Viewing the Moon in Infrared

A Senior Project by Kyle Beekman

Supervised by Dr. Gary Hughes

Table of Contents

Executive Summary

Man has been fascinated by the heavens since ancient times, yet there is still so much that we don't know. This project was created by Professor Gary Hughes with goal of obtaining information about the moon and other objects in the vicinity of the Earth. The project was mostly experimental in nature and there was no specific goal at the outset of the project. In the end the project focused on the moon and meteors that traveled through the Earth's upper atmosphere.

Throughout the month of August, students traveled to the Mount Barcroft Research Station in the Eastern Sierras to gather data. The data on the moon was gathered using a camera hooked up to a telescope that had been fitted with a silver coated mirror that allowed it to reflect infrared light. The data was gathered during a four hour period when the moon was highest in the sky. A MATLAB program was used to gather data in the form of fifteen second long tiff files.

While at the mountain, meteor data was also collected using a camera with a wide-angle lens. This was pointed at the sky for long periods of time while data was collected. This was then run through a MATLAB program that determined whether a meteor had been detected.

By the end of our time at Mount Barcroft, nearly two terabytes of data had been collected. When I returned home, I began to work on a MATLAB script that would analyze and interpret all of the data we had gathered on the moon. Each tiff file contained 450 frames that captured a portion of the moon. My goal was to combine these frames into a single image that contained the surface temperature data for the entire moon.

The process of gathering the moon data into one simple picture required many steps. First I had to read each frame from each file into MATLAB so it could be used. For each of these frames I used an edge detection function to find the edge of the moon. I applied Taubin's circle fit method to the data in order to find the center and radius of the moon in each frame. The center of the moon was used to align each frame. The center of the moon in each frame was placed at the center of a large matrix. I also created a separate matrix to keep track of how many frames had contained each cell in the matrix. This matrix was used in the end to find an average temperature for each cell in the matrix. Finally, the matrix was converted from infrared units into degrees and displayed.

In the end, I had a full picture of the moon that showed the surface temperature at every point on the moon. My analysis focused on the data obtained during the night of the full moon. I found an average surface temperature of approximately 40°C across the moon with slightly higher temperatures in the center than on the edges.

Introduction

The idea of using infrared light to study celestial bodies is a new field of scientific research. During the preparation for our work, we discovered only one other study with a similar method and purpose, an experiment carried out by A.H. Maghrabi. Maghrabi studied the surface of the moon by moving a single-point laser across its surface. We proposed that an infrared camera, attached to a custom built telescope, could be used to take measurements of large portions of the moon at once. The separate pictures of the moon could then be combined and averaged to get a complete view of the moon in infrared.

The process of obtaining results on the surface temperature of the moon consisted of two main steps. The first was the data acquisition using the telescope rig. The second was an analysis conducted using MATLAB. Throughout this report, I will give detailed information on the work that was done to complete each step. I will then discuss the results of the experiment as well as where we can take the experiment in the future.

Figure 1. The Moon.

Data Gathering

Initial Preparations

Before we could begin our experiment, we had to go through a number of preparations to make sure that our plans were even possible. It was initially thought that the Earth's atmosphere might cause a problem for the infrared camera. The atmosphere gives off a large amount of heat and it was feared that this might completely shield the surface of the moon from our camera. In the early summer, before our actual experiment began, we took our telescope rig outside with the goal of discovering whether or not our fears were true.

Figure 2. Our telescope with an infrared camera feeding data into a computer. Testing equipment outside of the Cal Poly Statistics Department.

As it turned out, our fears were unfounded. Even with our initial setup, we were able to see the moon regardless of the time of day. The only thing that blocked our view were thick clouds floating overhead. Our proof of concept excited us greatly and we set about preparing for our actual experiment.

Equipment

The tests we conducted outside our building at Cal Poly proved that our setup could work for the real experiment. However, we realized that the mirror for our telescope had a major issue that would render our data unusable. The mirror initially had an aluminum coating that is standard for telescopes. Unfortunately, aluminum does not reflect infrared light very well. Figure 2 shows an image taken using the aluminum mirror. Moving the telescope to high altitude, where the atmosphere is thinner, would help, but it would not completely solve the problem. A new mirror would have to be obtained.

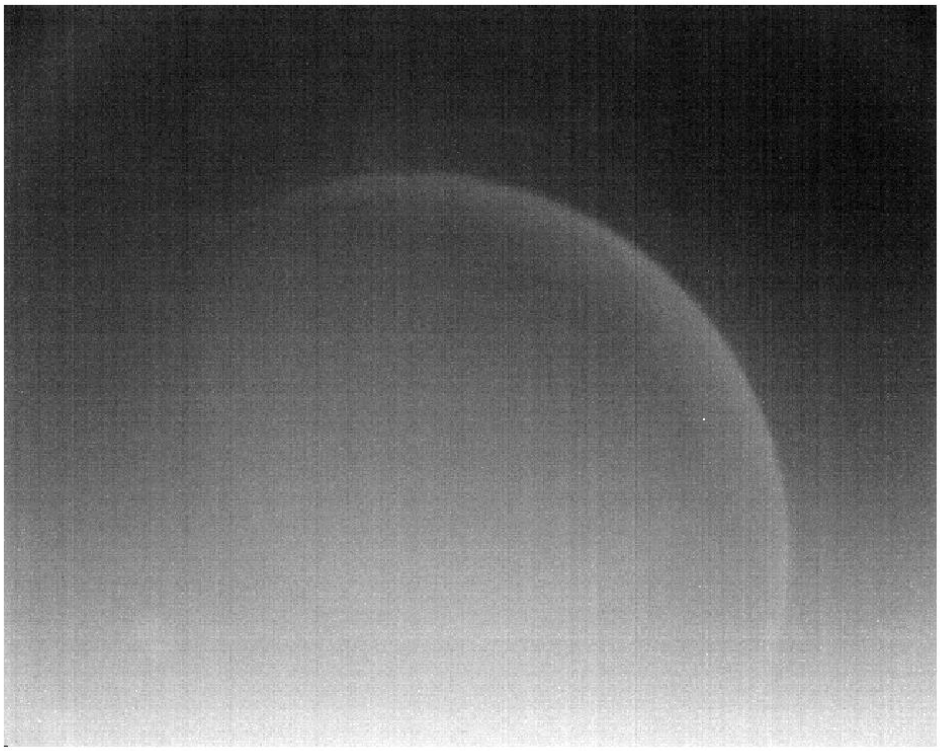

Figure 3. The moon in infrared using an aluminum coated mirror.

Aluminum provides an adequate view for the visible spectrum, wavelengths 400 nanometers to 700 nanometers, but fails when reflecting the higher wavelengths that make up infrared light. In contrast a silver coated mirror provides an excellent view for all higher spectrums of light including infrared which has wavelengths between 700 nanometers and 1000 nanometers. We had our mirror recoated in silver and began final preparations for our experiment.

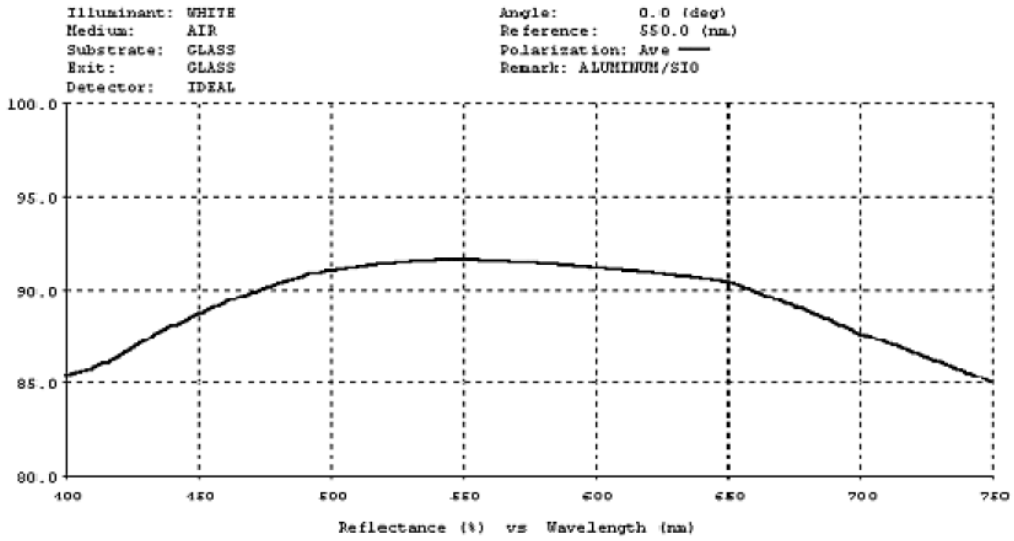

Figure 4. Percentage of Light Reflected vs. Wavelength for an aluminum coated mirror

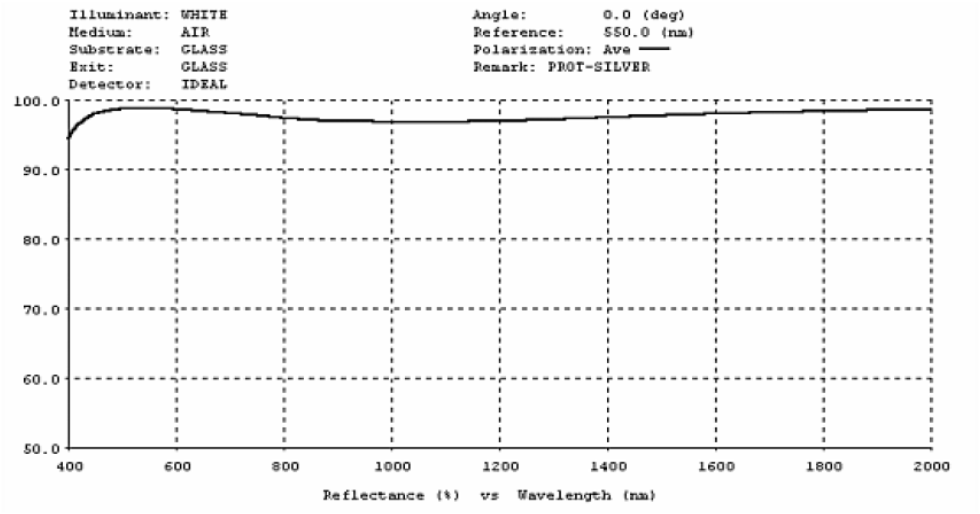

Figure 5. Percentage of Light Reflected vs. Wavelength for a silver coated mirror

Student Researchers

With our equipment ready, it was finally time to prepare our researchers for data gathering. The research team consisted of students form Cal Poly San Luis Obispo and the University of California at Santa Barbara. During the month of July, the telescope rig was ferried between San Luis Obispo and Santa Barbara so that the researchers could have a chance to practice all of the steps in the data gathering process. In this time, we also created a user manual that could be kept on hand to address any issues that came up during the data gathering process.

Research Facility

The University of California owns a research facility at Mount Barcroft in the Eastern Sierras near the border with Nevada. We were given permission to perform our experiments there during the month of August. This location was ideal because the high altitude meant that there would be less of the Earth's atmosphere to interfere with our telescope. The elevation of the White Mountain Research Facility at Mount Barcroft is 12,470 feet. This means that there is over two miles of atmosphere less to look through when compared to our initial tests, which were run at just above sea level. Researchers at the facility were provided with lodgings and meals for the duration of their stay. Throughout the month of August, researchers traveled to the facility in groups of two and stayed for approximately five days each. Every day the researchers attempted to gather data for the experiment.

Figure 6. Location of White Mountain Research Facility in the Eastern Sierras of California marked on the map.

Moon Position

As the moon orbits the earth, its rising and setting relative to the sun change every day. We decided that we would record data during the time that the moon was fifteen degrees from vertical in either direction. This meant that throughout the month of August our researchers would have to gather at different times of the day. Near the full moon, data was recorded in the middle of the night. Near the new moon data was recorded during the day. We were unable to record data in the days before and after the new moon because there would be no light on the lunar surface for our infrared camera to detect.

	Aug (PDT)						
Day	Moon Rise	Start Data	Meridian Passing	End Data	Moon Set	Illuminated	Phase
1	10:18 AM	6:25 PM	8:25 PM	8:25 PM	9:56 PM	0.303	
2	11:16 AM	7:11 PM	9:11 PM	9:11 PM	10:31 PM	0.401	
3	12:15 PM	7:59 PM	9:59 PM	9:59 PM	11:09 PM	0.506	First Quarter at 5:50 PM
4	1:15 PM	8:51 PM	10:51 PM	10:51 PM	11:52 PM	0.615	
5	2:16 PM	9:45 PM	11:45 PM	11:45 PM	12:00 AM	0.722	
6	3:17 PM	10:43 PM	12:43 AM	12:43 AM	12:42 AM	0.821	
7	4:15 PM	11:43 PM	1:43 AM	1:43 AM	1:39 AM	0.904	
8	5:09 PM	12:43 AM	2:43 AM	2:43 AM	2:43 AM	0.965	
9	5:59 PM				3:51 AM		
10	6:44 PM	1:43 AM	3:43 AM	3:43 AM	5:03 AM	0.995	Full Moon at 11:10 AM
11	7:26 PM	2:42 AM	4:42 AM	4:42 AM	6:16 AM	0.993	Perseids Meteor Shower
12	8:05 PM	3:38 AM	5:38 AM	5:38 AM	7:27 AM	0.958	Perseids Meteor Shower
13	8:43 PM	4:32 AM	6:32 AM	6:32 AM	8:37 AM	0.895	Perseids Meteor Shower
14	9:21 PM	5:25 AM	7:25 AM	7:25 AM	9:45 AM	0.810	
15	10:00 PM	6:17 AM	8:17 AM	8:17 AM	10:50 AM	0.710	
16	10:42 PM	7:09 AM	9:09 AM	9:09 AM	11:53 AM	0.603	
17	11:25 PM	8:00 AM	10:00 AM	10:00 AM	12:52 PM	0.495	Third Quarter at 5:26 AM
18		8:50 AM	10:50 AM	10:50 AM	1:47 PM	0.390	
19	12:11 AM	9:40 AM	11:40 AM	11:40 AM	2:39 PM	0.293	
20	1:00 AM	10:29 AM	12:29 PM	12:29 PM	3:26 PM	0.206	
21	1:52 AM	11:17 AM	1:17 PM	1:17 PM	4:08 PM	0.132	
22	2:44 AM	12:03 PM	2:03 PM	2:03 PM	4:47 PM	0.073	
23	3:38 AM	12:48 PM	2:48 PM	2:48 PM	5:23 PM	0.031	
24	4:33 AM	1:32 PM	3:32 PM	3:32 PM	5:56 PM	0.007	
25	5:27 AM	2:15 PM	4:15 PM	4:15 PM	6:27 PM	0.002	New Moon at 7:13 AM
26	6:22 AM	2:58 PM	4:58 PM	4:58 PM	6:58 PM	0.015	
27	7:17 AM	3:41 PM	5:41 PM	5:41 PM	7:28 PM	0.049	
28	8:13 AM	4:24 PM	6:24 PM	6:24 PM	8:00 PM	0.100	
29	9:10 AM	5:09 PM	7:09 PM	7:09 PM	8:33 PM	0.169	
30	10:08 AM	5:56 PM	7:56 PM	7:56 PM	9:09 PM	0.253	
31	11:06 AM	6:45 PM	8:45 PM	8:45 PM	9:50 PM	0.351	

Figure 7. Rising and Setting times for the moon throughout the month of August.

Gathering Process

In order to gather data in a consistent manner, students were directed to follow the instructions as stated in the user manual. Each day, the student researchers would follow the same procedures to set up the telescope, gather the data, and return the telescope and equipment into the storage shed. Gathering data using the telescope required the researchers to repeat several steps each time they attempted to collect data:

1. The students needed to align the telescope with the laser view finder

Figure 8. Laser View Finder

2. Once the telescope had been aligned, it needed to be focused so that the view of the moon was crisp and clear.
3. With a clear view, the researcher would select a section of the moon to capture. The telescope magnification did not allow us to see the whole moon at once so we decided to focus on one quarter of the moon at a time.
4. With a section of the moon in view, the researcher would then execute a MATLAB script that would direct the camera to capture footage for fifteen seconds.
5. Steps three and four would then be repeated, selecting a different section of the moon each time, until the moon had moved out of the optimal viewing range.

At the end of the day, all of the equipment would be stored in our work shed so that the experiment could be repeated the next day.

Complications

As with any experiment, several complications arose that prevented us from gathering as much data as we would have liked. At the beginning of the month, Mount Barcroft was hit by a snow storm which completely prevented data collection. Weather continued to be an issue throughout the month, but never fully prevented data collection after the first few days. Another issue faced by the researchers was altitude sickness. The thin air forced several of the researchers to leave the mountain before their planned stay was complete. This caused a major issue because they were unable to report the status of the project to the next group of students who traveled to the facility. A misplaced camera and an inability to contact the previous pair of researchers resulted in the loss of several days of work.

Secondary Experiment

A second experiment was proposed to fill up some of the time when students were not gathering data on the moon. This experiment involved attaching a second infrared to a wide-angle lens and pointing it directly up into the sky. This set up allowed us to search for meteors crossing through the Earth's upper atmosphere. The camera would record data for a set amount of time and store in on our hard drive. A MATLAB script was created to analyze this data as soon as it was gathered. The script would run through each file and search for any objects that moved through the screen. If one was found, it would output a video clip marked "Yes" to indicate that something had been seen. The script was fooled often and many clips contained bats, bugs, or satellites. However, the script did manage to find a number of meteors passing through the atmosphere.

Data Analysis

The Data

The data was recorded in the form of fifteen second long tiff files. The camera gathered data at thirty frames per second, resulting in four hundred and fifty frames in each file. All told, we gathered over a terabyte of data on the surface of the moon. The night of the full moon yielded almost thirty gigabytes of data alone.

Processing

The bulk of work on this project involved writing a MATLAB script that could process and analyze the data that we had gathered during our time on the mountain. I elected to use MATLAB because it would allow me to easily work with the code that had already been created to gather the data. My analysis would also make extensive use of matrices, making MATLAB the logical choice.

As stated previously, each tiff file only contained a portion of the moon. The moon also moves quite quickly through the frames and rotates throughout the course of the night. This meant that every frame was unique. My task was to gather these frames and realign them so that I could get a picture of the entire moon at once.

The first task my program had to complete was reading in the multitude of tiff files that we had created. I created a loop that would search for a specific subset of the tiff files in the folder which was listed as MATLAB's working directory. Along with each tiff file a video clip was created so that I could view the content of each file before using it. This was very helpful because some clips were clearly unusable due to the camera being out of focus or clouds blocking out the moon. For the bulk of my time I worked with data collected on the night of the full moon and the results presented later in this report were made from twenty-five tiff files created on that night.

My program, called TiffReader, worked on each frame of the tiff files individually. The plan was to place each frame in a larger matrix using the center of the moon as the point that each frame would use for its alignment. In order to find the center of the moon in each frame I employed edge detection using a Sobel filter in conjunction with Taubin's best fit circle algorithm.

MATLAB has a built in edge detection function that takes in several variables: the image for which edge detection will be performed, the method of edge detection, and the detection threshold. I experimented with the methods and quickly found that the Sobel method produced the best results for the images that I was using. The threshold of detection proved to be a trickier prospect.

The threshold refers to how strict MATLAB will be when looking for edges in the image. A high threshold means that the edge has to be incredibly clear whereas a lower threshold will accept blurrier borders as edges. For me, choosing a threshold to use was a tradeoff between clarity and inclusivity. If I used a high threshold and only accepted the clearest images, I would lose out on a lot of data that could still be useful. If I used a low threshold and allowed blurry data, it could compromise the integrity of the results. I elected the use a range of thresholds between .0005 units and .0006 units. My program would initially attempt to perform the detection at a .0006 threshold. If it failed, the threshold would be lowered by .00002 and the detection would be attempted again. If the threshold dropped below .0005 and the detection still failed, the frame would be considered unusable and I would move on to the next frame in the file.

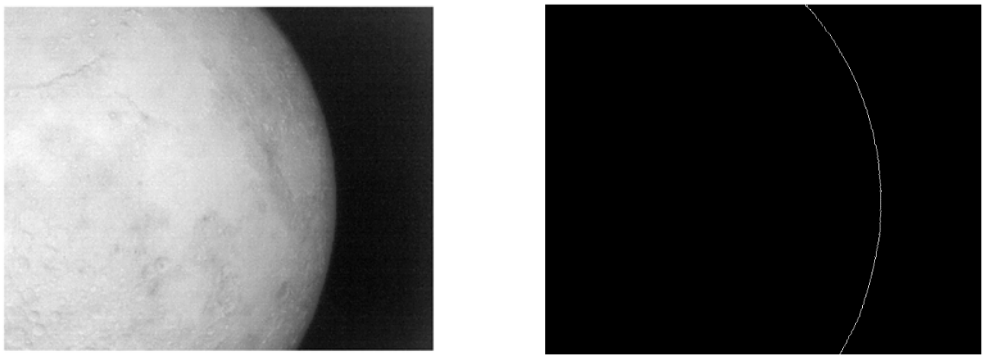

Figure 9. Sobel edge detection performed on a single frame with a threshold of .0006.

A second metric that I used for the edge detection was the number of pixels registered in the image. Sometimes the edge detection would complete successfully but there would only be a few pixels in the image. MATLAB would attempt to create a circle from these points and the results would completely throw off the rest of the analysis. I selected a minimum pixel count of twenty-five. I arrived at this number via experimentation. The number did not have to be incredibly exact as the edge detection tended to either produce only a few pixels or hundreds of pixels at a given threshold.

Once the edge detection had been completed, I recorded the coordinates of the pixels in a matrix and used it as input for Taubin's best fit circle method. There were several methods of circle fit, but Taubin's was the most robust. The method returns the radius of the circle and the coordinates of the center point. I used each of these to conduct my analysis. The radius served as one final check against bad data. I knew that the actual radius of the moon in our images was approximately four hundred pixels. I set the program to only accept circles that had a radius of less than five hundred pixels. If the radius was larger than this, the threshold of the edge detection would be lowered and the entire process would be repeated. If the circle passed all of my checks, then the center coordinates would be used to align the frame in a large matrix.

Each frame was 640 by 512 pixels. I chose a matrix size of 2281 by 2281 because it would allow the absolute worst case scenario allowed by my checks (which should be impossible) to be placed in the matrix without error. It was necessary that the dimensions of the matrix be odd so that there would be a single center point. To align the frame I employed four matrices:

1. Temp: This matrix would be where the data was initially centered. I would move the frame so that the center of the moon returned by the circle fit algorithm aligned with the center of the matrix. If the circle fit found the center to be on the right of the frame, then the data from that frame would end up in the left half of the matrix. This matrix was reset for each new frame.
2. FrameCount: After the temp matrix was updated for each new frame, the FrameCount matrix would be updated based on which cells in the Temp matrix contained data. The FrameCount matrix initially contained all zeros and for each cell in the Temp matrix that contained data, the cell in the FrameCount matrix would increment by one. This matrix did not reset during the course of the program's execution. When all of the frames had been read in I would have a count for every cell in the matrix that indicated how many data points factored into that cell. This was important for averaging the data at the end of the analysis.
3. Total: Once the counts had been updated using the Temp matrix, the data was then transferred from the Temp matrix to the Total matrix. The data was simply added to the current data. At the end of the analysis this matrix contained the total of all the data in each cell of the matrix. This matrix was initialized at zero and was never reset during the execution of the program.
4. Final: This matrix was created by dividing the Total matrix by the FrameCount matrix. Total contained the sum of all of the data in a cell and FrameCount contained a count of the data points in each cell, so by dividing them I was able to find the average value of the data on a cell by cell basis.

At the end of my process the final matrix contained average temperature data for the entire moon. However, this data was in the form of infrared units. I needed to convert it to degrees, Celsius, in order to obtain a meaningful result. The infrared scale ran from zero to 16,384 units. This corresponds, approximately linearly, to a scale that runs between negative twenty-five degrees Celsius and one hundred and thirty-five degrees Celsius. I wrote a second script, called DegreeConverter, which accepted the Final matrix as an input. The script converted the infrared units to Celsius and also assigned an artificial value to all of the cells that did not contain data. This was necessary to make a viewable image of the results.

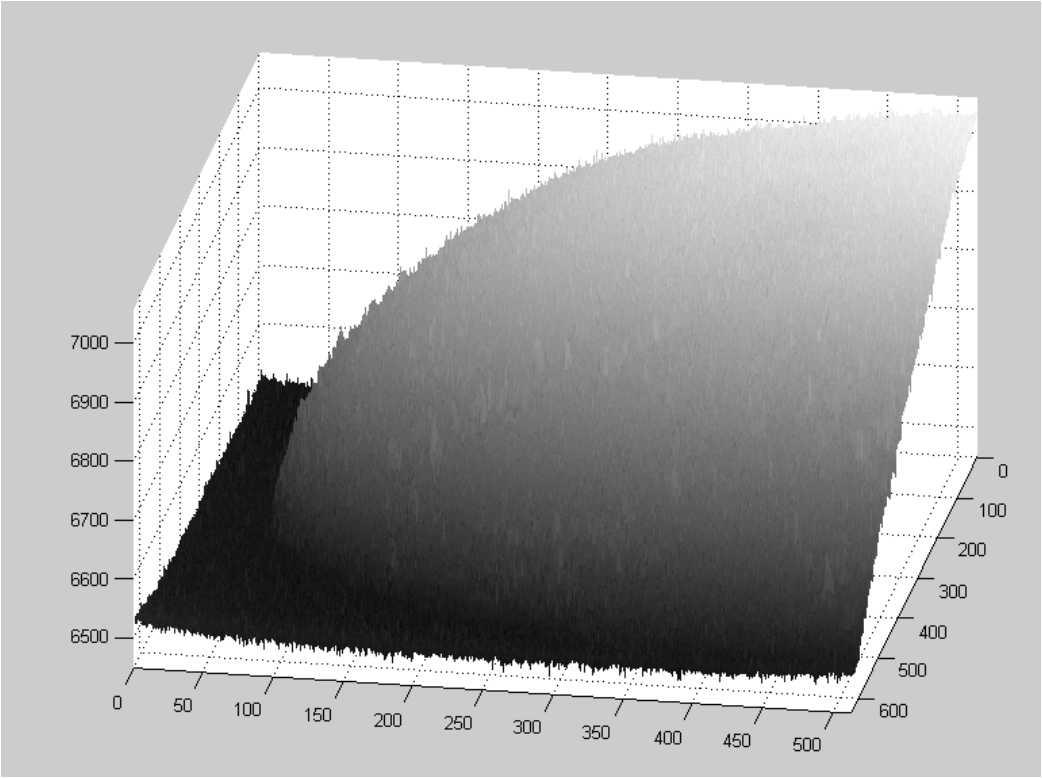

Figure 10. A single frame from a tiff file shown in 3D. The z axis represents the infrared temperature observed at that point on the moon.

Results

In the end, I was able to use nearly all of the frames from twenty-five tiff files to produce the following result.

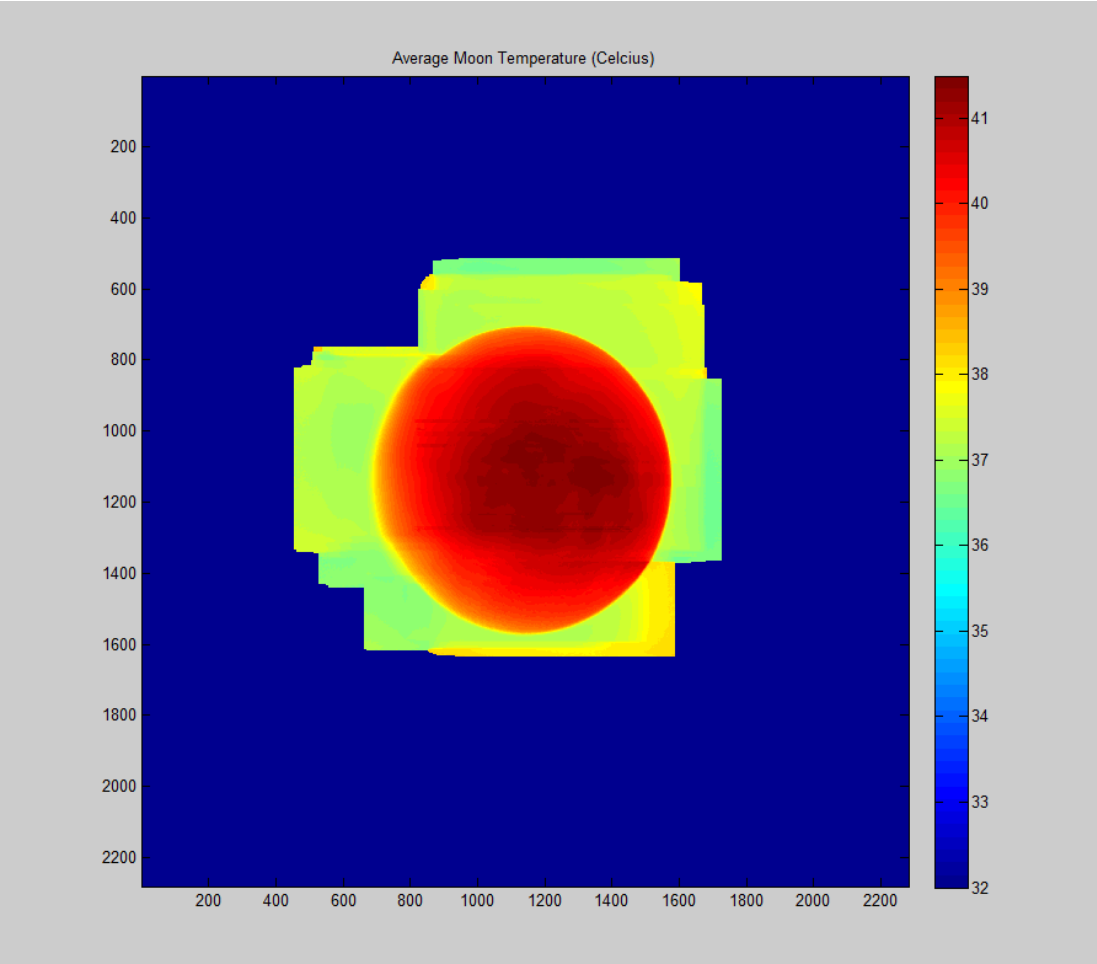

Figure 11. Heat Map of the surface of the moon. The dark blue area is where the artificial values were created to fill the empty spaces. It is not actual data.

The average temperature on the lunar surface during the full moon was found to be approximately 40°C. I was initially skeptical of this result, as it is extremely hot. However, this is makes sense considering there is no atmosphere between the Sun and the lunar surface to absorb or deflect any heat. The heat in the area beyond the edge of the moon is being given off by the Earth's atmosphere. There is atmosphere in front of the moon as well and it may be affecting the perceived temperature of the lunar surface in some way. The odd bulge on the left side of the moon is caused by blurry frames. For some reason all of the frames on the left side of the moon were less clear than the frames covering the rest of the moon. I decided to allow these frames to remain because the alternative would be having a huge chunk of the moon be missing.

Future Work

Phases of the Moon

I conducted my analysis on the full moon. The majority of my time was spent making sure that my program could handle the data and produce a result. Consequently, I chose to ignore the other phases of the moon and only focus on the full moon. I made this decision purely because I thought that the full moon would be the most interesting to look at.

More Data

As with any experiment, gathering more data can help us to get a clearer and more concrete result. I would suggest making trips to the White Mountain Research Facility or other similar location on a regular basis to obtain as much data as possible. While the moon itself should be unaffected by the rotation and revolution of the Earth, it is possible that the Earth's atmosphere may be different depending on the time of the year. This could cause the results to vary. With more data we should be able to compensate for the atmosphere of the Earth and remove it as a factor.

Better Alignment

I aligned the frames in my files using the center generated by Taubin's best fit circle algorithm. This method could be combined with several other methods in order to improve the alignment. The first method works by summing the column and row data of a frame and calculating the sum of squared errors between that frame and the data that has already been entered from previous frames. The frame will then be moved up, down, left, or right until the sum of squared errors is minimized. This will be the optimal alignment. The second method works on the same principle, attempting to minimize the sum of squared error. However, instead of using the columns or rows, the frame is moved in a square spiral outward from the starting point. If the frame is not aligned perfectly then the sum of squared errors will begin to decrease. At some point it will begin to increase again as the frame moves out of alignment. Once again, the optimal alignment is where the sum of squared errors is lowest. Ideally these three methods can be used together to ensure that all frames are matched as best as possible. Taubin's circle fit should place the center quite near where it should be. The rows and columns method should move it even closer. Finally, the square spiral method will move the frame into perfect alignment if it has not been done already.

Rotation

The moon rotates continuously as it moves through the sky. In singular tiff files, it is possible to view the craters on the moon because there has not been enough time for the moon to rotate. However, over the course of the data gathering period the moon rotates enough that particular features are no longer visible. We could compensate for this by measuring the amount that the moon rotates in a period of time and rotate our images in the opposite direction. This would allow us to see the craters and other features of the lunar surface in data that has been gathered over a long period of time.

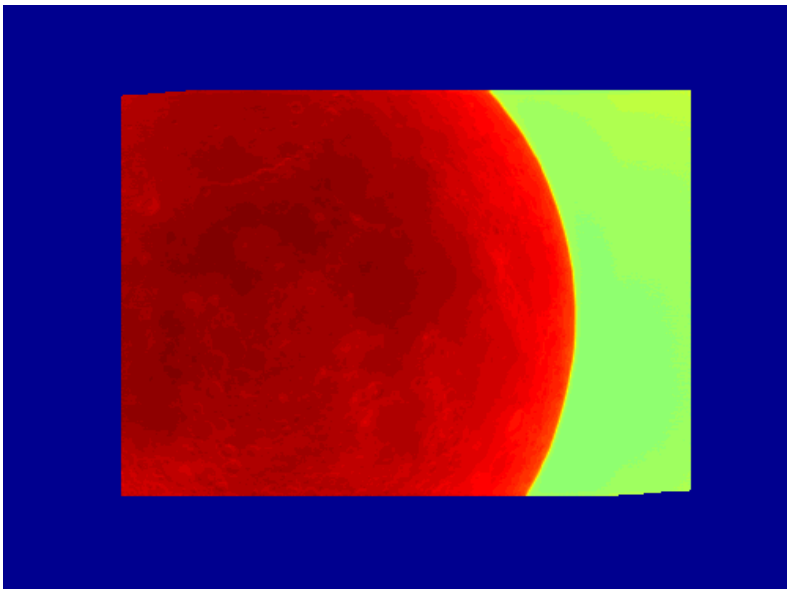

Figure 12. Image created from a single tiff file. The moon has not rotated very much and it is possible to view some of the features of the lunar surface.

Acknowledgements

I have had an amazing time working on this project and owe thanks to a great many people. I would first like to thank Dr. Gary Hughes for creating this project. I would also like to thank the University of California for allowing us to use its facilities during our data gathering process. Finally, I would also like to thank the student researchers who volunteered their time, battling altitude sickness, bad weather, and rough mountain roads, in order to gather the data for this analysis. Without the help of all of these people, I would not have been able to present this report.

References

Maghrabi, A. H. "On the measurements of the moon's infrared temperature and its relation to the phase angle." *Advances in Space Research* 53.2 (2014): 339-347.

MATLAB Code

Tiff Reader

```
%-----------------------------------------%
%   Infrared Moon Data Analysis Program   %
%   Kyle Beekman, Professor Gary Hughes   %
%-----------------------------------------%

%Frame, Count, and Loop are markers that can be used for diagnostics

Final = zeros(2281,2281); %final matrix that will contain the data
Total = zeros(2281,2281); %matrix containing total data in each cell
FrameCount = zeros(2281,2281); %used to average the data in the final matrix
Count = 0; %count number of files used
files = dir('F:\lunardata\lunar.2014.08.10\*.tif'); %file directory
Frame = zeros(size(files,1), 1); %track number of frames used in each file

%cycle through all of the files in the directory
for l=1:1%:size(files,1); %can use all files or a specific number
    fin = files(l).name;
    info = imfinfo(fin);
    imageStack = [];
    numberOfImages = length(info);

    %cycle through every frame in the current file
    for k = 1:numberOfImages
    Loop = 1;
        Temp = zeros(2281,2281); %holds data for current frame
        Threshhold = .0006; %threshhold of the edge detection
        Complete = 0; %for while loop

        %read in the current frame
        currentImage = imread(fin, k, 'Info', info);
        imageStack(:,:,k) = currentImage;

        nrows = 512;
        ncols = 640;

        %irframe = zeros (nrows, ncols, 'uint16');
        irframe = currentImage;

        % adjust some bad data
        irframe(1,:) = irframe(2,:);
    Loop = Loop + 1;

    %While loop conducts edge detection on the frame at lowering
    %threshholds until either a good result is obtained or the lower limit
    %for the threshhold is reached.
    %Complete == 1 means that a good result has been obtained
    %Complete == 2 means that we have given up
    while Complete == 0
        %edge detection using the sobel method
        BW = edge(irframe,'sobel', Threshhold);
```

```
    %imshow(BW); %shows the result of edge detection for the frame

    %convert edge detection data into form that can be used by
    %Taubin's Circle Fit method
    [a,b] = find(BW==1);
    XY = [a,b];

    %run circle fit and move on if there is an error
    try
    Circ = CircleFitByTaubin(XY);
    catch exception
    end;

    CircX = Circ(1);
    CircY = Circ(2);
    CircR = Circ(3);
    CircX; %x coordinate of circle center
    CircY; %y coordiante of circle center
    CircR; % radius of circle

    % following if statements are used to confirm a correct execution
    % of the Circle Fit, lowering the threshhold of the edge detection
    % if the circle fit failed
    if isnan(CircR)
        Threshhold = Threshhold - .00002;
        if Threshhold < .0005
            Complete = 2; %frame too blurry, we are giving up
        end;

    %circle fit occasionally works but gives ridiculous values for the
    %radius and center of the circle, radius should not be more than
    %500 pixels in size
    elseif CircR >= 500
        Threshhold = Threshhold - .00002;
        if Threshhold < .0005
            Complete = 2; %frame too blurry, we are giving up
        end;

    %we have successful circle fit with reasonable results
    else
        Complete = 1;
    end

    %circle detection mus have a reasonable number of pixels to
    %calculate, we will not accept results genrerated with less than 25
    %pixels
    s = size(XY,1);
    if Complete == 1
        if s < 25
            Complete = 0;
            Threshhold = Threshhold - .00002;
        end;
    end;
end;
```

```
%if we gave up on circle fit, skip to next frame
if Complete == 2
    continue;
end;

%imshow(BW); %shows the result of edge detection for the frame
Loop = Loop + 1;
    mini = min(irframe(:));
    maxi = max(irframe(:));

    %find the shift in coordinates from the center of the circle to the
    %center of the Temp matrix
    XDist = round(1141 - CircX);
    YDist = round(1141 - CircY);


Loop = Loop + 1;
    %use shift to allign frame within the Temp matrix
    for i=1:512;
        for j=1:640;
            Temp(i + XDist, j + YDist) = irframe(i,j);
        end;
    end;

Loop = Loop + 1;
    %for each cell in the matrix, determine if the current frame has
    %data in that cell. if yes, increase the count for the corresponding
    %cell in the FrameCount matrix, this will be used later
    for i=1:2281;
        for j=1:2281;
            if (Temp(i,j) ~= 0)
                FrameCount(i,j) = FrameCount(i,j) + 1;
            end;
        end;
    end;

Loop = Loop + 1;
    %add data from Temp matrix to Total matrix
    Total = Total + Temp;
Loop = Loop + 1;

    %counts the number of successfully read frames in the current file
    Frame(l,1) = Frame(l,1) + 1;

    %following commented out areas display a picture of the frame and a
    %3D map of the data in the frame

    %if (mod(k,2))
    % look at 3D rendering of raw data
    %surf (double (irframe), 'Edgecolor', 'none');
    %axis([0 640 0 512 min(irframe(:)) .95*max(irframe(:))])
    %axis([0 640 0 512 mini maxi]);
    %view([23 -70]);
    %colormap('gray');
```

```
        % print the 3D rendering to a .jpg file
        %print ('-djpeg', 'lwir01_01_3Ddata.jpg')

        % apply a histogram-based 'Plateau' AGC to the frame
        %irframeagc = agc (irframe, nrows, ncols);

        % display the AGCed image
        %figure,imagesc (irframeagc);
        %axis off;
        %axis image;
        %axis tight;
        %colormap('gray');

        % print the AGCed image to a .jpg file
        %print ('-djpeg', 'lwir01_01_AGCimage.jpg')
        %end
    end;

Count = Count + 1;
disp(Count);
end;


%use FrameCount matrix to find the average value for each frame in the
%Total matrix of values
for i=1:2281;
    for j=1:2281;
        Final(i,j) = round((Total(i,j)) / (FrameCount(i,j)));
    end;
end;

disp('Complete!');
```

Degree Converter

```
function Degrees = DegreeConverter(Final)

%convert infared data to degrees
%infared 0 to 16384 roughly corresponds to -25C to 135C
temp = ((Final / 16384) * 160) - 25;

%converts all NaN and negative values to 30 for a heatmap
%30 was chosen because it is far enough fromt he actuall values to be
%clearly faked, but close enough so that she scale is reasonable
for i=1:2281
    for j=1:2281
        if isnan(temp(i,j))
            temp(i,j) = 32;
        end;
        if temp(i,j) < 0
            temp(i,j) = 32;
        end;
    end;
end;

Degrees = temp;

end

%Use image(Degrees); to get a good 2d plot of the data
```

Circle Fit by Taubin

```
function Par = CircleFitByTaubin(XY)

%--------------------------------------------------------------------------
%
%     Circle fit by Taubin
%      G. Taubin, "Estimation Of Planar Curves, Surfaces And Nonplanar
%                  Space Curves Defined By Implicit Equations, With
%                  Applications To Edge And Range Image Segmentation",
%      IEEE Trans. PAMI, Vol. 13, pages 1115-1138, (1991)
%
%     Input:  XY(n,2) is the array of coordinates of n points x(i)=XY(i,1),
y(i)=XY(i,2)
%
%     Output: Par = [a b R] is the fitting circle:
%                           center (a,b) and radius R
%
%     Note: this fit does not use built-in matrix functions (except "mean"),
%           so it can be easily programmed in any programming language
%
%--------------------------------------------------------------------------

n = size(XY,1);      % number of data points

centroid = mean(XY);   % the centroid of the data set

%     computing moments (note: all moments will be normed, i.e. divided by n)

Mxx = 0; Myy = 0; Mxy = 0; Mxz = 0; Myz = 0; Mzz = 0;

for i=1:n
    Xi = XY(i,1) - centroid(1);  %  centering data
    Yi = XY(i,2) - centroid(2);  %  centering data
    Zi = Xi*Xi + Yi*Yi;
    Mxy = Mxy + Xi*Yi;
    Mxx = Mxx + Xi*Xi;
    Myy = Myy + Yi*Yi;
    Mxz = Mxz + Xi*Zi;
    Myz = Myz + Yi*Zi;
    Mzz = Mzz + Zi*Zi;
end

Mxx = Mxx/n;
Myy = Myy/n;
Mxy = Mxy/n;
Mxz = Mxz/n;
Myz = Myz/n;
Mzz = Mzz/n;

%    computing the coefficients of the characteristic polynomial

Mz = Mxx + Myy;
Cov_xy = Mxx*Myy - Mxy*Mxy;
A3 = 4*Mz;
```

```
A2 = -3*Mz*Mz - Mzz;
A1 = Mzz*Mz + 4*Cov_xy*Mz - Mxz*Mxz - Myz*Myz - Mz*Mz*Mz;
A0 = Mxz*Mxz*Myy + Myz*Myz*Mxx - Mzz*Cov_xy - 2*Mxz*Myz*Mxy + Mz*Mz*Cov_xy;
A22 = A2 + A2;
A33 = A3 + A3 + A3;

xnew = 0;
ynew = 1e+20;
epsilon = 1e-12;
IterMax = 20;

% Newton's method starting at x=0

for iter=1:IterMax
    yold = ynew;
    ynew = A0 + xnew*(A1 + xnew*(A2 + xnew*A3));
    if abs(ynew) > abs(yold)
       %disp('Newton-Taubin goes wrong direction: |ynew| > |yold|');
       xnew = 0;
       break;
    end
    Dy = A1 + xnew*(A22 + xnew*A33);
    xold = xnew;
    xnew = xold - ynew/Dy;
    if (abs((xnew-xold)/xnew) < epsilon), break, end
    if (iter >= IterMax)
        %disp('Newton-Taubin will not converge');
        xnew = 0;
    end
    if (xnew<0.)
        %fprintf(1,'Newton-Taubin negative root:  x=%f\n',xnew);
        xnew = 0;
    end
end

%  computing the circle parameters

DET = xnew*xnew - xnew*Mz + Cov_xy;
Center = [Mxz*(Myy-xnew)-Myz*Mxy , Myz*(Mxx-xnew)-Mxz*Mxy]/DET/2;

Par = [Center+centroid , sqrt(Center*Center'+Mz)];

end    %    CircleFitByTaubin
```

Image AGC

```
function img_agc = agc (img, n_rows, n_cols)

    % perform Plateau AGC on unsigned 16-bit image
    max_bit = 2^16;
    hist_bins = zeros (max_bit, 1);
    img_agc = uint8 (zeros (n_rows, n_cols));

    % set up the histogram bins, one for each 16-bit grayscale level
    for i_bin = 1:max_bit
        hist_bins(i_bin) = i_bin - 1;
    end

    % matlab calculates the image histogram
    img_hist = histc (img(:), hist_bins);

    % for Plateau Equalization, clip the bins to some plateau level
    img_hist(img_hist > 150) = 150;

    % calculate the cumulative frequency histogram from the clipped
    % histogram
    cum_hist = cumsum (img_hist);

    % scale the cumulative frequency histogram to 8-bit grayscale
    cum_hist = uint8 (255 * cum_hist / cum_hist(max_bit));

    % use the scaled cumulative frequency histogram as an intensity
    % transform table to map the original 16-bit data to 8-bit level
    for i_row = 1:n_rows
        for j_col = 1:n_cols
            img_agc(i_row,j_col) = cum_hist(img(i_row,j_col) + 1);
        end
    end

return
```

Square Spiral

```
function square_spiral
radius = 1;
n_branches = 3;
row_i = 6;
col_j = 10;

roi = zeros(20,20);
roi(:) = 1;

row_ind = row_i;
col_ind = col_j;
roi(row_ind, col_ind) = roi(row_ind, col_ind) + 20;

for i_branch = 1:n_branches
    for i_leg = 0:1
        row_base = row_ind;
        col_base = col_ind;
        for i_step = 1:i_branch*radius
            row_ind = row_base + i_step * abs (i_leg - 1) * (-1)^i_branch;
            col_ind = col_base + i_step * i_leg * (-1)^(i_branch + 1);
            roi(row_ind, col_ind) = roi(row_ind, col_ind) + 20;
        end
    end
end
disp(roi);
```