#
        ##########################################################################################################################

    # Delete the following variables for the next loop iteration
    if 'lat2_top' in locals():
        del lat2_top_left
        del lon2_top_left
        del lat2_top
        del lon2_top
        del lat2_top_right
        del lon2_top_right
    if 'lat2_center' in locals():
        del lat2_center_left
        del lon2_center_left
        del lat2_center
        del lon2_center
        del lat2_center_right
        del lon2_center_right
    if 'lat2_bottom' in locals():
        del lat2_bottom_left
        del lon2_bottom_left
        del lat2_bottom
        del lon2_bottom
        del lat2_bottom_right
        del lon2_bottom_right
    # Save image matrix data to master matrix
    all_pixel_data_multi_image = SaveMasterMatrix(x_pixel_range, v_pixel_top, y_pixel_range,
```

```
        image_matrix,
                                                    all_pixel_data_multi_image, filename_image
                                                        , filenames_total)

# The number of elements is equivalent to the total length of the all_pixel_data_multi_image
    array
#  (the total length is the maximum possible data points extracted from each image
num_elements = int((numFiles*x_max_step*y_max_step)-1)

# Variable corresponding to the row where the Nan values start
row_nan = 0

# Find the index where the Nan rows start (do this so Nan values are not written to the file
    )
# This variable will be used when saving the data to omit any Nan data values from the
    Processed_data text file
for i in range(0, num_elements):
    if numpy.isnan(all_pixel_data_multi_image[i][5]) == True:
        row_nan = i
        break

# Today's date is
today_date = datetime.date.today().strftime("%B_%d_%Y")

# Write Geographic, temperature, and Image Pixel Data to File. The name of the file XXXMEDIA
    ,
#  corresponds to the file name of images being processed
outputFileName = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/Processed_Data/
    August/102MEDIA_Temps.txt'
outputFile = open(outputFileName, 'w')
outputFile.write("# Date, Time, Lat, Long and Temp for each image \n")
outputFile.write("#By: Ryan Byerlay \n")
outputFile.write("#Created on "+today_date+" \n")
outputFile.write("#Recorded Time is Local Time (EDT) \n")
outputFile.write("#NOTE: The column represnting minutes may have a single digit such as 0X
    where X"
                 " is the number in the column so at the top of the hr only 0 would be
                     present \n")
outputFile.write("#0: Picture File Name \t #1:Year \t\t #2:Month \t\t #3:Day \t #4:Hour \t #5:
    Minute \t"
                 " #6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t #9: Y Pixel
                     Coordinate \t "
                 "#12: Temperature (K) (Emis != 1) \t #13: Temperature (C) (Emis != 1) \n")

# Save data to file
for i in range(0, row_nan):
     outputFile.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i \t %i \t %f \t %f"
                      " \n" % (filenames_total[i][0], int(all_pixel_data_multi_image[i][0]),
                               int(all_pixel_data_multi_image[i][1]), int(
                                   all_pixel_data_multi_image[i][2]),
                               int(all_pixel_data_multi_image[i][3]), int(
                                   all_pixel_data_multi_image[i][4]),
```

```
                                all_pixel_data_multi_image[i][5], all_pixel_data_multi_image[
                                    i][6],
                                int(all_pixel_data_multi_image[i][7]), int(
                                    all_pixel_data_multi_image[i][8]),
                                all_pixel_data_multi_image[i][9], all_pixel_data_multi_image[
                                    i][10]))
outputFile.close()

#
    ##########################################################################################################################

# Function to populate known latitude/longitude coordinates for each image
def determineFileName(filenames_total, file_names_array):
    for w in range(0, numFiles):
        if filenames_total[k][0] == file_names_array[w]:
            # NOTE x_latitudes[0] and y_longitudes[0] correspond to the TANAB2 location
            # For every new filename populate the known latitude, longitude, x pixel, and y
                pixel arrays
            x_Latitudes = [lat_TANAB_array[w], tLeft_lat_array[w], tCenter_lat_array[w],
                tRight_lat_array[w],
                           cLeft_lat_array[w], center_lat_array[w], cRight_lat_array[w],
                               bLeft_lat_array[w],
                           bCenter_lat_array[w], bRight_lat_array[w]]
            y_Longitudes = [lon_TANAB_array[w], tLeft_lon_array[w], tCenter_lon_array[w],
                tRight_lon_array[w],
                            cLeft_lon_array[w], center_lon_array[w], cRight_lon_array[w],
                                bLeft_lon_array[w],
                            bCenter_lon_array[w], bRight_lon_array[w]]
            x_pixels = [tLeft_x_pixel_array[w], tCenter_x_pixel_array[w],
                tRight_x_pixel_array[w],
                        cLeft_x_pixel_array[w], center_x_pixel_array[w],
                            cRight_x_pixel_array[w],
                        bLeft_x_pixel_array[w], bCenter_x_pixel_array[w],
                            bRight_x_pixel_array[w]]
            y_pixels = [tLeft_y_pixel_array[w], tCenter_y_pixel_array[w],
                tRight_y_pixel_array[w],
                        cLeft_y_pixel_array[w], center_y_pixel_array[w],
                            cRight_y_pixel_array[w],
                        bLeft_y_pixel_array[w], bCenter_y_pixel_array[w],
                            bRight_y_pixel_array[w]]
            break
    return x_Latitudes, y_Longitudes, x_pixels, y_pixels

#
    ##########################################################################################################################

# Save kml (Google Earth) file
# Save edge coordinates as red markers and inner image coordinates as yellow markers
kml = simplekml.Kml(open=1)
pt_label = ['Balloon', 'Top Left', 'Top Center', 'Top Right', 'Center Left', 'Center', '
    Center Right',
            'Bottom Left', 'Bottom Center', 'Bottom Right']
```

```
# Loop through all rows of final save matrix
for k in range(0, row_nan):
    # Find indices where the index and index+1 has mismatched file names
    # If file names are not equal, then save kml file for the specific image file
    if filenames_total[k][0] != filenames_total[k+1][0]:
        # Initialize variables
        # Consider edge coordinates and TANAB2 location for latitudes and longitudes
        x_Latitudes = numpy.zeros(10)
        y_Longitudes = numpy.zeros(10)
        # Only consider pixel coordinates for the specific image
        x_pixels = numpy.zeros(9)
        y_pixels = numpy.zeros(9)

        # Concatenate latitudes, longitudes, and pixel arrays accordingly and return
        x_Latitudes, y_Longitudes, x_pixels, y_pixels = determineFileName(filenames_total,
            file_names_array)

        # Save edge coordinate points to kml file
        # Set original counter value
        i = 0
        while i <= 9:
            known_pnts = kml.newpoint(name=str(pt_label[i]), coords=[(float(y_Longitudes[i])
                , float(x_Latitudes[i]))])
            known_pnts.style.iconstyle.color = simplekml.Color.red
            # Increase counter by 1
            i += 1

        # Save existing kml file, change July/August and XXXMEDIA directories based on the
            RawImages being processed
        kml.save("/export/home/users/username/Documents/DG_Temp/Guelph_2018/
            Google_Earth_Projections/August/102MEDIA/"
                "Lat_Lon_visualize_"+str(filenames_total[k][0])+".kml")

        # Delete old kml file variables including kml, known_pnts, x_coordinates,
            y_coordinates to get ready for
        #  next image file
        del kml
        del known_pnts
        del x_Latitudes
        del y_Longitudes

        # Create new kml file
        if 'kml' not in locals():
            kml = simplekml.Kml(open=1)
    # Save specific coordinate to existing kml file
    else:
        pnt = kml.newpoint(name='P('+str(all_pixel_data_multi_image[k][7])+','+str(
            all_pixel_data_multi_image[k][8])+
                               ')',coords = [(all_pixel_data_multi_image[k][6],
                                   all_pixel_data_multi_image[k][5])])

# Print process time in seconds
```

```
end = time.time()

# Get ending run time
print('The total run time of this script is: '+str(end-start)+' s')
```

A.3.5 Data Separation for Diurnal Temperature Mapping

```
import numpy
from numba import jit

# Current as of October 19, 2019
# Load in processed data and split up the data into five four-hour time intervals
    considering
#  data from each TANAB2 launch

# import the combined data file
filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/Processed_Data/
    Guelph_2018_Processed_Data.txt'

# Load data and separate columns
data = numpy.genfromtxt(filename, skip_header=6)

filename = data[:,0]
year = data[:,1]
month = data[:,2]
day = data[:,3]
hour = data[:,4]
min = data[:,5]
lat = data[:,6]
lon = data[:,7]
xpix = data[:,8]
ypix = data[:,9]
tempk_emis = data[:,10]
tempc_emis = data[:,11]

len_year = int(len(year))

# Create new arrays for data
zero_four_array = numpy.zeros((14, len_year))
four_eight_array = numpy.zeros((14, len_year))
eight_twelve_array = numpy.zeros((14, len_year))
twelve_sixteen_array = numpy.zeros((14, len_year))
sixteen_twenty_array = numpy.zeros((14, len_year))
twenty_twentyfour_array = numpy.zeros((14, len_year))

@jit(nopython=True, parallel=True)
def FindHour(year, month, day, hour, min, lat, lon, xpix, ypix, tempk_emis, tempc_emis,
    zero_four_array,
            four_eight_array, eight_twelve_array, twelve_sixteen_array,
                sixteen_twenty_array, twenty_twentyfour_array):
    for i in range(0, len(year)):
```

```
print(i)
if (int(hour[i]) < 0):
    print('This hour is less than Zero. There is a problem with the hour in the
        index')

# 00:00 to 04:00 Check
if ((int(hour[i]) >= 0) and (int(hour[i] <= 3))) or int(hour[i]) == 24:
    for j in range(0, len(year)):
        if zero_four_array[1][j] == 0:
            zero_four_array[1][j] = year[i]
            zero_four_array[2][j] = month[i]
            zero_four_array[3][j] = day[i]
            zero_four_array[4][j] = hour[i]
            zero_four_array[5][j] = min[i]
            zero_four_array[6][j] = lat[i]
            zero_four_array[7][j] = lon[i]
            zero_four_array[8][j] = xpix[i]
            zero_four_array[9][j] = ypix[i]
            zero_four_array[10][j] = tempk_emis[i]
            zero_four_array[11][j] = tempc_emis[i]
            break

# 04:00 to 08:00 check
elif (int(hour[i]) >= 4 and int(hour[i]) <= 7):
    for j in range(0, len(year)):
       if four_eight_array[1][j] == 0:
            four_eight_array[1][j] = year[i]
            four_eight_array[2][j] = month[i]
            four_eight_array[3][j] = day[i]
            four_eight_array[4][j] = hour[i]
            four_eight_array[5][j] = min[i]
            four_eight_array[6][j] = lat[i]
            four_eight_array[7][j] = lon[i]
            four_eight_array[8][j] = xpix[i]
            four_eight_array[9][j] = ypix[i]
            four_eight_array[10][j] = tempk_emis[i]
            four_eight_array[11][j] = tempc_emis[i]
            break

# 08:00 to 12:00 check
elif (int(hour[i]) >= 8 and int(hour[i]) <= 11) :
    for j in range(0, len(year)):
        if eight_twelve_array[1][j] == 0:
            eight_twelve_array[1][j] = year[i]
            eight_twelve_array[2][j] = month[i]
            eight_twelve_array[3][j] = day[i]
            eight_twelve_array[4][j] = hour[i]
            eight_twelve_array[5][j] = min[i]
            eight_twelve_array[6][j] = lat[i]
            eight_twelve_array[7][j] = lon[i]
            eight_twelve_array[8][j] = xpix[i]
            eight_twelve_array[9][j] = ypix[i]
```

```
                eight_twelve_array[10][j] = tempk_emis[i]
                eight_twelve_array[11][j] = tempc_emis[i]
                break

    # 12:00 to 16:00 check
    elif (int(hour[i]) >= 12 and int(hour[i]) <= 15) :
        for j in range(0, len(year)):
            if twelve_sixteen_array[1][j] == 0:
                twelve_sixteen_array[1][j] = year[i]
                twelve_sixteen_array[2][j] = month[i]
                twelve_sixteen_array[3][j] = day[i]
                twelve_sixteen_array[4][j] = hour[i]
                twelve_sixteen_array[5][j] = min[i]
                twelve_sixteen_array[6][j] = lat[i]
                twelve_sixteen_array[7][j] = lon[i]
                twelve_sixteen_array[8][j] = xpix[i]
                twelve_sixteen_array[9][j] = ypix[i]
                twelve_sixteen_array[10][j] = tempk_emis[i]
                twelve_sixteen_array[11][j] = tempc_emis[i]
                break

    # 16:00 to 20:00 check
    elif (int(hour[i]) >= 16 and int(hour[i]) <= 19):
        for j in range(0, len(year)):
            if sixteen_twenty_array[1][j] == 0:
                sixteen_twenty_array[1][j] = year[i]
                sixteen_twenty_array[2][j] = month[i]
                sixteen_twenty_array[3][j] = day[i]
                sixteen_twenty_array[4][j] = hour[i]
                sixteen_twenty_array[5][j] = min[i]
                sixteen_twenty_array[6][j] = lat[i]
                sixteen_twenty_array[7][j] = lon[i]
                sixteen_twenty_array[8][j] = xpix[i]
                sixteen_twenty_array[9][j] = ypix[i]
                sixteen_twenty_array[10][j] = tempk_emis[i]
                sixteen_twenty_array[11][j] = tempc_emis[i]
                break

    # 20:00 to 24:00 check
    elif (int(hour[i]) >= 20 and int(hour[i]) <= 23):
        for j in range(0, len(year)):
            if twenty_twentyfour_array[1][j] == 0:
                twenty_twentyfour_array[1][j] = year[i]
                twenty_twentyfour_array[2][j] = month[i]
                twenty_twentyfour_array[3][j] = day[i]
                twenty_twentyfour_array[4][j] = hour[i]
                twenty_twentyfour_array[5][j] = min[i]
                twenty_twentyfour_array[6][j] = lat[i]
                twenty_twentyfour_array[7][j] = lon[i]
                twenty_twentyfour_array[8][j] = xpix[i]
                twenty_twentyfour_array[9][j] = ypix[i]
                twenty_twentyfour_array[10][j] = tempk_emis[i]
```

```
                    twenty_twentyfour_array[11][j] = tempc_emis[i]
                    break

    return zero_four_array, eight_twelve_array, twelve_sixteen_array, sixteen_twenty_array,
        twenty_twentyfour_array

# Call functions
FindHour(year, month, day, hour, min, lat, lon, xpix, ypix, tempk_emis, tempc_emis,
    zero_four_array,
        four_eight_array, eight_twelve_array, twelve_sixteen_array, sixteen_twenty_array,
            twenty_twentyfour_array)

# Save arrays
filename_zero_four ='/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                    'Manufacturer_Calibrated/Zero_Four_Data_Processed.txt'
filename_four_eight = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                      'Manufacturer_Calibrated/Four_Eight_Data_Processed.txt'
filename_eight_twelve = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                        'Manufacturer_Calibrated/Eight_Twelve_Data_Processed.txt'
filename_twelve_sixteen = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                          'Manufacturer_Calibrated/Twelve_Sixteen_Data_Processed.txt'
filename_sixteen_twenty = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                          'Manufacturer_Calibrated/Sixteen_Twenty_Data_Processed.txt'
filename_twenty_twentyfour = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/' \
                             'Separated_Hours/Manufacturer_Calibrated/
                                 Twenty_Twentyfour_Data_Processed.txt'

#
    ##########################################################################################################################

# zero_four data
outputFile_zero_four = open(filename_zero_four, 'w')
outputFile_zero_four.write("#␣Date,␣Time,␣Latitude,␣Longitude␣and␣Temperature␣for␣each␣image
    ␣from␣00:00␣to␣03:59␣\n")
outputFile_zero_four.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_zero_four.write("#Recorded␣Time␣is␣Local␣Time␣(EDT)␣\n")
outputFile_zero_four.write("#NOTE:␣The␣column␣representing␣minutes␣may␣have␣a␣single␣digit␣
    such␣as␣0X␣where␣X"
                "␣is␣the␣number␣in␣the␣column␣so␣at␣the␣top␣of␣the␣hr␣only␣0␣would␣be␣
                    present␣\n")
outputFile_zero_four.write("#0:␣Picture␣File␣Name␣\t␣#1:Year␣\t␣␣#2:Month␣\t␣␣#3:Day␣\t␣#4:
    Hour␣\t␣#5:Minute␣\t"
                           "␣#6:␣Latitude␣\t␣#7:␣Longitude␣\t␣#8:␣X␣Pixel␣Coordinate␣\t␣#9:␣
                               Y␣Pixel␣Coordinate␣\t␣"
                           "#10:␣Temperature␣(K)␣(Emis␣!=␣1)␣\t␣#11:␣Temperature␣(C)␣(Emis␣
                               !=␣1)␣\n")
```

```
# Save data to file
for i in range(0, len_year):
    if zero_four_array[1][i] != 0:
        outputFile_zero_four.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i \t %i
            \t %f \t %f "
                                   "\n" % (zero_four_array[0][i], int(zero_four_array[1][i])
                                       ,
                                           int(zero_four_array[2][i]), int(zero_four_array
                                               [3][i]),
                                           int(zero_four_array[4][i]), int(zero_four_array
                                               [5][i]),
                                           zero_four_array[6][i], zero_four_array[7][i],
                                               int(zero_four_array[8][i]),
                                           int(zero_four_array[9][i]), zero_four_array[10][i
                                               ], zero_four_array[11][i]))
outputFile_zero_four.close()

#
    ##########################################################################################################################

# four eight data
outputFile_four_eight = open(filename_four_eight, 'w')
outputFile_four_eight.write("# Date, Time, Latitude, Longitude and Temperature for each
    image from 04:00 to 07:59 \n")
outputFile_four_eight.write("#By: Ryan Byerlay \n")
outputFile_four_eight.write("#Recorded Time is Local Time (EDT) \n")
outputFile_four_eight.write("#NOTE: The column representing minutes may have a single digit
    such as 0X where X"
                "is the number in the column so at the top of the hr only 0 would be
                    present \n")
outputFile_four_eight.write("#0: Picture File Name \t #1:Year \t\t #2:Month \t\t #3:Day \t #4:
    Hour \t #5:Minute"
                            "\t #6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t
                                #9: Y Pixel Coordinate"
                            "\t #10: Temperature (K) (Emis != 1) \t #11: Temperature (C) (
                                Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if four_eight_array[1][i] != 0:
        outputFile_four_eight.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i \t %
            i \t %f \t %f"
                                    "\n" % (four_eight_array[0][i], int(four_eight_array
                                        [1][i]),
                                            int(four_eight_array[2][i]), int(
                                                four_eight_array[3][i]),
                                            int(four_eight_array[4][i]), int(
                                                four_eight_array[5][i]),
                                            four_eight_array[6][i], four_eight_array[7][i],
                                            int(four_eight_array[8][i]), int(
                                                four_eight_array[9][i]),
```

```
                                                four_eight_array[10][i], four_eight_array[11][i
                                                    ]))
outputFile_four_eight.close()

#
    ##########################################################################################################################

# eight twelve data
outputFile_eight_twelve = open(filename_eight_twelve, 'w')
outputFile_eight_twelve.write("#␣Date,␣Time,␣Latitude,␣Longitude␣and␣Temperature␣for␣each␣
    image␣from␣08:00␣to␣11:59␣\n")
outputFile_eight_twelve.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_eight_twelve.write("#Recorded␣Time␣is␣Local␣Time␣(EDT)␣\n")
outputFile_eight_twelve.write("#NOTE:␣The␣column␣representing␣minutes␣may␣have␣a␣single␣
    digit␣such␣as␣0X␣where␣X"
                "␣is␣the␣number␣in␣the␣column␣so␣at␣the␣top␣of␣the␣hr␣only␣0␣would␣be␣
                    present\n")
outputFile_eight_twelve.write("#0:␣Picture␣File␣Name␣\t␣#1:Year␣\t␣␣#2:Month␣\t␣␣#3:Day␣\t␣
    #4:Hour␣\t␣#5:Minute"
                              "␣\t␣#6:␣Latitude␣\t␣#7:␣Longitude␣\t␣#8:␣X␣Pixel␣Coordinate␣\
                                  t␣#9:␣Y␣Pixel␣Coordinate"
                              "␣\t␣␣#10:␣Temperature␣(K)␣(Emis␣!=␣1)␣\t␣#11:␣Temperature␣(C)
                                  ␣(Emis␣!=␣1)␣\n")

# Save data to file
for i in range(0, len_year):
    if eight_twelve_array[1][i] != 0:
        outputFile_eight_twelve.write("%s␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%i␣\t␣%f␣\t␣%f␣\t␣%i␣\t
            ␣%i␣\t␣%f␣\t␣%f␣"
                                      "\n" % (eight_twelve_array[0][i], int(
                                          eight_twelve_array[1][i]),
                                              int(eight_twelve_array[2][i]), int(
                                                  eight_twelve_array[3][i]),
                                              int(eight_twelve_array[4][i]), int(
                                                  eight_twelve_array[5][i]),
                                              eight_twelve_array[6][i], eight_twelve_array
                                                  [7][i],
                                              int(eight_twelve_array[8][i]), int(
                                                  eight_twelve_array[9][i]),
                                              eight_twelve_array[10][i], eight_twelve_array
                                                  [11][i]))
outputFile_eight_twelve.close()

#
    ##########################################################################################################################

# twelve sixteen data
outputFile_twelve_sixteen = open(filename_twelve_sixteen, 'w')
outputFile_twelve_sixteen.write("#␣Date,␣Time,␣Latitude,␣Longitude␣and␣Temperature␣for␣each␣
    image␣from␣12:00␣to␣15:59"
                                "␣\n")
outputFile_twelve_sixteen.write("#By:␣Ryan␣Byerlay␣\n")
```

```
outputFile_twelve_sixteen.write("#Recorded Time is Local Time (EDT) \n")
outputFile_twelve_sixteen.write("#NOTE: The column represnting minutes may have a single
    digit such as 0X where X"
                "is the number in the column so at the top of the hr only 0 would be
                    present \n")
outputFile_twelve_sixteen.write("#0: Picture File Name \t #1:Year \t  #2:Month \t  #3:Day \t
    #4:Hour \t #5:Minute \t"
                                "#6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t
                                    #9: Y Pixel Coordinate"
                                "\t #10: Temperature (K) (Emis != 1) \t #11: Temperature (C
                                    ) (Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if twelve_sixteen_array[1][i] != 0:
        outputFile_twelve_sixteen.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i
            \t %i \t %f \t %f"
                                        "\n" % (twelve_sixteen_array[0][i], int(
                                            twelve_sixteen_array[1][i]),
                                                int(twelve_sixteen_array[2][i]), int(
                                                    twelve_sixteen_array[3][i]),
                                                int(twelve_sixteen_array[4][i]), int(
                                                    twelve_sixteen_array[5][i]),
                                                twelve_sixteen_array[6][i],
                                                    twelve_sixteen_array[7][i],
                                                int(twelve_sixteen_array[8][i]), int(
                                                    twelve_sixteen_array[9][i]),
                                                twelve_sixteen_array[10][i],
                                                    twelve_sixteen_array[11][i]))
outputFile_twelve_sixteen.close()

#
    ##########################################################################################################################################

# sixteen twenty data
outputFile_sixteen_twenty = open(filename_sixteen_twenty, 'w')
outputFile_sixteen_twenty.write("# Date, Time, Latitude, Longitude and Temperature for each
    image from 16:00 to 19:59"
                                "\n")
outputFile_sixteen_twenty.write("#By: Ryan Byerlay \n")
outputFile_sixteen_twenty.write("#Recorded Time is Local Time (EDT) \n")
outputFile_sixteen_twenty.write("#NOTE: The column representing minutes may have a single
    digit such as 0X where X"
                "is the number in the column so at the top of the hr only 0 would be
                    present \n")
outputFile_sixteen_twenty.write("#0: Picture File Name \t #1:Year \t  #2:Month \t  #3:Day \t
    #4:Hour \t #5:Minute \t"
                                "#6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate \t
                                    #9: Y Pixel Coordinate \t"
                                "#10: Temperature (K) (Emis != 1) \t #11: Temperature (C) (
                                    Emis != 1) \n")
```

```
# Save data to file
for i in range(0, len_year):
    if sixteen_twenty_array[1][i] != 0:
        outputFile_sixteen_twenty.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t %i
            \t %i \t %f \t %f"
                                        "\n" % (sixteen_twenty_array[0][i], int(
                                            sixteen_twenty_array[1][i]),
                                                int(sixteen_twenty_array[2][i]), int(
                                                    sixteen_twenty_array[3][i]),
                                                int(sixteen_twenty_array[4][i]), int(
                                                    sixteen_twenty_array[5][i]),
                                                sixteen_twenty_array[6][i],
                                                    sixteen_twenty_array[7][i],
                                                int(sixteen_twenty_array[8][i]), int(
                                                    sixteen_twenty_array[9][i]),
                                                sixteen_twenty_array[10][i],
                                                    sixteen_twenty_array[11][i]))
outputFile_sixteen_twenty.close()

#
    ##########################################################################################################################################

# twenty twenty-four data
outputFile_twenty_twentyfour = open(filename_twenty_twentyfour, 'w')
outputFile_twenty_twentyfour.write("# Date, Time, Latitude, Longitude and Temperature for
    each image from 20:00 to"
                                   " 23:59 \n")
outputFile_twenty_twentyfour.write("#By: Ryan Byerlay \n")
outputFile_twenty_twentyfour.write("#Recorded Time is Local Time (EDT) \n")
outputFile_twenty_twentyfour.write("#NOTE: The column representing minutes may have a single
    digit such as 0X where X"
                   " is the number in the column so at the top of the hr only 0 would be
                       present \n")
outputFile_twenty_twentyfour.write("#0: Picture File Name \t #1:Year \t  #2:Month \t  #3:Day
    \t #4:Hour \t #5:Minute \t"
                                   " #6: Latitude \t #7: Longitude \t #8: X Pixel Coordinate
                                       \t #9: Y Pixel Coordinate"
                                   " \t #10: Temperature (K) (Emis != 1) \t #11: Temperature
                                       (C) (Emis != 1) \n")

# Save data to file
for i in range(0, len_year):
    if twenty_twentyfour_array[1][i] != 0:
        outputFile_twenty_twentyfour.write("%s \t %i \t %i \t %i \t %i \t %i \t %f \t %f \t
            %i \t %i \t %f \t %f"
                                           "\n" % (twenty_twentyfour_array[0][i], int(
                                               twenty_twentyfour_array[1][i]),
                                                   int(twenty_twentyfour_array[2][i]),
                                                   int(twenty_twentyfour_array[3][i]),
                                                   int(twenty_twentyfour_array[4][i]),
                                                   int(twenty_twentyfour_array[5][i]),
                                                       twenty_twentyfour_array[6][i],
```

```
                                                    twenty_twentyfour_array[7][i], int(
                                                        twenty_twentyfour_array[8][i]),
                                                    int(twenty_twentyfour_array[9][i]),
                                                        twenty_twentyfour_array[10][i],
                                                    twenty_twentyfour_array[11][i]))
outputFile_twenty_twentyfour.close()
```

A.3.6 Surface Temperature Maps

```
# Current as of October 19, 2019
# Plot Guelph diurnal surface temperatures

import numpy
import sys
import os
import numpy
import matplotlib.pyplot as plt

# Declare calibration constants and original constants
# State original FLIR factory Planck constants
R1_flir = 17096.453
R2_flir = 0.046642166
R_flir = R1_flir/R2_flir
B_flir = 1428
O_flir =-342
F_flir =1

# Calibrated camera constants
# Grass constants
R_grass = 314531
B_grass = 1391
O_grass = -513
F_grass = 1.5

# Developed land (concrete) Constants
R_concrete = 247614
B_concrete = 1322
O_concrete = -513
F_concrete = 1.5

# Note: Images were not recorded during the 00:00-04:00 time interval
#
    ##########################################################################################################

# Load in data for 04:00 to 08:00
four_eight_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                      'Manufacturer_Calibrated/Four_Eight_Data_Processed.txt'

four_eight_data = numpy.genfromtxt(four_eight_filename)
four_eight_lon = four_eight_data[:,7]
```

```
four_eight_lat = four_eight_data[:,6]
four_eight_tempK = four_eight_data[:,10]

# Apply Temperature Correction Constants
# Calculate Upixel using the Horny, 2003 (https://doi.org/10.1016/S1350-4495(02)00183-4)
    formula
# Initialize Upixel
Upixel_four_eight = numpy.zeros((len(four_eight_lat)))

# Initialize new corrected temperature array
corrected_temp_four_eight = numpy.zeros((len(four_eight_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(four_eight_lat)):
    Upixel_four_eight[i] = (R_flir/(numpy.exp(B_flir/four_eight_tempK[i])-F_flir))-O_flir
    print(Upixel_four_eight)

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(four_eight_tempK)):
    # For Johnston Green
    if ((four_eight_lon[i] >= -80.2323044000  and four_eight_lon[i] <= -80.2276727000)
            and (four_eight_lat[i] >= 43.5302811700 and four_eight_lat[i] <= 43.5337142400))
                :
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South East Grass Area, corner of Gordon and Stone
    elif ((four_eight_lon[i] >= -80.2262640000 and four_eight_lon[i] <= -80.2241124000)
          and (four_eight_lat[i] >= 43.5268688200 and four_eight_lat[i] <= 43.5283545100)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South Residence Area
    elif ((four_eight_lon[i] >= -80.2240591000 and four_eight_lon[i] <= -80.2190782000)
          and (four_eight_lat[i] >= 43.5284211000 and four_eight_lat[i] <= 43.5320258200)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Gryphons Football Field
    elif ((four_eight_lon[i] >= -80.2274877882 and four_eight_lon[i] <= -80.2257549752)
          and (four_eight_lat[i] >= 43.5344304570 and four_eight_lat[i] <= 43.5357193383)):
        # Grass
        R = R_grass
```

```
        B = B_grass
        O = O_grass
        F = F_grass

    # Main baseball diamond
    elif ((four_eight_lon[i] >= -80.2258437615 and four_eight_lon[i] <= -80.2241568215)
          and (four_eight_lat[i] >= 43.5354322626 and four_eight_lat[i] <= 43.5367196642)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Baseball Diamonds
    elif ((four_eight_lon[i] >= -80.2186283934 and four_eight_lon[i] <= -80.2161897292)
          and (four_eight_lat[i] >= 43.5321175734 and four_eight_lat[i] <= 43.5338193111)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 1
    elif ((four_eight_lon[i] >= -80.2193490000 and four_eight_lon[i] <= -80.2172004000)
          and (four_eight_lat[i] >= 43.5329714000 and four_eight_lat[i] <= 43.5345399600)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 2
    elif ((four_eight_lon[i] >= -80.2210063870 and four_eight_lon[i] <= -80.2182125775)
          and (four_eight_lat[i] >= 43.5333383853 and four_eight_lat[i] <= 43.5352265743)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 3
    elif ((four_eight_lon[i] >= -80.2211869000 and four_eight_lon[i] <= -80.2197589000)
          and (four_eight_lat[i] >= 43.5322655500 and four_eight_lat[i] <= 43.5333013900)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 4
    elif ((four_eight_lon[i] >= -80.2211469653 and four_eight_lon[i] <= -80.2198270087)
          and (four_eight_lat[i] >= 43.5344970467 and four_eight_lat[i] <= 43.5355092107)):
```

```
            # Grass
            R = R_grass
            B = B_grass
            O = O_grass
            F = F_grass

        # Main Soccer Field
        elif ((four_eight_lon[i] >= -80.2256395530 and four_eight_lon[i] <= -80.2233103919)
              and (four_eight_lat[i] >= 43.5360685645 and four_eight_lat[i] <= 43.5378975627)):
            # Grass
            R = R_grass
            B = B_grass
            O = O_grass
            F = F_grass

        # East Residence Forest Area
        elif ((four_eight_lon[i] >= -80.2231505765 and four_eight_lon[i] <= -80.2200726509)
              and (four_eight_lat[i] >= 43.5352206552 and four_eight_lat[i] <= 43.5375823712)):
            # Grass
            R = R_grass
            B = B_grass
            O = O_grass
            F = F_grass

        # All other areas are considered to be developed land (concrete)
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Calculate corrected temperature
        corrected_temp_four_eight[i] = B / (numpy.log(R / (Upixel_four_eight[i] + O) + F))

    # Save To file
    outputFileName_four_eight = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
        Processed_Data/' \
                                'Separated_Hours/Campus_Calibrated/Four_eight_data_filtered.txt'
    outputFile_four_eight = open(outputFileName_four_eight, 'w')
    outputFile_four_eight.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected Temp
        [K] \n")

    # Save data to file
    for i in range(0, len(four_eight_lat)):
        outputFile_four_eight.write("%f \t %f \t %f \n" % (four_eight_lon[i], four_eight_lat[i],
                                                           corrected_temp_four_eight[i]))
    outputFile_four_eight.close()

    #
        ##########################################################################################################################
```

```
# Load in data for 08:00 to 12:00
eight_twelve_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                    'Manufacturer_Calibrated/Eight_Twelve_Data_Processed.txt'

eight_twelve_data = numpy.genfromtxt(eight_twelve_filename)
eight_twelve_lon = eight_twelve_data[:,7]
eight_twelve_lat = eight_twelve_data[:,6]
eight_twelve_tempK = eight_twelve_data[:,10]

# Apply Temperature Correction Constants
# Calculate Upixel/Uobject using the Horny, 2003 (https://doi.org/10.1016/S1350-4495(02)
    00183-4) formula
# Initialize Upixel
Upixel_eight_twelve = numpy.zeros((len(eight_twelve_lat)))

# Initialize new corrected temperature array
corrected_temp_eight_twelve = numpy.zeros((len(eight_twelve_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(eight_twelve_lat)):
    Upixel_eight_twelve[i] = (R_flir/(numpy.exp(B_flir/eight_twelve_tempK[i])-F_flir))-
        O_flir
    print(Upixel_eight_twelve)

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(eight_twelve_tempK)):
    # For Johnston Green
    if ((eight_twelve_lon[i] >= -80.2323044000  and eight_twelve_lon[i] <= -80.2276727000)
            and (eight_twelve_lat[i] >= 43.5302811700 and eight_twelve_lat[i] <=
                43.5337142400)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South East Grass Area, corner of Gordon and Stone
    elif ((eight_twelve_lon[i] >= -80.2262640000 and eight_twelve_lon[i] <= -80.2241124000)
          and (eight_twelve_lat[i] >= 43.5268688200 and eight_twelve_lat[i] <=
              43.5283545100)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South Residence Area
    elif ((eight_twelve_lon[i] >= -80.2240591000 and eight_twelve_lon[i] <= -80.2190782000)
          and (eight_twelve_lat[i] >= 43.5284211000 and eight_twelve_lat[i] <=
              43.5320258200)):
        # Grass
```

```
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Gryphons Football Field
    elif ((eight_twelve_lon[i] >= -80.2274877882 and eight_twelve_lon[i] <= -80.2257549752)
          and (eight_twelve_lat[i] >= 43.5344304570 and eight_twelve_lat[i] <=
              43.5357193383)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Main baseball diamond
    elif ((eight_twelve_lon[i] >= -80.2258437615 and eight_twelve_lon[i] <= -80.2241568215)
          and (eight_twelve_lat[i] >= 43.5354322626 and eight_twelve_lat[i] <=
              43.5367196642)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Baseball Diamonds
    elif ((eight_twelve_lon[i] >= -80.2186283934 and eight_twelve_lon[i] <= -80.2161897292)
          and (eight_twelve_lat[i] >= 43.5321175734 and eight_twelve_lat[i] <=
              43.5338193111)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 1
    elif ((eight_twelve_lon[i] >= -80.2193490000 and eight_twelve_lon[i] <= -80.2172004000)
          and (eight_twelve_lat[i] >= 43.5329714000 and eight_twelve_lat[i] <=
              43.5345399600)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 2
    elif ((eight_twelve_lon[i] >= -80.2210063870 and eight_twelve_lon[i] <= -80.2182125775)
          and (eight_twelve_lat[i] >= 43.5333383853 and eight_twelve_lat[i] <=
              43.5352265743)):
        # Grass
        R = R_grass
        B = B_grass
```

```
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 3
    elif ((eight_twelve_lon[i] >= -80.2211869000 and eight_twelve_lon[i] <= -80.2197589000)
          and (eight_twelve_lat[i] >= 43.5322655500 and eight_twelve_lat[i] <=
              43.5333013900)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 4
    elif ((eight_twelve_lon[i] >= -80.2211469653 and eight_twelve_lon[i] <= -80.2198270087)
          and (eight_twelve_lat[i] >= 43.5344970467 and eight_twelve_lat[i] <=
              43.5355092107)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Main Soccer Field
    elif ((eight_twelve_lon[i] >= -80.2256395530 and eight_twelve_lon[i] <= -80.2233103919)
          and (eight_twelve_lat[i] >= 43.5360685645 and eight_twelve_lat[i] <=
              43.5378975627)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # East Residence Forest Area
    elif ((eight_twelve_lon[i] >= -80.2231505765 and eight_twelve_lon[i] <= -80.2200726509)
        and (eight_twelve_lat[i] >= 43.5352206552 and eight_twelve_lat[i] <= 43.5375823712))
            :
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # All other areas are considered to be developed land (concrete)
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
```

```
    corrected_temp_eight_twelve[i] = B / (numpy.log(R / (Upixel_eight_twelve[i] + O) + F))

# Save To file
outputFileName_eight_twelve = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/' \
                              'Separated_Hours/Manufacturer_Calibrated/
                                  Eight_twelve_data_filtered.txt'
outputFile_eight_twelve = open(outputFileName_eight_twelve, 'w')
outputFile_eight_twelve.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(eight_twelve_lat)):
    outputFile_eight_twelve.write("%f \t %f \t %f \n" % (eight_twelve_lon[i],
        eight_twelve_lat[i],
                                                         corrected_temp_eight_twelve[i]))
outputFile_eight_twelve.close()

#
    ##########################################################################################################################

# Load in data for 12:00-16:00
twelve_sixteen_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/' \
                          'Separated_Hours/Manufacturer_Calibrated/
                              Twelve_Sixteen_Data_Processed.txt'

twelve_sixteen_data = numpy.genfromtxt(twelve_sixteen_filename)

twelve_sixteen_lon = twelve_sixteen_data[:,7]
twelve_sixteen_lat = twelve_sixteen_data[:,6]
twelve_sixteen_tempK = twelve_sixteen_data[:,10]

# Apply Temperature Correction Constants
# Calculate Upixel/Uobject using the Horny, 2003 (https://doi.org/10.1016/S1350-4495(02)
    00183-4) formula
# Initialize Upixel
Upixel_twelve_sixteen = numpy.zeros((len(twelve_sixteen_lat)))

# Initialize new corrected temperature array
corrected_temp_twelve_sixteen = numpy.zeros((len(twelve_sixteen_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(twelve_sixteen_lat)):
    Upixel_twelve_sixteen[i] = (R_flir/(numpy.exp(B_flir/twelve_sixteen_tempK[i])-F_flir))-
        O_flir
    print(Upixel_twelve_sixteen)

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(twelve_sixteen_tempK)):
    # For Johnston Green
    if ((twelve_sixteen_lon[i] >= -80.2323044000  and twelve_sixteen_lon[i] <=
```

```
    -80.2276727000)
        and (twelve_sixteen_lat[i] >= 43.5302811700 and twelve_sixteen_lat[i] <=
            43.5337142400)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# South East Grass Area, corner of Gordon and Stone
elif ((twelve_sixteen_lon[i] >= -80.2262640000 and twelve_sixteen_lon[i] <=
    -80.2241124000)
      and (twelve_sixteen_lat[i] >= 43.5268688200 and twelve_sixteen_lat[i] <=
          43.5283545100)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# South Residence Area
elif ((twelve_sixteen_lon[i] >= -80.2240591000 and twelve_sixteen_lon[i] <=
    -80.2190782000)
      and (twelve_sixteen_lat[i] >= 43.5284211000 and twelve_sixteen_lat[i] <=
          43.5320258200)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Gryphons Football Field
elif ((twelve_sixteen_lon[i] >= -80.2274877882 and twelve_sixteen_lon[i] <=
    -80.2257549752)
      and (twelve_sixteen_lat[i] >= 43.5344304570 and twelve_sixteen_lat[i] <=
          43.5357193383)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Main baseball diamond
elif ((twelve_sixteen_lon[i] >= -80.2258437615 and twelve_sixteen_lon[i] <=
    -80.2241568215)
      and (twelve_sixteen_lat[i] >= 43.5354322626 and twelve_sixteen_lat[i] <=
          43.5367196642)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass
```

```
# Stone Rd Baseball Diamonds
elif ((twelve_sixteen_lon[i] >= -80.2186283934 and twelve_sixteen_lon[i] <=
    -80.2161897292)
      and (twelve_sixteen_lat[i] >= 43.5321175734 and twelve_sixteen_lat[i] <=
          43.5338193111)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Rd Soccer Field 1
elif ((twelve_sixteen_lon[i] >= -80.2193490000 and twelve_sixteen_lon[i] <=
    -80.2172004000)
      and (twelve_sixteen_lat[i] >= 43.5329714000 and twelve_sixteen_lat[i] <=
          43.5345399600)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Rd Soccer Field 2
elif ((twelve_sixteen_lon[i] >= -80.2210063870 and twelve_sixteen_lon[i] <=
    -80.2182125775)
      and (twelve_sixteen_lat[i] >= 43.5333383853 and twelve_sixteen_lat[i] <=
          43.5352265743)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Road Soccer Field 3
elif ((twelve_sixteen_lon[i] >= -80.2211869000 and twelve_sixteen_lon[i] <=
    -80.2197589000)
      and (twelve_sixteen_lat[i] >= 43.5322655500 and twelve_sixteen_lat[i] <=
          43.5333013900)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Road Soccer Field 4
elif ((twelve_sixteen_lon[i] >= -80.2211469653 and twelve_sixteen_lon[i] <=
    -80.2198270087)
      and (twelve_sixteen_lat[i] >= 43.5344970467 and twelve_sixteen_lat[i] <=
          43.5355092107)):
    # Grass
    R = R_grass
```

```
            B = B_grass
            O = O_grass
            F = F_grass

        # Main Soccer Field
        elif ((twelve_sixteen_lon[i] >= -80.2256395530 and twelve_sixteen_lon[i] <=
            -80.2233103919)
              and (twelve_sixteen_lat[i] >= 43.5360685645 and twelve_sixteen_lat[i] <=
                  43.5378975627)):
            # Grass
            R = R_grass
            B = B_grass
            O = O_grass
            F = F_grass

        # East Residence Forest Area
        elif ((twelve_sixteen_lon[i] >= -80.2231505765 and twelve_sixteen_lon[i] <=
            -80.2200726509)
              and (twelve_sixteen_lat[i] >= 43.5352206552 and twelve_sixteen_lat[i] <=
                  43.5375823712)):
            # Grass
            R = R_grass
            B = B_grass
            O = O_grass
            F = F_grass

        # All other areas are considered to be developed land (concrete)
        else:
            # Developed land (concrete)
            R = R_concrete
            B = B_concrete
            O = O_concrete
            F = F_concrete

        # Calculate corrected temperature
        corrected_temp_twelve_sixteen[i] = B / (numpy.log(R / (Upixel_twelve_sixteen[i] + O) + F
            ))

    # Save To file
    outputFileName_twelve_sixteen = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
        Processed_Data/' \
                                    'Separated_Hours/Manufacturer_Calibrated/
                                        Twelve_sixteen_data_filtered.txt'
    outputFile_twelve_sixteen = open(outputFileName_twelve_sixteen, 'w')
    outputFile_twelve_sixteen.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
        Temp [K] \n")

    # Save data to file
    for i in range(0, len(twelve_sixteen_lat)):
        outputFile_twelve_sixteen.write("%f \t %f \t %f \n" % (twelve_sixteen_lon[i],
            twelve_sixteen_lat[i],
                                                               corrected_temp_twelve_sixteen[i])
```

```
                                                                    )
outputFile_twelve_sixteen.close()

#
    ##########################################################################################################################

# Load in data for 16:00 to 20:00
sixteen_twenty_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/Separated_Hours/' \
                      'Manufacturer_Calibrated/Sixteen_Twenty_Data_Processed.txt'

sixteen_twenty_data = numpy.genfromtxt(sixteen_twenty_filename)

sixteen_twenty_lon = sixteen_twenty_data[:,7]
sixteen_twenty_lat = sixteen_twenty_data[:,6]
sixteen_twenty_tempK = sixteen_twenty_data[:,10]

# Apply Temperature Correction Constants
# Calculate Upixel/Uobject using the Horny, 2003 (https://doi.org/10.1016/S1350-4495(02)
    00183-4) formula
# Initialize Upixel
Upixel_sixteen_twenty = numpy.zeros((len(sixteen_twenty_lat)))

# Initialize new corrected temperature array
corrected_temp_sixteen_twenty = numpy.zeros((len(sixteen_twenty_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(sixteen_twenty_lat)):
    Upixel_sixteen_twenty[i] = (R_flir/(numpy.exp(B_flir/sixteen_twenty_tempK[i])-F_flir))-
        O_flir
    print(Upixel_sixteen_twenty)

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(sixteen_twenty_tempK)):
    # For Johnston Green
    if ((sixteen_twenty_lon[i] >= -80.2323044000  and sixteen_twenty_lon[i] <=
        -80.2276727000)
            and (sixteen_twenty_lat[i] >= 43.5302811700 and sixteen_twenty_lat[i] <=
                43.5337142400)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South East Grass Area, corner of Gordon and Stone
    elif ((sixteen_twenty_lon[i] >= -80.2262640000 and sixteen_twenty_lon[i] <=
        -80.2241124000)
          and (sixteen_twenty_lat[i] >= 43.5268688200 and sixteen_twenty_lat[i] <=
              43.5283545100)):
        # Grass
        R = R_grass
```

```
        B = B_grass
        O = O_grass
        F = F_grass

    # South Residence Area
    elif ((sixteen_twenty_lon[i] >= -80.2240591000 and sixteen_twenty_lon[i] <=
        -80.2190782000)
          and (sixteen_twenty_lat[i] >= 43.5284211000 and sixteen_twenty_lat[i] <=
              43.5320258200)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Gryphons Football Field
    elif ((sixteen_twenty_lon[i] >= -80.2274877882 and sixteen_twenty_lon[i] <=
        -80.2257549752)
          and (sixteen_twenty_lat[i] >= 43.5344304570 and sixteen_twenty_lat[i] <=
              43.5357193383)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Main baseball diamond
    elif ((sixteen_twenty_lon[i] >= -80.2258437615 and sixteen_twenty_lon[i] <=
        -80.2241568215)
          and (sixteen_twenty_lat[i] >= 43.5354322626 and sixteen_twenty_lat[i] <=
              43.5367196642)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Baseball Diamonds
    elif ((sixteen_twenty_lon[i] >= -80.2186283934 and sixteen_twenty_lon[i] <=
        -80.2161897292)
          and (sixteen_twenty_lat[i] >= 43.5321175734 and sixteen_twenty_lat[i] <=
              43.5338193111)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 1
    elif ((sixteen_twenty_lon[i] >= -80.2193490000 and sixteen_twenty_lon[i] <=
        -80.2172004000)
          and (sixteen_twenty_lat[i] >= 43.5329714000 and sixteen_twenty_lat[i] <=
```

```
            43.5345399600)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Rd Soccer Field 2
    elif ((sixteen_twenty_lon[i] >= -80.2210063870 and sixteen_twenty_lon[i] <=
        -80.2182125775)
          and (sixteen_twenty_lat[i] >= 43.5333383853 and sixteen_twenty_lat[i] <=
              43.5352265743)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 3
    elif ((sixteen_twenty_lon[i] >= -80.2211869000 and sixteen_twenty_lon[i] <=
        -80.2197589000)
          and (sixteen_twenty_lat[i] >= 43.5322655500 and sixteen_twenty_lat[i] <=
              43.5333013900)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 4
    elif ((sixteen_twenty_lon[i] >= -80.2211469653 and sixteen_twenty_lon[i] <=
        -80.2198270087)
          and (sixteen_twenty_lat[i] >= 43.5344970467 and sixteen_twenty_lat[i] <=
              43.5355092107)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Main Soccer Field
    elif ((sixteen_twenty_lon[i] >= -80.2256395530 and sixteen_twenty_lon[i] <=
        -80.2233103919)
          and (sixteen_twenty_lat[i] >= 43.5360685645 and sixteen_twenty_lat[i] <=
              43.5378975627)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # East Residence Forest Area
```

```
    elif ((sixteen_twenty_lon[i] >= -80.2231505765 and sixteen_twenty_lon[i] <=
        -80.2200726509)
          and (sixteen_twenty_lat[i] >= 43.5352206552 and sixteen_twenty_lat[i] <=
              43.5375823712)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # All other areas are considered to be developed land (concrete)
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
        O = O_concrete
        F = F_concrete

    # Calculate corrected temperature
    corrected_temp_sixteen_twenty[i] = B / (numpy.log(R / (Upixel_sixteen_twenty[i] + O) + F
        ))

# Save To file
outputFileName_sixteen_twenty = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/' \
                                'Separated_Hours/Manufacturer_Calibrated/
                                    Sixteen_twenty_data_filtered.txt'
outputFile_sixteen_twenty = open(outputFileName_sixteen_twenty, 'w')
outputFile_sixteen_twenty.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use Corrected
    Temp [K] \n")

# Save data to file
for i in range(0, len(sixteen_twenty_lat)):
    outputFile_sixteen_twenty.write("%f \t %f \t %f \n" % (sixteen_twenty_lon[i],
        sixteen_twenty_lat[i],
                                                           corrected_temp_sixteen_twenty[i])
                                                               )
outputFile_sixteen_twenty.close()

#
    ##########################################################################################################################

# Load in data for 20:00 to 24:00
twenty_twentyfour_filename = '/export/home/users/username/Documents/DG_Temp/Guelph_2018/
    Processed_Data/' \
                             'Separated_Hours/Manufacturer_Calibrated/
                                 Twenty_Twentyfour_Data_Processed.txt'

twenty_twentyfour_data = numpy.genfromtxt(twenty_twentyfour_filename)

twenty_twentyfour_lon = twenty_twentyfour_data[:,7]
twenty_twentyfour_lat = twenty_twentyfour_data[:,6]
```

```
twenty_twentyfour_tempK = twenty_twentyfour_data[:,10]

# Apply Temperature Correction Constants
# Calculate Upixel/Uobject using the Horny, 2003 (https://doi.org/10.1016/S1350-4495(02)
    00183-4) formula
# Initialize Upixel
Upixel_twenty_twentyfour = numpy.zeros((len(twenty_twentyfour_lat)))

# Initialize new corrected temperature array
corrected_temp_twenty_twentyfour = numpy.zeros((len(twenty_twentyfour_lon)))

# Calculate Pixel coordinate A/D Counts for each latitude/longitude pair
for i in range(0, len(twenty_twentyfour_lat)):
    Upixel_twenty_twentyfour[i] = (R_flir/(numpy.exp(B_flir/twenty_twentyfour_tempK[i])-
        F_flir))-O_flir
    print(Upixel_twenty_twentyfour)

# Apply land use camera parameters filter and calculate corrected temperature
for i in range(0, len(twenty_twentyfour_tempK)):
    # For Johnston Green
    if ((twenty_twentyfour_lon[i] >= -80.2323044000  and twenty_twentyfour_lon[i] <=
        -80.2276727000)
            and (twenty_twentyfour_lat[i] >= 43.5302811700 and twenty_twentyfour_lat[i] <=
                43.5337142400)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South East Grass Area, corner of Gordon and Stone
    elif ((twenty_twentyfour_lon[i] >= -80.2262640000 and twenty_twentyfour_lon[i] <=
        -80.2241124000)
          and (twenty_twentyfour_lat[i] >= 43.5268688200 and twenty_twentyfour_lat[i] <=
              43.5283545100)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # South Residence Area
    elif ((twenty_twentyfour_lon[i] >= -80.2240591000 and twenty_twentyfour_lon[i] <=
        -80.2190782000)
          and (twenty_twentyfour_lat[i] >= 43.5284211000 and twenty_twentyfour_lat[i] <=
              43.5320258200)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass
```

```
# Gryphons Football Field
elif ((twenty_twentyfour_lon[i] >= -80.2274877882 and twenty_twentyfour_lon[i] <=
    -80.2257549752)
      and (twenty_twentyfour_lat[i] >= 43.5344304570 and twenty_twentyfour_lat[i] <=
          43.5357193383)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Main baseball diamond
elif ((twenty_twentyfour_lon[i] >= -80.2258437615 and twenty_twentyfour_lon[i] <=
    -80.2241568215)
      and (twenty_twentyfour_lat[i] >= 43.5354322626 and twenty_twentyfour_lat[i] <=
          43.5367196642)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Rd Baseball Diamonds
elif ((twenty_twentyfour_lon[i] >= -80.2186283934 and twenty_twentyfour_lon[i] <=
    -80.2161897292)
      and (twenty_twentyfour_lat[i] >= 43.5321175734 and twenty_twentyfour_lat[i] <=
          43.5338193111)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Rd Soccer Field 1
elif ((twenty_twentyfour_lon[i] >= -80.2193490000 and twenty_twentyfour_lon[i] <=
    -80.2172004000)
      and (twenty_twentyfour_lat[i] >= 43.5329714000 and twenty_twentyfour_lat[i] <=
          43.5345399600)):
    # Grass
    R = R_grass
    B = B_grass
    O = O_grass
    F = F_grass

# Stone Rd Soccer Field 2
elif ((twenty_twentyfour_lon[i] >= -80.2210063870 and twenty_twentyfour_lon[i] <=
    -80.2182125775)
      and (twenty_twentyfour_lat[i] >= 43.5333383853 and twenty_twentyfour_lat[i] <=
          43.5352265743)):
    # Grass
    R = R_grass
    B = B_grass
```

```
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 3
    elif ((twenty_twentyfour_lon[i] >= -80.2211869000 and twenty_twentyfour_lon[i] <=
        -80.2197589000)
          and (twenty_twentyfour_lat[i] >= 43.5322655500 and twenty_twentyfour_lat[i] <=
              43.5333013900)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Stone Road Soccer Field 4
    elif ((twenty_twentyfour_lon[i] >= -80.2211469653 and twenty_twentyfour_lon[i] <=
        -80.2198270087)
          and (twenty_twentyfour_lat[i] >= 43.5344970467 and twenty_twentyfour_lat[i] <=
              43.5355092107)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # Main Soccer Field
    elif ((twenty_twentyfour_lon[i] >= -80.2256395530 and twenty_twentyfour_lon[i] <=
        -80.2233103919)
          and (twenty_twentyfour_lat[i] >= 43.5360685645 and twenty_twentyfour_lat[i] <=
              43.5378975627)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # East Residence Forest Area
    elif ((twenty_twentyfour_lon[i] >= -80.2231505765 and twenty_twentyfour_lon[i] <=
        -80.2200726509)
          and (twenty_twentyfour_lat[i] >= 43.5352206552 and twenty_twentyfour_lat[i] <=
              43.5375823712)):
        # Grass
        R = R_grass
        B = B_grass
        O = O_grass
        F = F_grass

    # All other areas are considered to be developed land (concrete)
    else:
        # Developed land (concrete)
        R = R_concrete
        B = B_concrete
```

```
        O = O_concrete
        F = F_concrete


    # Calculate corrected temperature
    corrected_temp_twenty_twentyfour[i] = B / (numpy.log(R / (Upixel_twenty_twentyfour[i] +
        O) + F))

# Save To file
outputFileName_twenty_twentyfour = '/export/home/users/username/Documents/DG_Temp/
    Guelph_2018/Processed_Data/' \
                                   'Separated_Hours/Manufacturer_Calibrated/
                                       Twenty_twentyfour_data_filtered.txt'
outputFile_twenty_twentyfour = open(outputFileName_twenty_twentyfour, 'w')
outputFile_twenty_twentyfour.write("#0: Longitude \t #1:Latitude \t #2:FLIR Land Use
    Corrected Temp [K] \n")

# Save data to file
for i in range(0, len(twenty_twentyfour_lat)):
    outputFile_twenty_twentyfour.write("%f \t %f \t %f \n" % (twenty_twentyfour_lon[i],
        twenty_twentyfour_lat[i],
                                                              corrected_temp_twenty_twentyfour
                                                                  [i]))
outputFile_twenty_twentyfour.close()

#
    ##########################################################################################################################

# Temperature distribution boundaries
# Guelph Summer 2018 Site Boundaries as of Sept 23/2019
Latmin = 43.5257257364
Latmax = 43.5385583188

Lonmax = -80.2324729223
Lonmin = -80.2150621430

# # For 20 m resolution, maximum/minimum latitude and longitude = 1 km area
nLat = 50
nLon = 50

# # For 50 m resolution, maximum/minimum latitude and longitude = 1 km area
# nLat = 20
# nLon = 20
#
    ##########################################################################################################################

# Crete temperature array for each interval
# 04:00 to 08:00
TMatrix_four_eight = numpy.zeros(((nLat+1), (nLon+1), (len(four_eight_lat))))
TMatrix_four_eight[:] = numpy.nan

# 08:00 to 12:00
```

```
TMatrix_eight_twelve = numpy.zeros(((nLat+1), (nLon+1), (len(eight_twelve_lat))))
TMatrix_eight_twelve[:] = numpy.nan

# 12:00 to 16:00
TMatrix_twelve_sixteen = numpy.zeros(((nLat+1), (nLon+1), (len(twelve_sixteen_lat))))
TMatrix_twelve_sixteen[:] = numpy.nan

# 16:00 to 20:00
TMatrix_sixteen_twenty = numpy.zeros(((nLat+1), (nLon+1), (len(sixteen_twenty_lat))))
TMatrix_sixteen_twenty[:] = numpy.nan

# 20:00 to 24:00
TMatrix_twenty_twentyfour = numpy.zeros(((nLat+1), (nLon+1), (len(twenty_twentyfour_lat))))
TMatrix_twenty_twentyfour[:] = numpy.nan

#
    ##########################################################################################################################

# Create median temperature array for each interval
# For 04:00 to 08:00
TMatrix_four_eight_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_four_eight_median[:] = numpy.nan

# For 08:00 to 12:00
TMatrix_eight_twelve_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_eight_twelve_median[:] = numpy.nan

# For 12:00 to 16:00
TMatrix_twelve_sixteen_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_twelve_sixteen_median[:] = numpy.nan

# For 16:00 to 20:00
TMatrix_sixteen_twenty_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_sixteen_twenty_median[:] = numpy.nan

# For 20:00 to 24:00
TMatrix_twenty_twentyfour_median = numpy.zeros(((nLat+1), (nLon+1)))
TMatrix_twenty_twentyfour_median[:] = numpy.nan

#
    ##########################################################################################################################

# From 04:00 to 08:00
for i in range(0, len(four_eight_lat)):
    if numpy.isnan(four_eight_lat[i]) == False or numpy.isnan(four_eight_lon[i]) == False:
        LatIndex_four_eight = int((four_eight_lat[i] - Latmin) * nLat / (Latmax - Latmin))
        LonIndex_four_eight = int((four_eight_lon[i] - Lonmin) * nLon / (Lonmax - Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if four_eight_lat[i] > Latmax or four_eight_lat[i] < Latmin or four_eight_lon[i] <
```

```
            Lonmax\
                or four_eight_lon[i] > Lonmin:
            continue
        else:
            TMatrix_four_eight[LatIndex_four_eight][LonIndex_four_eight][i] =
                corrected_temp_four_eight[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(four_eight_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_four_eight[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be = to Nan
        if (k == len(four_eight_lat)) & (TMatrix_four_eight[i][j][k] == numpy.nan):
            TMatrix_four_eight_median[i][j] = numpy.nan
        else:
            TMatrix_four_eight_median[i][j] = numpy.nanpercentile(TMatrix_four_eight[i][j
                ][:],50)

#
    ##########################################################################################################################

# From 08:00 to 12:00
for i in range(0, len(eight_twelve_lat)):
    if numpy.isnan(eight_twelve_lat[i]) == False or numpy.isnan(eight_twelve_lon[i]) ==
        False:
        LatIndex_eight_twelve = int((eight_twelve_lat[i] - Latmin) * nLat / (Latmax - Latmin
            ))
        LonIndex_eight_twelve = int((eight_twelve_lon[i] - Lonmin) * nLon / (Lonmax - Lonmin
            ))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if eight_twelve_lat[i] > Latmax or eight_twelve_lat[i] < Latmin or eight_twelve_lon[
            i] < Lonmax\
                or eight_twelve_lon[i] > Lonmin:
            continue

        else:
            TMatrix_eight_twelve[LatIndex_eight_twelve][LonIndex_eight_twelve][i] =
                corrected_temp_eight_twelve[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check if Nan
        for k in range(0, len(eight_twelve_lat)):
            # If a real number is encountered, a median can be calculated
```

```
            if TMatrix_eight_twelve[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be = to Nan
        if (k == len(eight_twelve_lat)) & (TMatrix_eight_twelve[i][j][k] == numpy.nan):
            TMatrix_eight_twelve_median[i][j] = numpy.nan
        else:
            TMatrix_eight_twelve_median[i][j] = numpy.nanpercentile(TMatrix_eight_twelve[i][
                j][:],50)

#
    ##########################################################################################################

# From 12:00 to 16:00
for i in range(0, len(twelve_sixteen_lat)):
    if numpy.isnan(twelve_sixteen_lat[i]) == False or numpy.isnan(twelve_sixteen_lon[i]) ==
        False:
        LatIndex_twelve_sixteen = int((twelve_sixteen_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_twelve_sixteen = int((twelve_sixteen_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if twelve_sixteen_lat[i] > Latmax or twelve_sixteen_lat[i] < Latmin or
            twelve_sixteen_lon[i] < Lonmax\
                or twelve_sixteen_lon[i] > Lonmin:
            continue

        else:
            TMatrix_twelve_sixteen[LatIndex_twelve_sixteen][LonIndex_twelve_sixteen][i] \
                = corrected_temp_twelve_sixteen[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check if Nan
        for k in range(0, len(twelve_sixteen_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_twelve_sixteen[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be = to Nan
        if (k == len(twelve_sixteen_lat)) & (TMatrix_twelve_sixteen[i][j][k] == numpy.nan):
            TMatrix_twelve_sixteen_median[i][j] = numpy.nan
        else:
            TMatrix_twelve_sixteen_median[i][j] = numpy.nanpercentile(TMatrix_twelve_sixteen
                [i][j][:],50)

#
    ##########################################################################################################

# From 16:00 to 20:00
```

```
for i in range(0, len(sixteen_twenty_lat)):
    if numpy.isnan(sixteen_twenty_lat[i]) == False or numpy.isnan(sixteen_twenty_lon[i]) ==
        False:
        LatIndex_sixteen_twenty = int((sixteen_twenty_lat[i] - Latmin) * nLat / (Latmax -
            Latmin))
        LonIndex_sixteen_twenty = int((sixteen_twenty_lon[i] - Lonmin) * nLon / (Lonmax -
            Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if sixteen_twenty_lat[i] > Latmax or sixteen_twenty_lat[i] < Latmin or
            sixteen_twenty_lon[i] < Lonmax\
                or sixteen_twenty_lon[i] > Lonmin:
            continue

        else:
            TMatrix_sixteen_twenty[LatIndex_sixteen_twenty][LonIndex_sixteen_twenty][i]\
                = corrected_temp_sixteen_twenty[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check for Nan
        for k in range(0, len(sixteen_twenty_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_sixteen_twenty[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be = to Nan
        if (k == len(sixteen_twenty_lat)) & (TMatrix_sixteen_twenty[i][j][k] == numpy.nan):
            TMatrix_sixteen_twenty_median[i][j] = numpy.nan
        else:
            TMatrix_sixteen_twenty_median[i][j] = numpy.nanpercentile(TMatrix_sixteen_twenty
                [i][j][:],50)

#
    ##########################################################################################################################

# From 20:00 to 24:00
for i in range(0, len(twenty_twentyfour_lat)):
    if numpy.isnan(twenty_twentyfour_lat[i]) == False or numpy.isnan(twenty_twentyfour_lon[i
        ]) == False:
        LatIndex_twenty_twentyfour = int((twenty_twentyfour_lat[i] - Latmin) * nLat / (
            Latmax - Latmin))
        LonIndex_twenty_twentyfour = int((twenty_twentyfour_lon[i] - Lonmin) * nLon / (
            Lonmax - Lonmin))

        # Ignore latitude/longitude values greater than the latitude/longitude maximum or
            less than the
        #  latitude/longitude minimum
        if twenty_twentyfour_lat[i] > Latmax or twenty_twentyfour_lat[i] < Latmin or
            twenty_twentyfour_lon[i] < Lonmax\
```

```
                or twenty_twentyfour_lon[i] > Lonmin:
            continue
        else:
            TMatrix_twenty_twentyfour[LatIndex_twenty_twentyfour][LonIndex_twenty_twentyfour
                ][i]\
                = corrected_temp_twenty_twentyfour[i]

# Calculate the median temperature for each bin
for i in range(0, nLat+1):
    for j in range(0, nLon+1):
        # Check if Nan
        for k in range(0, len(twenty_twentyfour_lat)):
            # If a real number is encountered, a median can be calculated
            if TMatrix_twenty_twentyfour[i][j][k] != numpy.nan:
                break
        # If at the last index and it is a Nan, assign TMatrix to be = to Nan
        if (k == len(twenty_twentyfour_lat)) & (TMatrix_twenty_twentyfour[i][j][k] == numpy.
            nan):
            TMatrix_twenty_twentyfour_median[i][j] = numpy.nan
        else:
            TMatrix_twenty_twentyfour_median[i][j] = numpy.nanpercentile(
                TMatrix_twenty_twentyfour[i][j][:],50)

#
    ##########################################################################################################################################

# State minimum and maximum colour bar ranges for each respective time interval
# 4-8
color_bar_min_four_eight = 280
color_bar_max_four_eight = 295

# 8-12
color_bar_min_eight_twelve = 285
color_bar_max_eight_twelve = 315

# 12-16
color_bar_min_twelve_sixteen = 285
color_bar_max_twelve_sixteen = 315

# 16-20
color_bar_min_sixteen_twenty = 290
color_bar_max_sixteen_twenty = 315

# 20-24
color_bar_min_twenty_twentyfour = 280
color_bar_max_twenty_twentyfour = 300

# May 24 surface temperature colour bar range
color_bar_min_May_24_SA = 295
color_bar_max_May_24_SA = 325

# Figure size
```

```
figuresize = (10,6)

# Font size
font_size = 16
title_font_size = 16

tick_size = 11
cbar_tick_size = 16

# Figure parameters
# For surface temperature maps
axes_label_fontsize = 36
axes_ticks_fontsize = 34

# For boxplots
axes_bxplt_label_fontsize = 42
axes_bxplt_ticks_fontsize = 40

# For colour bars
axes_clrbar_label_fontsize = 32
axes_clrbar_ticks_fontsize = 30

# Position of x and y labels away from respective axes in points
x_labelpad = -5
y_labelpad = -5

# Geographic identifier marker size
geog_ident_size = 175

#
    ##########################################################################################################################

# Filename resolution
# 20m and 50m resolution
if nLat == 50:
    filename_res = '20m'
elif nLat == 20:
    filename_res = '50m'
else:
    print('You have problems with the resolution size in the outputted filename')

#
    ##########################################################################################################################

# Declare TANAB2 launch location at Reek Walk
base_lon = [-80.2253889000]
base_lat = [43.5323355400]

# Declare Known geographical locations:
arena_lon = [-80.2235723116]
arena_lat = [43.5323454583]
```

```
uc_lon = [-80.2263779592]
uc_lat = [43.5305549342]

athletics_lon = [-80.2244601747]
athletics_lat = [43.5336269408]

football_field_lon = [-80.2266413586]
football_field_lat = [43.5350800769]

johnston_green_lon = [-80.2299619669]
johnston_green_lat = [43.5320435848]

fieldhouse_lon = [-80.2251245923]
fieldhouse_lat = [43.5343054162]

# colour bar label
color_bar_label = '$T$ [K]'

# TANAB2_dot size
launch_size = 25

# Use latex font for labels
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

# Directory to save images
direct_save ='/export/home/users/username/Documents/DG_Temp/Guelph_2018/' \
             'Processed_Data/Figures/'

#
    ##########################################################################################################################

# At 04:00 to 08:00
Lataxis_four_eight = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_four_eight = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_four_eight,LatAxis_four_eight = numpy.meshgrid(Lonaxis_four_eight,Lataxis_four_eight
    )

fig_four_eight, ax = plt.subplots(figsize=figuresize)
Tpcolor_four_eight=plt.pcolor(LonAxis_four_eight,LatAxis_four_eight,
    TMatrix_four_eight_median,
                              vmin=color_bar_min_four_eight, vmax=color_bar_max_four_eight)
cbar_four_eight = plt.colorbar(Tpcolor_four_eight)
cbar_four_eight.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_four_eight.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(arena_lon, arena_lat, edgecolors='k', facecolors='none', s=geog_ident_size)
plt.scatter(uc_lon, uc_lat, edgecolors='m', facecolors='none', s=geog_ident_size)
plt.scatter(athletics_lon, athletics_lat, edgecolors='b', facecolors='none', s=
    geog_ident_size)
plt.scatter(football_field_lon, football_field_lat, edgecolors='y', facecolors='none', s=
    geog_ident_size)
```

```
plt.scatter(johnston_green_lon, johnston_green_lat, edgecolors='c', facecolors ='none', s=
    geog_ident_size)
plt.scatter(fieldhouse_lon, fieldhouse_lat, edgecolors='w', facecolors='none', s=
    geog_ident_size)
plt.scatter(base_lon, base_lat, c='r',s=launch_size)
# Verified distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xlabel('Decimal Degrees [deg]',fontsize=12)
plt.ylabel('Decimal Degrees [deg]',fontsize=12)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_four_eight.show()
plt.savefig(direct_save+'0400_0800_map_'+filename_res+'.png')
plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_four_eight = numpy.zeros((nLat, 1))
LonAxis_median_four_eight = numpy.zeros((nLon, 1))
LatAxisIndex_four_eight = numpy.empty((nLat+1,1))
LatAxisIndex_four_eight[:] = numpy.nan
LonAxisIndex_four_eight = numpy.empty((nLon+1,1))
LonAxisIndex_four_eight[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_four_eight[a] = ((Lataxis_four_eight[a])+(Lataxis_four_eight[a+1]))/2

for j in range(0, nLon):
    LonAxis_median_four_eight[j] = (Lonaxis_four_eight[j]+Lonaxis_four_eight[j+1])/2

# Save latitude/longitude indices and median temperatures
output_four_eight_filename = direct_save+'Figure_Data/Four_Eight_MedianTemp_'+filename_res+'
    .txt'
outputFile_four_eight = open(output_four_eight_filename, 'w')
outputFile_four_eight.write("# Lat, Lon indices, median temp for UofG Campus \n")
outputFile_four_eight.write("#By: Ryan Byerlay \n")
outputFile_four_eight.write("#Recorded Time is Local Time (EDT) \n")
outputFile_four_eight.write("#0:LatAxis_four_eight \t #1:LonAxis_four_eight"
                            " \t #2:MedianTemp_four_eight(K) {lat ,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_four_eight)-1):
    for j in range(0, len(LonAxis_four_eight)-1):
        print(TMatrix_four_eight_median[i][j])
        if numpy.isnan(TMatrix_four_eight_median[i][j]) == False:
            outputFile_four_eight.write("%f \t %f \t %f \n" % (LatAxis_median_four_eight[i],
                                                               LonAxis_median_four_eight[j],
                                                               TMatrix_four_eight_median[i][
                                                                   j]))
outputFile_four_eight.close()

#
    ##########################################################################################################################
```

```
# At 08:00 to 12:00
Lataxis_eight_twelve = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_eight_twelve = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_eight_twelve,LatAxis_eight_twelve = numpy.meshgrid(Lonaxis_eight_twelve,
    Lataxis_eight_twelve)

fig_eight_twelve, ax = plt.subplots(figsize=figuresize)
Tpcolor_eight_twelve=plt.pcolor(LonAxis_eight_twelve,LatAxis_eight_twelve,
    TMatrix_eight_twelve_median,
                                vmin=color_bar_min_eight_twelve, vmax=
                                    color_bar_max_eight_twelve)
cbar_eight_twelve = plt.colorbar(Tpcolor_eight_twelve)
cbar_eight_twelve.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_eight_twelve.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(arena_lon, arena_lat, edgecolors='k', facecolors='none', s=geog_ident_size)
plt.scatter(uc_lon, uc_lat, edgecolors='m', facecolors='none', s=geog_ident_size)
plt.scatter(athletics_lon, athletics_lat, edgecolors='b', facecolors='none', s=
    geog_ident_size)
plt.scatter(football_field_lon, football_field_lat, edgecolors='y', facecolors='none', s=
    geog_ident_size)
plt.scatter(johnston_green_lon, johnston_green_lat, edgecolors='c', facecolors ='none', s=
    geog_ident_size)
plt.scatter(fieldhouse_lon, fieldhouse_lat, edgecolors='w', facecolors='none', s=
    geog_ident_size)
plt.scatter(base_lon, base_lat, c='r',s=launch_size)
# Verified distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xlabel('Decimal␣Degrees␣[deg]',fontsize=12)
plt.ylabel('Decimal␣Degrees␣[deg]',fontsize=12)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_eight_twelve.show()
plt.savefig(direct_save+'0800_1200_map_'+filename_res+'.png')
plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_eight_twelve = numpy.zeros((nLat, 1))
LonAxis_median_eight_twelve = numpy.zeros((nLon, 1))
LatAxisIndex_eight_twelve = numpy.empty((nLat+1,1))
LatAxisIndex_eight_twelve[:] = numpy.nan
LonAxisIndex_eight_twelve = numpy.empty((nLon+1,1))
LonAxisIndex_eight_twelve[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_eight_twelve[a] = ((Lataxis_eight_twelve[a])+(Lataxis_eight_twelve[a+1]))
        /2

for j in range(0, nLon):
    LonAxis_median_eight_twelve[j] = (Lonaxis_eight_twelve[j]+Lonaxis_eight_twelve[j+1])/2
```

```
# Save latitude/longitude indices and median temperatures
output_eight_twelve_filename = direct_save+'Figure_Data/Eight_Twelve_MedianTemp_'+
    filename_res+'.txt'
outputFile_eight_twelve = open(output_eight_twelve_filename, 'w')
outputFile_eight_twelve.write("#␣Lat,␣Lon␣indices,␣median␣temp␣and␣lat/lon␣bounds␣for␣UofG␣
    Campus␣\n")
outputFile_eight_twelve.write("#By:␣Ryan␣Byerlay␣\n")
outputFile_eight_twelve.write("#Recorded␣Time␣is␣Local␣Time␣(EDT)␣\n")
outputFile_eight_twelve.write("#0:LatAxis_eight_twelve␣\t␣#1:LonAxis_eight_twelve"
                              "␣\t␣#2:MedianTemp_eight_twelve(K)␣{lat,lon}␣␣\n")

# Save data to file
for i in range(0, len(LatAxis_eight_twelve)-1):
    for j in range(0, len(LonAxis_eight_twelve)-1):
        print(TMatrix_eight_twelve_median[i][j])
        if numpy.isnan(TMatrix_eight_twelve_median[i][j]) == False:
            outputFile_eight_twelve.write("%f␣\t␣%f␣\t␣%f␣\n" % (LatAxis_median_eight_twelve
                [i],
                                                                  LonAxis_median_eight_twelve
                                                                      [j],
                                                                  TMatrix_eight_twelve_median
                                                                      [i][j]))
outputFile_eight_twelve.close()

#
    ##########################################################################################################################

# At 12:00 to 16:00
Lataxis_twelve_sixteen = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_twelve_sixteen = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_twelve_sixteen,LatAxis_twelve_sixteen = numpy.meshgrid(Lonaxis_twelve_sixteen,
    Lataxis_twelve_sixteen)

fig_twelve_sixteen,ax = plt.subplots(figsize=figuresize)
Tpcolor_twelve_sixteen=plt.pcolor(LonAxis_twelve_sixteen,LatAxis_twelve_sixteen,
    TMatrix_twelve_sixteen_median,
                                  vmin=color_bar_min_twelve_sixteen, vmax=
                                      color_bar_max_twelve_sixteen)
cbar_twelve_sixteen = plt.colorbar(Tpcolor_twelve_sixteen)
cbar_twelve_sixteen.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_twelve_sixteen.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(arena_lon, arena_lat, edgecolors='k', facecolors='none', s=geog_ident_size)
plt.scatter(uc_lon, uc_lat, edgecolors='m', facecolors='none', s=geog_ident_size)
plt.scatter(athletics_lon, athletics_lat, edgecolors='b', facecolors='none', s=
    geog_ident_size)
plt.scatter(football_field_lon, football_field_lat, edgecolors='y', facecolors='none', s=
    geog_ident_size)
plt.scatter(johnston_green_lon, johnston_green_lat, edgecolors='c', facecolors ='none', s=
    geog_ident_size)
plt.scatter(fieldhouse_lon, fieldhouse_lat, edgecolors='w', facecolors='none', s=
    geog_ident_size)
```

```
plt.scatter(base_lon, base_lat, c='r',s=launch_size)
# Verified distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xlabel('Decimal Degrees [deg]',fontsize=12)
plt.ylabel('Decimal Degrees [deg]',fontsize=12)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_twelve_sixteen.show()
plt.savefig(direct_save+'1200_1600_map_'+filename_res+'.png')
plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_twelve_sixteen = numpy.zeros((nLat, 1))
LonAxis_median_twelve_sixteen = numpy.zeros((nLon, 1))
LatAxisIndex_twelve_sixteen = numpy.empty((nLat+1,1))
LatAxisIndex_twelve_sixteen[:] = numpy.nan
LonAxisIndex_twelve_sixteen = numpy.empty((nLon+1,1))
LonAxisIndex_twelve_sixteen[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_twelve_sixteen[a] = ((Lataxis_twelve_sixteen[a])+(Lataxis_twelve_sixteen[
        a+1]))/2

for j in range(0, nLon):
    LonAxis_median_twelve_sixteen[j] = (Lonaxis_twelve_sixteen[j]+Lonaxis_twelve_sixteen[j
        +1])/2

# Save latitude/longitude indices and median temperatures
output_twelve_sixteen_filename = direct_save+'Figure_Data/Twelve_Sixteen_MedianTemp'+
    filename_res+'.txt'
outputFile_twelve_sixteen = open(output_twelve_sixteen_filename, 'w')
outputFile_twelve_sixteen.write("# Lat, Lon indices, median temp and lat/lon bounds for UofG
    Campus \n")
outputFile_twelve_sixteen.write("#By: Ryan Byerlay \n")
outputFile_twelve_sixteen.write("#Recorded Time is Local Time (EDT) \n")
outputFile_twelve_sixteen.write("#0:LatAxis_twelve_sixteen \t #1:LonAxis_twelve_sixteen"
                                " \t #2:MedianTemp_twelve_sixteen(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_twelve_sixteen)-1):
    for j in range(0, len(LonAxis_twelve_sixteen)-1):
        print(TMatrix_twelve_sixteen_median[i][j])
        if numpy.isnan(TMatrix_twelve_sixteen_median[i][j]) == False:
            outputFile_twelve_sixteen.write("%f \t %f \t %f \n" % (
                LatAxis_median_twelve_sixteen[i],
                                                                   LonAxis_median_twelve_sixteen
                                                                       [j],
                                                                   TMatrix_twelve_sixteen_median
                                                                       [i][j]))
outputFile_twelve_sixteen.close()

#
```

```
    ##########################################################################################################################

# At 16:00 to 20:00
Lataxis_sixteen_twenty = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_sixteen_twenty = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_sixteen_twenty,LatAxis_sixteen_twenty = numpy.meshgrid(Lonaxis_sixteen_twenty,
    Lataxis_sixteen_twenty)

fig_sixteen_twenty, ax = plt.subplots(figsize=figuresize)
Tpcolor_sixteen_twenty=plt.pcolor(LonAxis_sixteen_twenty,LatAxis_sixteen_twenty,
    TMatrix_sixteen_twenty_median,
                                  vmin=color_bar_min_sixteen_twenty, vmax=
                                      color_bar_max_sixteen_twenty)
cbar_sixteen_twenty = plt.colorbar(Tpcolor_sixteen_twenty)
cbar_sixteen_twenty.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_sixteen_twenty.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(arena_lon, arena_lat, edgecolors='k', facecolors='none', s=geog_ident_size)
plt.scatter(uc_lon, uc_lat, edgecolors='m', facecolors='none', s=geog_ident_size)
plt.scatter(athletics_lon, athletics_lat, edgecolors='b', facecolors='none', s=
    geog_ident_size)
plt.scatter(football_field_lon, football_field_lat, edgecolors='y', facecolors='none', s=
    geog_ident_size)
plt.scatter(johnston_green_lon, johnston_green_lat, edgecolors='c', facecolors ='none', s=
    geog_ident_size)
plt.scatter(fieldhouse_lon, fieldhouse_lat, edgecolors='w', facecolors='none', s=
    geog_ident_size)
plt.scatter(base_lon, base_lat, c='r',s=launch_size)
# Verified distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xlabel('Decimal␣Degrees␣[deg]',fontsize=12)
plt.ylabel('Decimal␣Degrees␣[deg]',fontsize=12)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_sixteen_twenty.show()
plt.savefig(direct_save+'1600_2000_map_'+filename_res+'.png')
plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_sixteen_twenty = numpy.zeros((nLat, 1))
LonAxis_median_sixteen_twenty = numpy.zeros((nLon, 1))
LatAxisIndex_sixteen_twenty = numpy.empty((nLat+1,1))
LatAxisIndex_sixteen_twenty[:] = numpy.nan
LonAxisIndex_sixteen_twenty = numpy.empty((nLon+1,1))
LonAxisIndex_sixteen_twenty[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_sixteen_twenty[a] = ((Lataxis_sixteen_twenty[a])+(Lataxis_sixteen_twenty[
        a+1]))/2

for j in range(0, nLon):
    LonAxis_median_sixteen_twenty[j] = (Lonaxis_sixteen_twenty[j]+Lonaxis_sixteen_twenty[j
```

```
        +1])/2

# Save latitude/longitude indices and median temperatures
output_sixteen_twenty_filename = direct_save+'Figure_Data/Sixteen_Twenty_MedianTemp_'+
    filename_res+'.txt'
outputFile_sixteen_twenty = open(output_sixteen_twenty_filename, 'w')
outputFile_sixteen_twenty.write("# Lat, Lon indices, median temp and lat/lon bounds for UofG
    campus \n")
outputFile_sixteen_twenty.write("#By: Ryan Byerlay \n")
outputFile_sixteen_twenty.write("#Recorded Time is Local Time (EDT) \n")
outputFile_sixteen_twenty.write("#0:LatAxis_sixteen_twenty \t #1:LonAxis_sixteen_twenty"
                                " \t #2:MedianTemp_sixteen_twenty(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_sixteen_twenty)-1):
    for j in range(0, len(LonAxis_sixteen_twenty)-1):
        print(TMatrix_sixteen_twenty_median[i][j])
        if numpy.isnan(TMatrix_sixteen_twenty_median[i][j]) == False:
            outputFile_sixteen_twenty.write("%f \t %f \t %f \n" % (
                LatAxis_median_sixteen_twenty[i],
                                                                   LonAxis_median_sixteen_twenty
                                                                       [j],
                                                                   TMatrix_sixteen_twenty_median
                                                                       [i][j]))
outputFile_sixteen_twenty.close()
#
#
    ##########################################################################################################################

# # At 20:00 to 24:00
Lataxis_twenty_twentyfour = numpy.linspace(Latmin,Latmax,nLat+1)
Lonaxis_twenty_twentyfour = numpy.linspace(Lonmin,Lonmax,nLon+1)
LonAxis_twenty_twentyfour,LatAxis_twenty_twentyfour = numpy.meshgrid(
    Lonaxis_twenty_twentyfour,
                                                                     Lataxis_twenty_twentyfour
                                                                         )

fig_twenty_twentyfour,ax = plt.subplots(figsize=figuresize)
Tpcolor_twenty_twentyfour=plt.pcolor(LonAxis_twenty_twentyfour,LatAxis_twenty_twentyfour,
                                     TMatrix_twenty_twentyfour_median,vmin=
                                         color_bar_min_twenty_twentyfour,
                                     vmax=color_bar_max_twenty_twentyfour)
cbar_twenty_twentyfour = plt.colorbar(Tpcolor_twenty_twentyfour)
cbar_twenty_twentyfour.set_label(color_bar_label,labelpad=-75,y=1.1,rotation=0,fontsize=
    axes_clrbar_label_fontsize)
cbar_twenty_twentyfour.ax.tick_params(labelsize=axes_clrbar_ticks_fontsize)
plt.scatter(arena_lon, arena_lat, edgecolors='k', facecolors='none', s=geog_ident_size)
plt.scatter(uc_lon, uc_lat, edgecolors='m', facecolors='none', s=geog_ident_size)
plt.scatter(athletics_lon, athletics_lat, edgecolors='b', facecolors='none', s=
    geog_ident_size)
plt.scatter(football_field_lon, football_field_lat, edgecolors='y', facecolors='none', s=
    geog_ident_size)
```

```
plt.scatter(johnston_green_lon, johnston_green_lat, edgecolors='c', facecolors ='none', s=
    geog_ident_size)
plt.scatter(fieldhouse_lon, fieldhouse_lat, edgecolors='w', facecolors='none', s=
    geog_ident_size)
plt.scatter(base_lon, base_lat, c='r',s=launch_size)
# Verified distances via https://www.nhc.noaa.gov/gccalc.shtml
plt.xlabel('Decimal Degrees [deg]',fontsize=12)
plt.ylabel('Decimal Degrees [deg]',fontsize=12)
plt.gcf().subplots_adjust(bottom=0.15)
plt.tight_layout()
fig_twenty_twentyfour.show()
plt.savefig(direct_save+'2000_2400_map_'+filename_res+'.png')
plt.show()

# Save the median temperature corresponding to the middle of each bin to a file
LatAxis_median_twenty_twentyfour = numpy.zeros((nLat, 1))
LonAxis_median_twenty_twentyfour = numpy.zeros((nLon, 1))
LatAxisIndex_twenty_twentyfour = numpy.empty((nLat+1,1))
LatAxisIndex_twenty_twentyfour[:] = numpy.nan
LonAxisIndex_twenty_twentyfour = numpy.empty((nLon+1,1))
LonAxisIndex_twenty_twentyfour[:] = numpy.nan

# Calculate average between each "bin" and save to new median array
for a in range(0, nLat):
    LatAxis_median_twenty_twentyfour[a] = ((Lataxis_twenty_twentyfour[a])+(
        Lataxis_twenty_twentyfour[a+1]))/2

for j in range(0, nLon):
    LonAxis_median_twenty_twentyfour[j] = (Lonaxis_twenty_twentyfour[j]+
        Lonaxis_twenty_twentyfour[j+1])/2

# Save latitude/longitude indices and median temperatures
output_twenty_twentyfour_filename = direct_save+'Figure_Data/Twenty_Twentyfour_MedianTemp_'+
    filename_res+'.txt'
outputFile_twenty_twentyfour = open(output_twenty_twentyfour_filename, 'w')
outputFile_twenty_twentyfour.write("# Lat, Lon indices, median temp and lat/lon bounds for
    UofG campus \n")
outputFile_twenty_twentyfour.write("#By: Ryan Byerlay \n")
outputFile_twenty_twentyfour.write("#Recorded Time is Local Time (EDT) \n")
outputFile_twenty_twentyfour.write("#0:LatAxis_twenty_twentyfour \t #1:
    LonAxis_twenty_twentyfour \t"
                                   " #2:MedianTemp_twenty_twentyfour(K) {lat,lon}  \n")

# Save data to file
for i in range(0, len(LatAxis_twenty_twentyfour)-1):
    for j in range(0, len(LonAxis_twenty_twentyfour)-1):
        print(TMatrix_twenty_twentyfour_median[i][j])
        if numpy.isnan(TMatrix_twenty_twentyfour_median[i][j]) == False:
            outputFile_twenty_twentyfour.write("%f \t %f \t %f \n" % (
                LatAxis_median_twenty_twentyfour[i],
                                                                    LonAxis_median_twenty_twentyfour
                                                                        [j],
```

```
                                                                    TMatrix_twenty_twentyfour_median
                                                                        [i][j]))
    outputFile_twenty_twentyfour.close()
```

Appendix B

Published Work

B.1 Peer-Reviewed Journal Papers

1. Byerlay, R. A. E., Nambiar, M. K., Nazem, A., Nahian, M. R., Biglarbegian, M., and Aliabadi, A. A., An Imaging Technique to Identify Land Surface Temperatures Using Oblique Angle Airborne Observations. *International Journal of Remote Sensing*, (in press). Content from this paper is included in **Chapters 1**, **2**, **3**, and **4** within this thesis.

2. Nambiar, M. K., Byerlay, R. A. E., Nazem, A., Nahian, M. R., Moradi, M., and Aliabadi, A. A., A Tethered And Navigated Air Blimp (TANAB) for observing the microclimate over a complex terrain. *Geoscientific Instrumentation, Methods and Data Systems*, (under review). Content from this paper is included in **Chapters 1**, **2**, and **3** within this thesis.

3. Nahian, M. R., Nazem, A., Nambiar, M. K., Byerlay, R., Mahmud, S., Seguin, M., Robe, F., Ravenhill, J., and Aliabadi, A. A., Complex Meteorology over a Complex Mining Facility: Assessment of Topography, Land Use, Grid Resolution, and PBL Scheme Modifications in WRF. *Applied Meteorology and Climatology*, (under review).

B.2 Refereed Conferences

1. Byerlay, R., Biglarbegian, M. and Aliabadi, A. A., An Airborne Thermal Imaging Methodology for Mapping Land Surface Temperature (LST) with a High Spatiotem-

poral Resolution In *Proceedings of The Joint Canadian Society for Mechanical Engineering (CSME) and CFD Society of Canada (CFDSC) International Congress (2019)*, London, Ontario, Canada.

B.3 Poster Presentations

1. Nazem, A., Nahian, M. R., Nambiar, M. K., Byerlay, R., and Aliabadi, A. A., Complex Meteorology over a Complex Mining Facility. In *Proceedings of The 27th IUGG General Assembly (2019)*, Montréal, Québec, Canada.