University of Arkansas, Fayetteville
ScholarWorks@UARK

Mechanical Engineering Undergraduate Honors Theses

Mechanical Engineering

5-2019

Development of Small-Scale and Low-Power Attitude Determination System for Nanoscale Satellites by Infrared Earth-Imaging Sensors

Zachary Tolar

Development of Small-Scale and Low-Power Attitude Determination System for Nanoscale Satellites by Infrared Earth-Imaging Sensors

Zachary Tolar
Dr. Adam Huang
University of Arkansas Department of Mechanical Engineering

Table of Contents

I. Abstract

Many space missions require that the spacecraft be oriented in a specific direction to operate correctly. ARKSAT 1, the University of Arkansas's first satellite, is a 1U CubeSat designed to perform atmospheric spectroscopy from Low-Earth Orbit and as such requires precise attitude determination and control. Currently, attitude determination systems for 1U CubeSats with small space, low power, and low cost restrictions do not exist. This paper discusses the development of an earth-imaging infrared camera system for attitude determination on CubeSats and SmallSats that meets these requirements. Melexis MLX 90640 IR arrays and Microchip 8-bit microcontrollers are used to create infrared images of various test targets. The contrast of the resulting images is discussed along with recommendations for future development of the system.

II. Definitions

1U, 2U, …	1 Unit, 2 Unit, … (in reference to CubeSat size)
EEPROM	Electrically Erasable Programmable Read-Only Memory
FOV	Field of View
FPU	Floating-Point Unit
GNC	Guidance, Navigation, and Control
GUI	Graphical User Interface
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IR	Infrared
kB	kilobyte
LED	Light Emitting Diode
MCC	Microchip Code Configurator
MCU	Microcontroller Unit
RAM	Random Access Memory
UART	Universal Asynchronous Receiver/Transmitter

III. Introduction

Cubesats, a type of small satellite with dimensions on the scale of 10 cm, give smaller organizations such as universities or high schools access to space exploration by decreasing the financial barrier to sending a satellite into orbit. Since the conception of the CubeSat in 1999, over 900 CubeSats have been successfully deployed [1]. The University of Arkansas has its very own SmallSat team, called ARKSAT, working on its own set of CubeSats. ARKSAT 1, the first CubeSat in the ARKSAT lineage, is a 1U CubeSat designed to use a bright, focused LED to perform atmospheric spectroscopy from low earth orbit to ground stations on earth. As ARKSAT 1 is the first of the University of Arkansas's satellites, subsystems that are instrumental to its functionality and the functionality of future ARKSAT missions must be developed. The GNC (Guidance, Navigation, and Control) subsystem is responsible for orienting the spacecraft to its surroundings and adjusting the CubeSat's attitude.

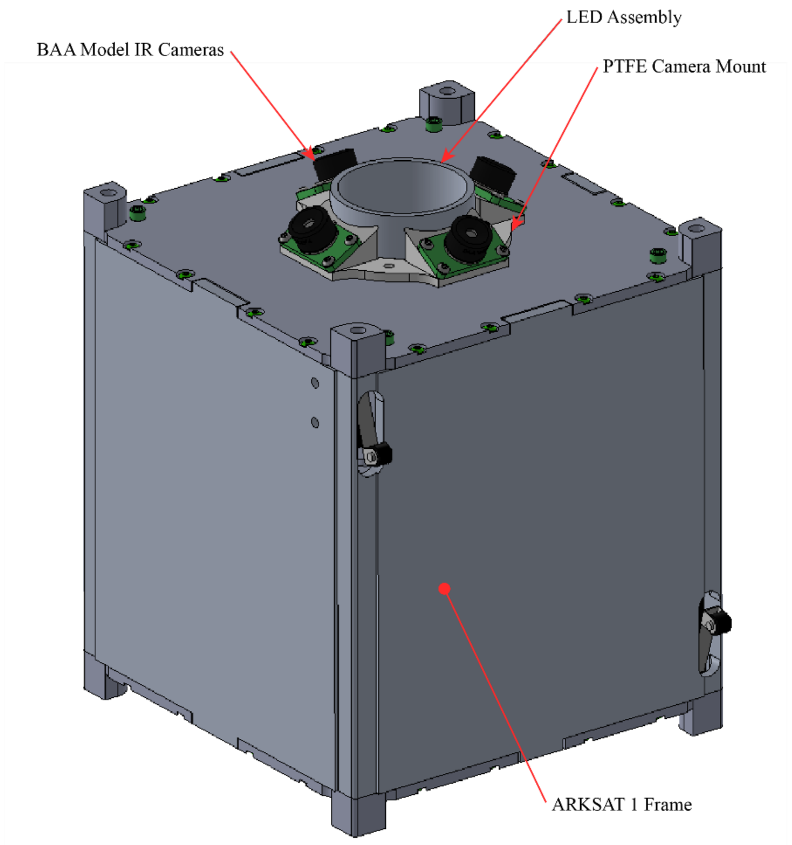


Figure 1. *Preliminary CAD model of ARKSAT 1. CAD model does not show solar panels, exterior circuit boards, or secondary IR cameras.*

ARKSAT 1 requires precise attitude determination and control for the successful operation of the LED: a misalignment of the spacecraft will result in the LED missing its intended target on the ground. ARKSAT 1 will use a collection of infrared cameras, among other instruments, to help orient itself to the earth. This paper reviews the development of the infrared camera system that will be used to relay precise attitude information to the GNC module. The GNC will make corrections to the CubeSat orientation based off of this information and data taken from other sensors.

IV. Background and Motivation

The issue of satellite attitude control is not new; many satellites have flown and are currently flying that require accurate orientation. A variety of sensors have been developed to determine a spacecraft's attitude. These include gyroscopes, horizon sensors, magnetometers, earth sensors, star trackers, and more. While ARKSAT 1 will utilize magnetometers, gyroscopes, and accelerometers, these alone cannot fully define the attitude of the spacecraft. The infrared cameras on ARKSAT 1 fall under the category of earth sensors; however, ARKSAT 1's earth sensor system differs from existing earth sensors in that it must be small, low cost, and lower power. Other cubesat teams, such as the University of Minnesota Small Satellite Project, use magnetometers and gyroscopes to point their cubesat at the sun with a tolerance of ±25° [2]. Private companies, such as Rockwell Collins, produce spacecraft orientation and attitude control equipment for spacecraft that range in mass from 30 kg to 7000 kg [3, 4]. 1U (1 unit, or 10cm cubed) cubesats have a maximum mass of 1.33 kg – much smaller than the range that these products are designed for. Clyde Space, a company that produces flight hardware specifically for CubeSats and SmallSats, produces a system that contains hardware to adjust a spacecraft's attitude down to a high precision of 0.5° but uses large and expensive star trackers to gather attitude information [5]. As ARKSAT 1 is a 1U CubeSat with limited space and, like many other CubeSat teams, a limited budget, smaller and cheaper alternatives to attitude control must be developed. Thus, the main motivation of developing this IR camera system is to produce an attitude determination system for spacecraft in low earth orbit that have low power, small size, and low-cost needs.

V. ARKSAT 1 Terminology and Design

The GNC subsystem of ARKSAT 1 and future ARKSAT missions contains all of the components necessary to determine the spacecraft's current attitude *and* make attitude adjustments (i.e. attitude determination system + magnetorquers). However, the scope of this project is limited to only the attitude determination aspect of the GNC. Further-more, while the attitude determination aspect of the GNC subsystem includes sensors other than the IR camera system (such as the magnetometers, accelerometers, etc.), this paper focuses on the development of the IR camera system. Complete attitude determination, however, requires the cooperation of all of these sensors.

V.1. Primary and Secondary Cameras

ARKSAT 1 will use two different sets of IR cameras called the primary and secondary camera groups. While the model of cameras used in each of these groups will be the same BAA model (with a wide field of view), their locations on ARKSAT 1 and intended functionalities are different. The primary camera group consists of 5 IR cameras located on the same face that the LED is mounted to. These cameras are oriented in such a way that together, they create a combined 180° solid-angle view of that side of the spacecraft with no gaps. The secondary group of cameras consists of 5 cameras in total, with one camera located on each face of the cubesat that *does not* have the LED mounted to it. These cameras do not provide overlapping coverage, but each cover a large view area of each side of the CubeSat. The secondary camera group does not have a camera mounted to the LED-face because that side already has full coverage from the primary camera group.

V.2. Phases of Attitude Adjustment

The process of attitude adjustment can be broken down into two phases. A diagram of the attitude adjustment process is shown in Figure 2 below. Note that this may not be the exact process the spacecraft uses to adjust its attitude as there may be intermittent steps that are irrelevant to this discussion. The process of attitude adjustment will occur many times over the life of the spacecraft, not just after deployment and power on as the diagram suggests.

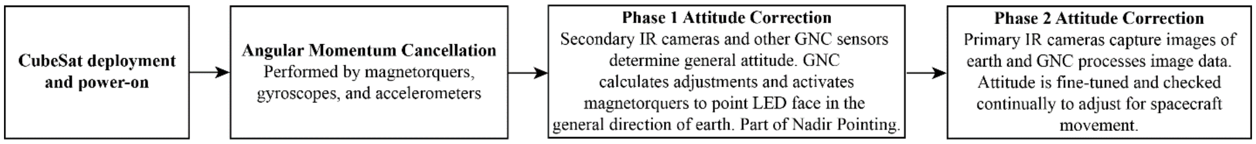

CubeSat deployment and power-on
Angular Momentum Cancellation
Performed by magnetorquers, gyroscopes, and accelerometers
Phase 1 Attitude Correction
Secondary IR cameras and other GNC sensors determine general attitude. GNC calculates adjustments and activates magnetorquers to point LED face in the general direction of earth. Part of Nadir Pointing.
Phase 2 Attitude Correction
Primary IR cameras capture images of earth and GNC processes image data. Attitude is fine-tuned and checked continually to adjust for spacecraft movement.

Figure 2. *Operational timeline for secondary and primary IR cameras on ARKSAT 1.*

The first phase of attitude adjustment will occur after the spacecraft has detumbled (cancelled its angular momentum to stop spinning). In Phase 1 of attitude correction, the secondary group of IR cameras will be utilized to get a general sense of how the spacecraft is oriented relative to the earth, sun, or moon. For example, an image can be taken with each of the 5 secondary cameras and one of the primary cameras (specifically, the primary camera that points in the direction of the face normal). If one of the cameras sees a large, relatively warm object in its FOV, it can be deduced that that face of the Cubesat is pointing in the general direction of earth. Or, if one camera sees a small but extremely bright object, it can be deduced that that face is pointing in the general direction of the sun. Using this knowledge and the known locations of the CubeSat, earth, sun, and moon, a general idea of the CubeSat orientation can be determined. This knowledge, combined with data from the magnetometers, will be given to the GNC. The GNC will then activate the magnetorquers to rotate the CubeSat so that the LED face is pointing in the general direction of the earth.

Phase 2 of attitude adjustment can begin once the LED face is generally oriented toward the earth. Phase 2 will utilize the primary IR camera group to more accurately point the LED toward a desired target on the ground. Because the primary IR cameras have a complete 180° FOV of the LED side of the spacecraft, Phase 1 will not have to accurately orient the spacecraft for the successful operation of Phase 2. During Phase 2, the GNC will read data from all of the primary IR cameras and combine the images from each camera into a single image, accounting for overlap and distortion. Image processing will be performed on the combined image to determine the attitude of the spacecraft relative to the earth and/or any specific target on the earth's surface. For example, if it were desired to point the LED face toward Fayetteville, the spacecraft could activate the magnetorquers until the image of the earth and the magnetometer readings were aligned correctly. Note that neither the cameras alone nor the magnetometer alone can fully constrain all of the degrees of rotational freedom of the CubeSat: data from both components is required to know exactly which direction the CubeSat is pointing. This will be discussed in section VI.

VI. Attitude Determination Theory

No single sensor can completely determine the spacecraft's attitude by itself – data from multiple sensors is required to fully define all of the rotational degrees of freedom relative to the earth. This section will cover how the primary IR cameras can be used in coordination with the magnetometers to fully define the spacecraft's attitude down to a level of uncertainty.

VI.1. Magnetometer Function

The magnetometers to be used on ARKSAT 1 are the InvenSense ICM-20948. This package contains, among other sensors, 3 compasses oriented perpendicular to each other so that the earth's magnetic field can be measured in the CubeSat x, y, and z directions. Of course, it doesn't do much good to know the magnetic field vector relative to the CubeSat if the magnetic field vector relative to the earth is unknown. For this, NOAA's magnetic field calculators can be used to export magnetic field data for the known CubeSat trajectory [6]. Since NASA will be continuously tracking the CubeSat and providing ephemeris data, the location of the CubeSat in orbit can be calculated for any given time. If the CubeSat location is known and the magnetic field vector is known at that location, the attitude of the CubeSat can be calculated down to 1 remaining degree of freedom. Figure 3 below demonstrates the concept.

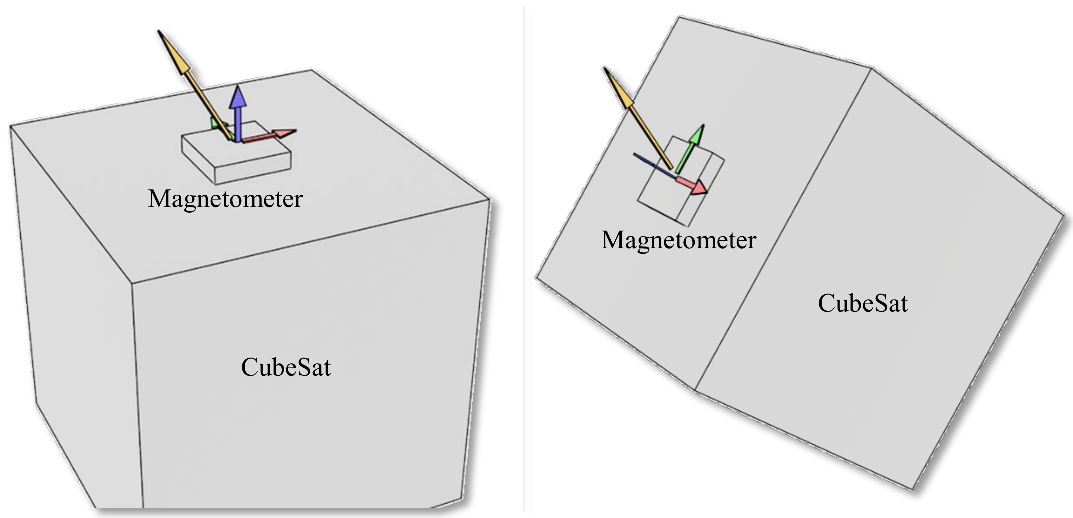


Figure 3. *The red, green, and blue arrows represent the magnetometer x, y, and z coordinates respectively. The orange arrow represents an example magnetic field vector. Notice that the x, y, and z component of the magnetic field as read by the magnetometer stay the same when the CubeSat rotates about the magnetic field vector.*

Mathematically, when the CubeSat is rotated about the magnetic field vector, the projection of the magnetic field vector onto each magnetometer axes will remain unchanged. Thus, the GNC can rotate the CubeSat until each of the magnetometer axes reads the correct calculated value, but there will still be a remaining degree of freedom that the magnetometer will not be able to constrain: the CubeSat's rotation about the magnetic field vector.

VI.2. Primary IR Camera Function

The primary IR camera group can be utilized to constrain this final degree of freedom. Once the GNC has aligned the magnetorquer to the magnetic field correctly, the CubeSat can begin to rotate itself about the magnetic field line. While rotating, the GNC will read the combined image from the IR cameras and process the image. To perform a nadir point, the image of the earth will have to appear in the center of the combined image.

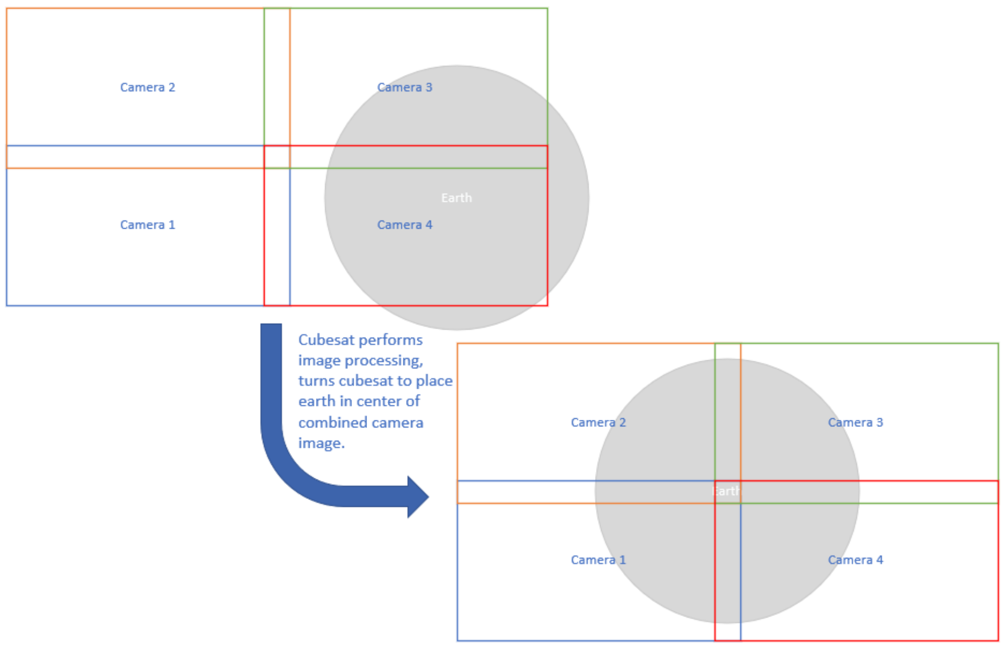

Camera 2
Camera 3
Camera 1
Camera 4
Earth
Cubesat performs image processing, turns cubesat to place earth in center of combined camera image.
Camera 2
Camera 3
Earth
Camera 1
Camera 4

Figure 4. *Example of alignment of earth-image to center of combined image frame.*

Once the image of the earth appears in the center of the frame, ARKSAT 1 will have completed a nadir point. From here, if it desired to point the LED at a specific target on earth, the CubeSat can repeat the process with updated magnetometer and earth-image requirements.

VII. IR Camera System Development

This section will discuss the process of developing the IR camera system, including the equipment that was used, descriptions of camera interfacing requirements, computational requirements, and issues encountered during the process.

VII.1. Equipment

The following equipment was used to develop the IR camera system. Some of these components are discussed in detail in later sections of this paper:

- Melexis IR Cameras (see section VII.2)
- Power supplies, oscilloscopes, multimeters
- Breadboarding and electronics equipment (resistors, capacitors, jumpers, etc.)
- Microcontrollers (see section VII.3) and necessary programmers/software
- Serial converter for RS-232 UART connect to computer
- MATLAB for image creation (see section VII.4)
- Thermoelectric cooler assembly (see section VII.4)

VII.2. IR Camera Interfacing and Control

The infrared camera chosen for ARKSAT 1 is the MLX 90640 32x24 IR array [7]. The MLX 90640 comes in two models – the BAA model with a wider field of view and the BAB model with a narrower field of view. While all of ARKSAT 1's cameras will be the BAA model, the BAB model was used for development. Because both models function exactly the same as far as interfacing with an MCU, code can be developed using the BAB model and then transferred to be used with the BAA model.

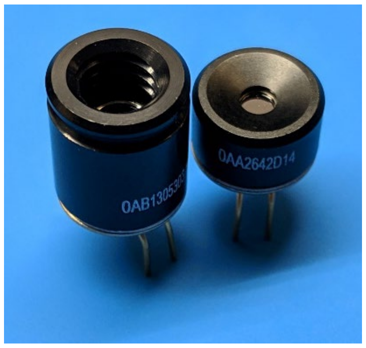


Figure 5. *Melexis IR cameras used for ARKSAT 1. Narrow FOV model BAB shown on the left, wide FOV model BAA shown on the right.*

The MLX 90640 cameras communicate with an MCU via I^2C in slave mode – the MLX 90640 will not output data unless contacted by a master I^2C device with a request. The data read/write protocols can vary between I^2C devices, and the Melexis cameras have a specific sequence of read and write commands that need to be performed by the master. See Appendix section XI.1 for a detailed description of this process.

To read temperature data from the cameras, the cameras must first be configured to operate with the desired settings. Once the camera settings have been programmed, calibration data must be extracted from the camera RAM and EEPROM. When the calibration data has been extracted, raw data corresponding to pixel temperatures can be read from the camera memory – however, this data must first be processed with the calibration data to attain accurate temperature readings or accurate pixel contrast.[1] The MCU can continually read raw data from the camera and process it to produce usable images for attitude determination. See section XI.2 in the appendix for a wiring diagram of the Melexis camera and PIC MCU.

VII.3. PIC Microcontroller Operation

Microchip's 8-bit and advanced 8-bit microcontrollers were selected to operate the Melexis cameras. Initially the PIC16F877A model 8-bit MCU was used to communicate with the PIC because of its availability, but as progress was made in communicating with the cameras, it became apparent that other MCUs, as well as other IDEs, would be necessary to successfully read data from the IR cameras. The following subsections outline the programming process with these different PIC MCUs. See section XI.3 for a comparison of the specs for the PICs used during this process.

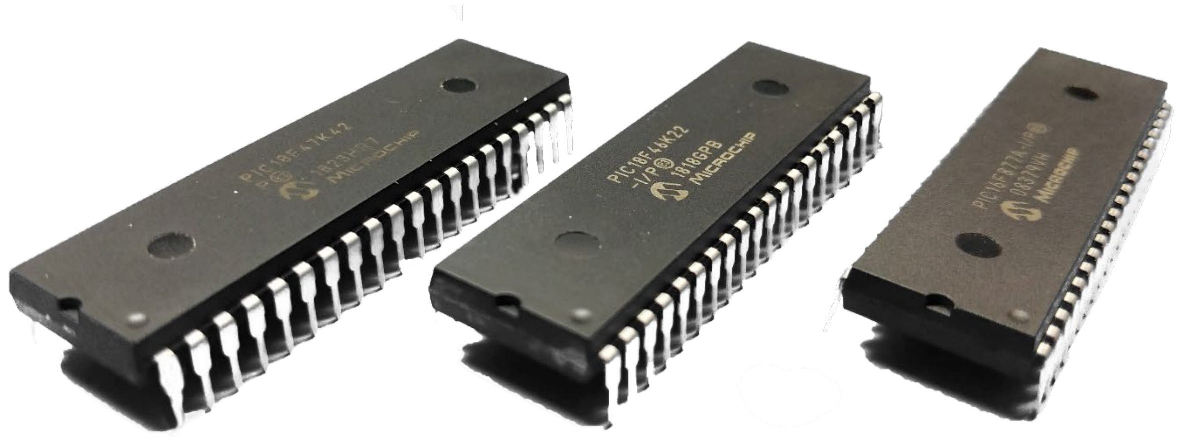


Figure 6. *From left to right, PIC18F47K42, PIC18F46K22, PIC16F877A. 40-DIP packaging was used for all PICs for easier prototyping.*

VII.3.i. PIC16F877A Model and PICBasic Pro

The PIC16F877A is an older model MCU that was readily available in the ARKSAT lab. PICBasic Pro compiler and IDE were used to write code for this MCU, while the U2 programmer was used to upload hex files into the MCU's program space. While this MCU does have a hardware

[1] In some cases, it may not be necessary to obtain accurate pixel temperatures. It may only be necessary to create an image with accurate *contrast* between pixels – the exact temperature of each pixel is not important, but rather the *difference* in temperature between pixels.

I^{2}C interface, PICBasic only has built-in *software* I^{2}C functions. While the software serial protocols are slower and can keep the processor busy during operation, they are convenient and easy-to-use functions that are built in to the compiler and IDE. This made PICBasic Pro with the 16F877A a suitable first-step for interacting with the IR camera, but it didn't take long to realize that neither the 16F877A model PIC *nor* PICBasic Pro compiler would be suitable for operating the IR cameras. This was due to the fact that the 16F877A has only 368 *bytes* of RAM while the Melexis camera requires multiple kB of data to be read, stored, and calculated. Nonetheless, a rudimentary image-export process could be programmed that did not perform any of the necessary calibration calculations and simply exported the raw pixel data. This process is shown below in Figure 7.

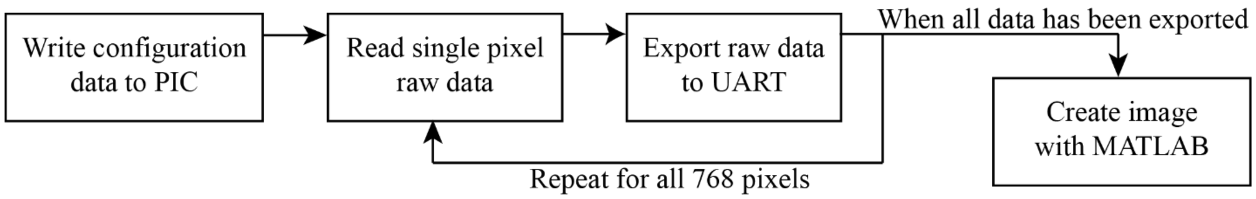

Write configuration data to PIC
Read single pixel raw data
Export raw data to UART
When all data has been exported
Create image with MATLAB
Repeat for all 768 pixels

Figure 7. *Rudimentary image creation process for PIC16F877A and PICBasic Pro.*

Additionally, issues were encountered with the PICBasic Pro compiler and IDE. PICBasic Pro is an integer-based language and does not support floating-point operations. While this *does* increase the speed of math operations, it would make writing the code to perform the necessary calibration calculations extremely difficult.

VII.3.ii. PIC18F46K22 Model and MPLAB X IDE

The next MCU selected was the PIC18F46K22 advanced 8-bit MCU. This MCU has 3896 bytes of RAM – enough to store 2 full raw images from the camera simultaneously. The programming environment was also upgraded to MPLAB X IDE, and the compiler to Microchip's XC8 compiler. This C-based programming language has support for floating point variables and even comes with precompiled header files that give C definitions to hardware that would otherwise only be accessible through the insertion of assembly code. Unfortunately, this compiler suite does not come with UART/I^{2}C libraries, so these functions had to be hand written to allow for UART/I^{2}C operation. This turned out to be a rather time-consuming task as existing online examples and code is not specific to the new Melexis cameras. Fortunately, the faster hardware I^{2}C peripheral registers could be written to and read from easily with the precompiled header files.

With new memory resources available and the ability to use floating point operations, processing actual temperature data (rather than simply outputting raw data) was attempted. However, further issues were encountered, again with the memory available to the MCU. As discussed in section XI.4 in the appendix, the amount of memory required to compute temperature data far exceeds the memory required to simply store calibration data and raw image data. 3200 bytes of data, including calibration data and raw image data, must be read from the camera and stored in the MCU RAM before any calculations can begin. This would encompass 82% of the MCUs total available memory, leaving only 696 bytes of memory available to process the data and perform the baseline operations required to run the PIC. Despite the lack of memory, an inefficient method of outputting a semi-calibrated[2] image could be performed. This method is inefficient because it requires the repeated calculation of variables due to the lack of memory available. See Figure 8 below.

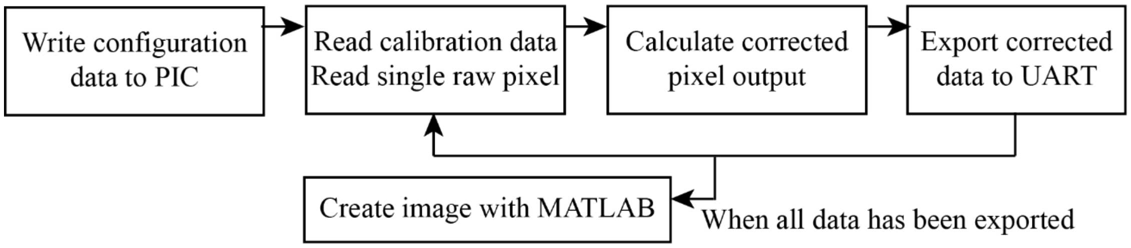

Write configuration data to PIC
Read calibration data Read single raw pixel
Calculate corrected pixel output
Export corrected data to UART
Create image with MATLAB
When all data has been exported

Figure 8. *Process for exporting corrected[2] pixel data to UART with insufficient memory on PIC18F46K22.*

VII.3.iii. <u>PIC18F47K42 Model and MCC</u>

Finally, one more MCU upgrade was made to the PIC18F47K42 model. This newer MCU comes with 8192 bytes of RAM, more than enough to store all of the calibration data, raw data, and variables necessary to complete the IR calculations and run the basic MCU functions. Microchip Code Configurator (MCC) was also added to the MPLAB X IDE, which makes setting up the pinout, oscillators, UART, and I^{2}C for the MCU easier. Unfortunately, the functions that MCC generates for I^{2}C are not compatible with the Melexis IR cameras. This new model of PIC also has a reworked hardware I^{2}C interface that functions differently from the PIC18F46K22, so the functions that were written for that MCU could not easily be transferred over. After rewriting the I^{2}C functions, a new, more efficient process of exporting data from the camera could be written.

[2] At this point in the development process, code for completely correct temperature outputs had not been written. The corrections referred to in this section only partially correct the raw pixel data.

This new code exports the completely-calibrated temperatures from the camera in a relatively efficient manner[3]. Figure 9 below outlines the process.

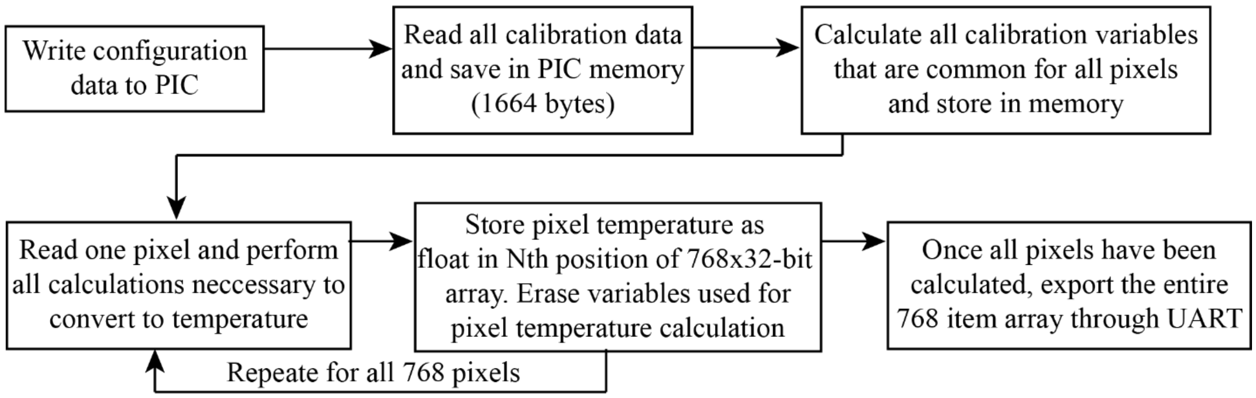

Write configuration data to PIC
Read all calibration data and save in PIC memory (1664 bytes)
Calculate all calibration variables that are common for all pixels and store in memory
Read one pixel and perform all calculations neccessary to convert to temperature
Store pixel temperature as float in Nth position of 768x32-bit array. Erase variables used for pixel temperature calculation
Once all pixels have been calculated, export the entire 768 item array through UART
Repeate for all 768 pixels

Figure 9. *Process for calculating and exporting temperatures with PIC18F47K42.*

VII.4. MATLAB Image Processing

Because the PIC microcontrollers have no way of displaying an image on a computer screen, the data from the camera (processed or unprocessed) had to be output through the PIC's serial port. The computer's RS-232 converter would read the received data to the serial monitor program on the computer. The output stream consists of a stream of 768 integers or floating-point numbers corresponding to the raw data or temperatures of each pixel. This data was then saved to a text file and read by a MATLAB program to create an image. The fully-commented code can be found in the appendix section XI.5. The code operates in the following manner:

1. Read each line from the specified text file into a 1x768 array.
2. Scale all values to be between 0 and 1. The lowest value in the entire dataset will be assigned a value of 0 and the highest value in the dataset will be assigned a value of 1. All other values will be scaled linearly in between.
3. Move the scaled entries into an array with the same dimensions as the camera array (24 tall by 32 wide).
4. Use this new array to create a grayscale image where white is the hottest pixel and black is the coldest.

[3] See section IX, "Conclusions and Future Development", for more information on future improvement of this process.

VII.5. Thermoelectric Cooler Assembly

To examine the camera's response to scenes of different temperature contrast and to simulate the image of the earth as seen by the primary camera group, a thermoelectric cooler assembly was constructed. The assembly consists of a large thermoelectric cooler with an air heat exchanger mounted to the back side of a large, 2-foot by 2-foot square of sheet metal. Attached to the front side of the large sheet metal square with thermally conductive tape is a Peltier with a smaller, circular sheet of metal taped to its hot side. When the Peltier and thermoelectric cooler are powered, the large square sheet will be cooled on both sides and the small circular sheet will be warmed from the heat extracted from the large square by the Peltier. A Melexis IR camera will point at the assembly and record pictures to simulate the view of earth (a relatively warm object with a cold background). Both the cooler assembly and the IR camera are mounted on sliding rails so the scene can be captured from different distances. Figure 10 below depicts a diagram of the cooler assembly.[4]

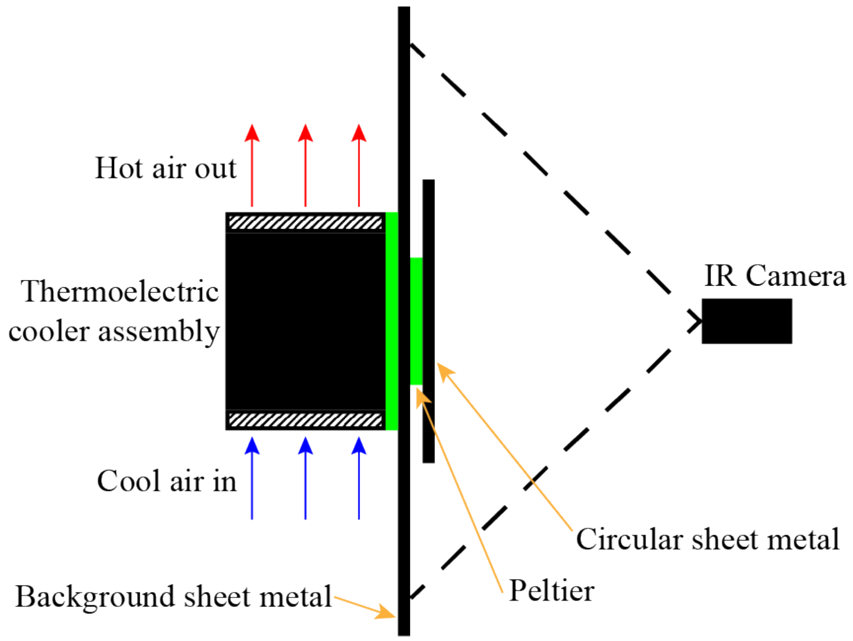


Figure 10. *Diagram of thermoelectric cooler assembly. Green objects are heating/cooling surfaces of thermoelectric cooler (left) and Peltier (right). Note that the cold side of the Peltier is it's left side while the warm side is its right side as shown in the diagram.*

[4] A special thanks to Noah Sherry and Justin Austin. The cooler assembly was designed by, and manufactured in part by them and Dr. Huang as part of the Freshmen Engineering Symposium Honors Research Colloquium.

VIII. Results and Discussion

VIII.1. Sample IR Images

The images below were created with the data from the PIC18F47K42, outputting fully-calibrated temperature data. The MATLAB function described above in section VII.4 was used to convert the data stream into a grayscale image.

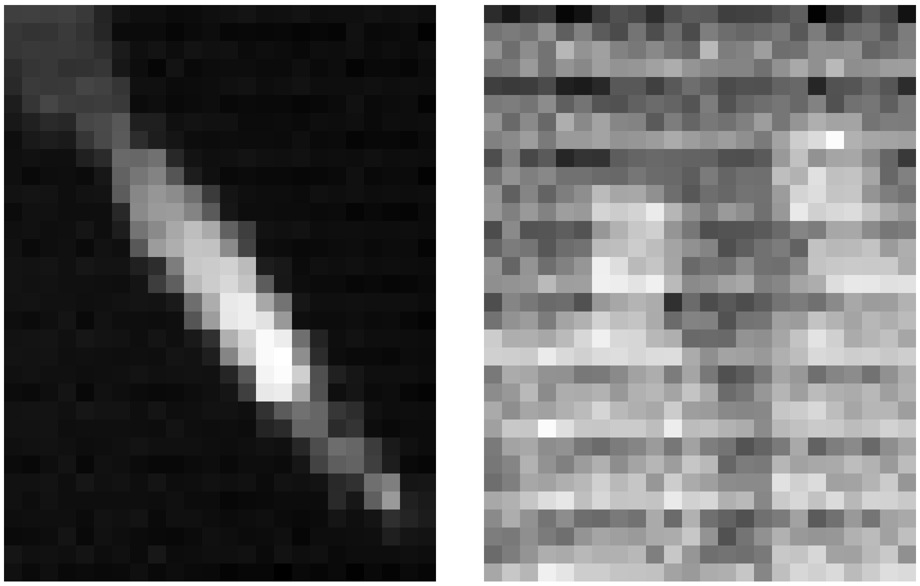


Figure 11. *First images taken with IR camera temperature data. Left: Image of soldering iron at roughly 300°C in a 20°C room, approximately 15 cm away from the camera. Right: Image of Zach Tolar and Zac Hyden at 37°C in a 20°C room, approximately 2 meters away from the camera.*

Figure 11 above shows the first pictures taken with the IR camera. Notice the image of the soldering iron on the left has much less noise than the image of the two people in the right. This is in part due to the way the MATLAB image processing works – a scene with higher thermal contrast will have low-temperature values scaled down to similar a pixel brightness, while scenes with less thermal contrast will scale pixels with a relatively close temperature to noticeably different pixel brightness. For example, if the minimum and maximum temperatures in the soldering iron scene are 17°C and 300°C respectively, then any pixel that reads 20°C or 25°C will appear nearly black, creating an image with less noise. However, with a scene that contains temperatures ranging from 17°C to 37°C, pixels with a temperature of 20°C or 25°C will appear much brighter, creating a higher-noise image.

One of the largest unknowns for the IR camera system is what the image of the earth will look like from LEO. While infrared images of earth have been taken from orbit, it is impossible to know exactly what the images will look like to the primary infrared camera group. However, images of the sun and moon can be taken from the ground to get an idea of what the secondary camera group might see. The images below of the sun and moon were created by the same code as in Figure 11.

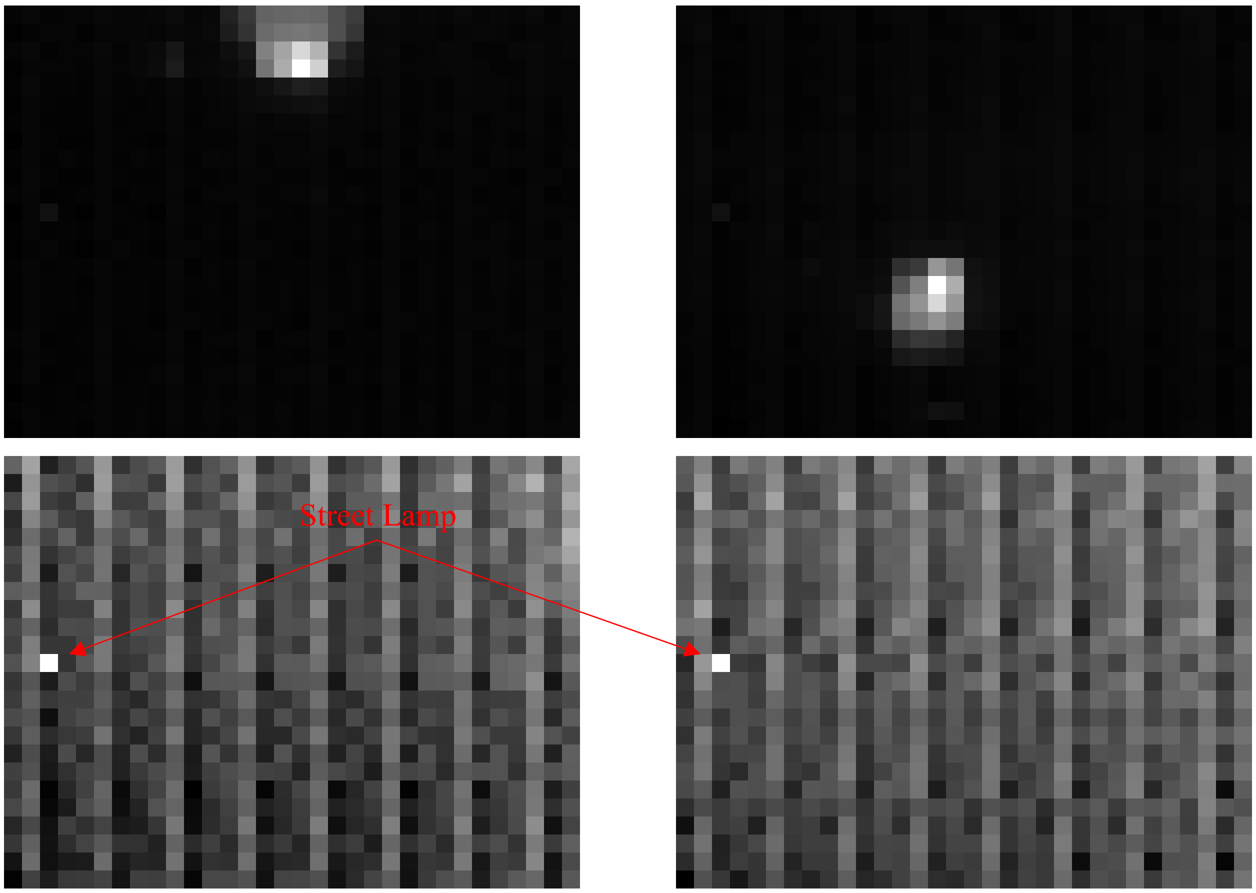


Figure 12. *Top right and top left: images of the sun taken at noon with clear skies and 17°C ambient temperature. Bottom right and bottom left: images of the moon taken at 1am with clear skies and 11°C ambient temperature.*

While the atmosphere may have some effect on how these images appear the most important piece of information these pictures provide is the relative size of the sun and the moon. There was concern that the sun and moon would not be large enough to appear in the IR image as they each have a solid angle of about 0.5° as seen from the earth. For the BAB model camera, this is about 11% of the FOV *for a single pixel*, and 2% of a single pixel for the BAA model. The BAB model can see the sun the with an extremely high contrast, even through the atmosphere, so the BAA model should have no issue detecting the sun from orbit (perhaps with a slightly lower contrast due to the larger pixel FOV). However, even the BAB model camera is not able to detect the moon. This is likely due to the atmosphere partially blocking some of the infrared radiation emitted by the moon, but the surface of the moon also has a much lower temperature than the surface of the sun (~127°C when in sunlight, as compared to the nearly 6000°C sun). It may be that the moon simply does not provide enough radiation to illuminate the 11% of a single pixel and create a significant contrast from the surrounding noise. Perhaps, however, the contrast will be greater when the thermal background is closer to 3 Kelvin, rather than the 22°C atmosphere of the earth. Temperature distributions and further analysis of the contrast of these images can be found in section 0 of the appendix. Additionally, more IR images can be found in section XI.7 of the appendix.

VIII.2. Thermoelectric Cooler Contrast Testing

The images of thermoelectric cooler assembly were taken with the cold, square background plate at 14°C and the warm, circular earth plate at 73°C. Clearly there isn't much

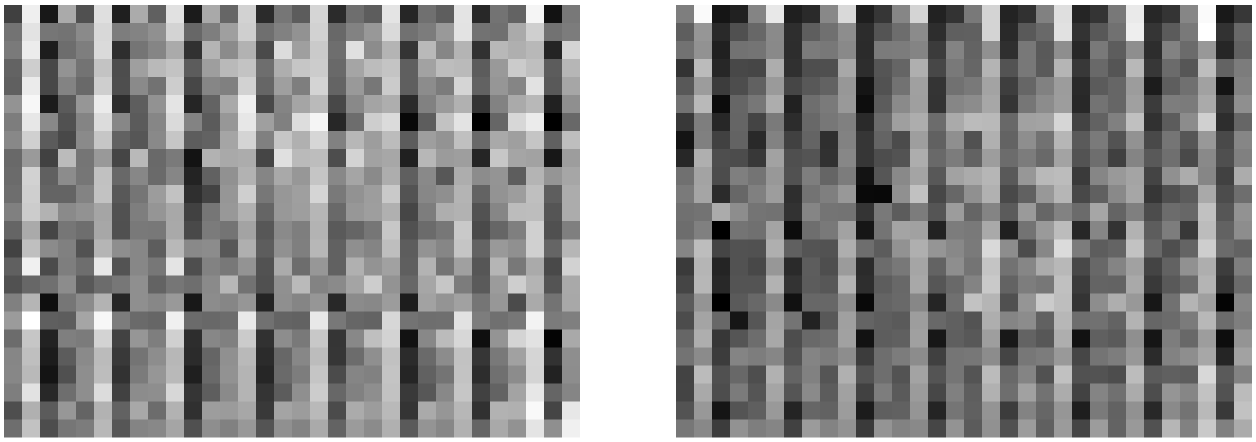


Figure 13. *Images taken of IR cooler s\assembly with BAB model camera.* $T_{background} = 14°C, T_{circle} = 73°C$.

contrast in these images, even with the relatively large temperature difference. The image on the left has no clear object in the center. However, the image the right has a *slightly* brighter circular area in the center of the image. This *is* the warm circular plate, but there is far too much noise in either of these images to perform any sort of image processing. It is proposed that this is due to the emissivity of sheet metal versus the emissivity of other objects that have been captured with the camera thus far. Bare aluminum can have emissivity in the range of 0.11, while human skin has an emissivity of about 0.98 [8]. This means that human skin is nearly a perfect blackbody – in other words, it radiates electromagnetic radiation (including IR radiation) very well in accordance with its temperature. Aluminum, on the other hand, does not. The solution to this may be painting or anodizing the aluminum, which could give it an emissivity in the range of 0.7-0.8 – but this value is highly dependent on the surface texture, color, smoothness, etc. The IR camera *does* have a calibration parameter to adjust for the emissivity of an object, however adjusting this parameter seems does not seem to affect the contrast of the images created.

VIII.3. Image Reading and Processing Time

As discussed in section VII.3.ii, a transition from PICBasic Pro programming language to MPLAB X's XC8 C was made so floating-point operations could be performed easily. Floating point operations, however, require much more overhead work from the CPU than integer-based operations do. As a result, performing repeated floating-point operations for the temperature calculation of each pixel is extremely time consuming. It can take up to 25 ms from the reading of a single pixel's raw data to the end of the its temperature calculation. For 768 pixels, this can take upwards of 20 seconds. Additionally, multiple camera images must be read and calculated within a short period of time – up to 5 cameras at time for either camera group. If the spacecraft is rotating too quickly and the processor cannot acquire data from the cameras fast enough, the images may be blurred or irregular. Potentially solutions to this are discussed in section IX.2.i, "Increasing Temperature Calculation Speed".

VIII.4. Power Consumption and Cost

As intended, the IR cameras operate with fairly low power consumption. Each camera consumes approximately 66 mW and the PIC18F47K42 MCU consumes approximately 100 mW when performing calculations. If the system ran continuously, it could run for approximately a day on ARKSAT 1's 20 W-h battery (assuming no other systems were running and the solar panels were not charging the batteries. The Melexis cameras have a price tag of about $60 each while the PIC MCU is extremely inexpensive at just a few dollars. This means a system of 10 cameras will cost on the order of $600 – much lower than the thousands of dollars required to purchase an existing attitude determination system [9].

IX. Conclusions and Future Development

IX.1. Conclusions

The MLX 90640 cameras are a cost-effective solution to earth-sensor type attitude determination systems for CubeSats. With a low profile, these cameras can be easily mounted almost anywhere on the CubeSat to provide orientation information to a CubeSat GNC subsystem. The larger FOV of the BAA model camera can give the satellite a full 180° FOV with the proper alignment of multiple cameras. While the cameras do not boast a particularly high resolution, it is expected that there will be significant contrast between high temperature objects and the cold background of space. Because the cameras require a significant amount of calculations to extract calibrated data, a fast processor may be required to handle the image computation for multiple cameras. Special care should be taken when selecting processors or MCUs for this purpose, as CubeSats usually have a low power requirement.

IX.2. Future Development

This paper covers the initial steps of developing this IR cameras system. There is still work to be done in developing the code to perform image orientation calculation, as well as fine-tuning the IR camera settings for the best noise and contrast results. The sections below contain suggestions for addressing some of the issues discussed in this paper.

IX.2.i. Increasing Temperature Calculation Speed

Currently, it takes approximately 25 ms to process temperature data for a single pixel. This sums to about 20 seconds for an entire 768-pixel temperature calculation. Part of this is due to the use of floating-point numbers, which take longer to perform calculations on than integers.

32-Bit Floating Point							
FP Addition				**FP Multiplication**			
float x	float y	period [µs]	frequency [kHz]	float x	float y	period [µs]	frequency [kHz]
1.1	3.7	12.8	78.125	1.1	3.7	36.7	27.248
17.12	37.71	12.12	82.508	17.12	37.71	36.1	27.701
173.128	337.718	12.08	82.781	173.128	337.718	36.1	27.701
9173.4128	7337.7718	15.72	63.613	9173.4128	7337.7718	36.8	27.174
4679173.4145628	4567337.7798718	11.84	84.459	4679173.4145628	4567337.7798718	37.9	26.385
				-4679173.4145628	4567337.7798718	37.8	26.455
Average:		**12.912**	**78.297**	**Average:**		**36.9**	**27.111**
32-Bit Integer							
Integer Addition				**Integer Multiplication**			
int x	int y	period [µs]	frequency [kHz]	int x	int y	period [µs]	frequency [kHz]
23	56	0.0448	22321.4286	23	56	10.8	92.593
299	-121	0.0448	22321.4286	299	-121	16.2	61.728
4867	8971	0.0448	22321.4286	4867	8971	21.2	47.170
99999	-37589	0.0480	20833.3333	99999	-37589	28.4	35.211
-678521	481222	0.0480	20833.3333	-678521	481222	52.2	19.157
				678521	481222	32.6	30.675
Average:		**0.04608**	**21726.190**	**Average:**		**26.9**	**47.756**
Math.h pow() (x^y)							
float x	float y	period [µs]	frequency [kHz]				
12.3	3.6	2810	0.3559				

Figure 14. *Average time for PIC18F47K42 to complete different mathematical operations with floating-point numbers and integers. The "math.h" header function "pow()" was also tested.*

Figure 14 above shows the frequency at which the PIC18F47K42 can perform different operations on different data types. On average, it can perform integer addition nearly 280 times faster than floating-point addition, and integer multiplication 1.37 times faster than floating-point multiplication. However, all of these operations can be performed in a period on the order of magnitude of microseconds. The *pow()* function from the "*math.h"* C header, however, takes a staggering *2.81 milliseconds* to return a value. This function is called *29* times each time a pixel is calculated. By that math, it should take much longer than 25 ms to compute a single pixel's temperature. So, if the need to use the *pow()* function can be reduced or eliminated, it is possible the temperature calculation process could be accelerated significantly. On the other hand, *pow()* is a floating-point function, so if the temperature calculation process were reprogrammed with integer computation only, the process may also significantly speed up. However, this would grossly complicate the code and mathematics, and keeping track of shifting the bits would be a complex task. Therefore, this is not recommended.

Another possible method of decreasing computation time is using a more powerful processor with a wider instruction set and a dedicated floating-point unit. For example, some Arduino units with SAMD processors have a 32-bit instruction set and FPUs built to handle floating point operations. The downsides of using a more powerful processor are the increased size of the board and the increased power consumption. It is the author's opinion that, since camera operation is not a task that will be operating continuously on the CubeSat for long periods of time, this will be the most simple and feasible option to increase the speed of pixel temperature calculation. However, special care must be taken in choosing a processor that will not draw excessive power. Arduino boards or newer PIC 32-bit processors may be the solution to this problem.

IX.2.ii. *<u>CubeSat Guidance from IR Camera Images</u>*

While this paper's subject is the development of an IR camera for use as an attitude determination system, the extent of the work done thus far does not cover using the images for CubeSat attitude determination. Most of the work for communicating with the camera, calculating and correcting its data, and determining its contrast capabilities has been done, however it is likely that some of these tasks will have to be reworked when new constraints or issues are encountered. This paper can serve as a detailed explanation of the work done to this point for future ARKSAT development and/or development of similar systems by other CubeSat teams.

Remaining is one of the largest unknowns to the IR camera system: "What will the image of the earth look like from LEO?" There is no good way to answer this question with any certainty from the ground, so it is likely that ARKSAT 1 will have to be launched perhaps without functioning image-processing code. Once in orbit, images of the earth can be taken, relayed to the ground station, and code can be reliably written for a known set of images.

X. Acknowledgements

A special thanks Dr. Huang for assisting in the development of this IR camera system by procuring equipment and software, as well as helping debug certain features while transitioning between different MCUs. I would also like to thank freshmen engineering students Noah Sherry and Justin Austin for designing and manufacturing the thermoelectric cooler assembly.

XI. References

[1] Kulu, E., "Nanosatellite & CubeSat Database," **2019**(Apr 13,) .

[2] Regents of the University of Minnesota, 2019, "Orientation and Tilt Control," **2019**(, April 13) .

[3] Dan Lockney, "Attitude Control," **2019**(, April 13) .

[4] Collins Aerospace, 2019, "Space," **2019**(, April 13) .

[5] Clyde Space, 2019, "Products," **2019**(Apr 13,) .

[6] National Geophysical Data Center, "NCEI Geomagnetic Calculators," **2019**(Apr 20,) .

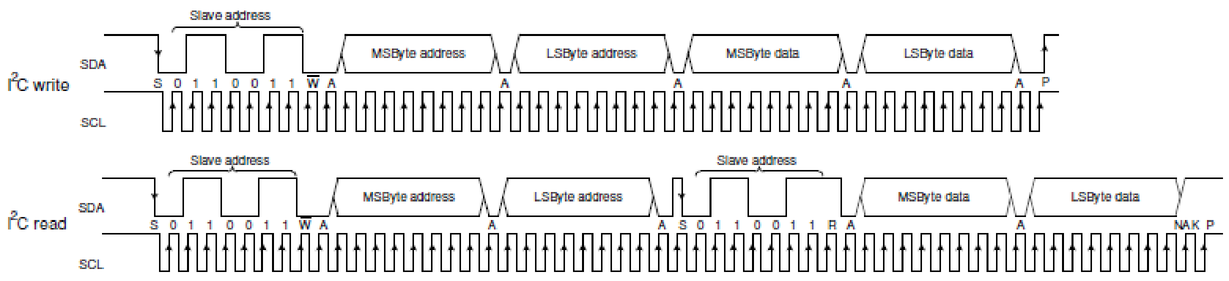
[7] Melexis, 2018, "MLX90640 32x24 IR Array," **2019**.

[8] Thermoworks, "Emissivity Table," **2019**(4/25/) .

[9] CubeSat Shop, 2019, "Attitude Sensors," .

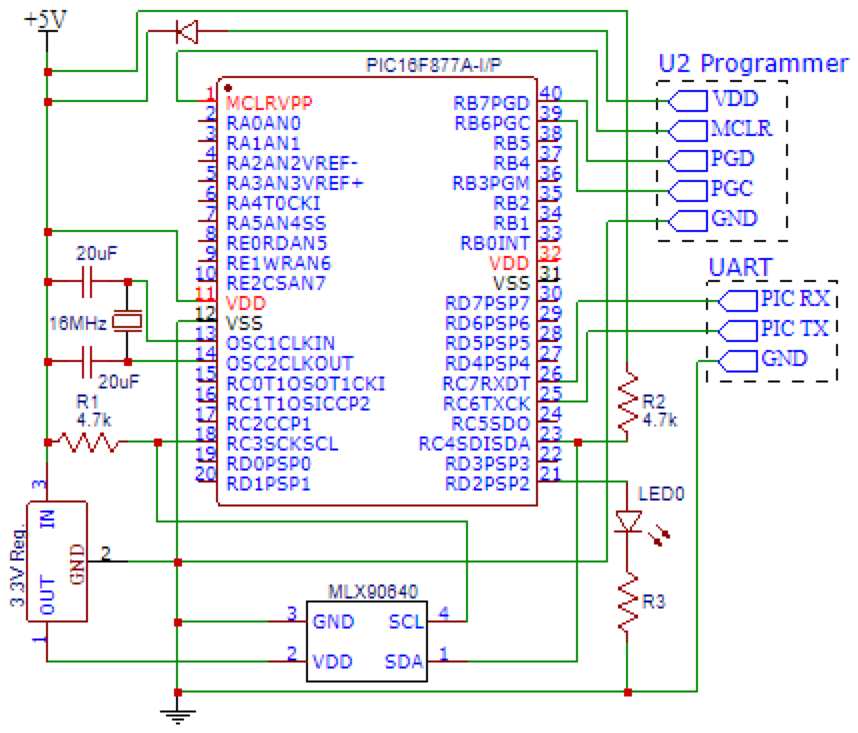
XII. Appendix

XII.1. I^{2}C Read/Write Protocol for Melexis Cameras



Images taken from Melexis 90640 datasheet [7].

XII.2. Wiring Diagram for PIC MCU and Melexis Camera



XII.3. Comparison of Specifications for PIC MCUs

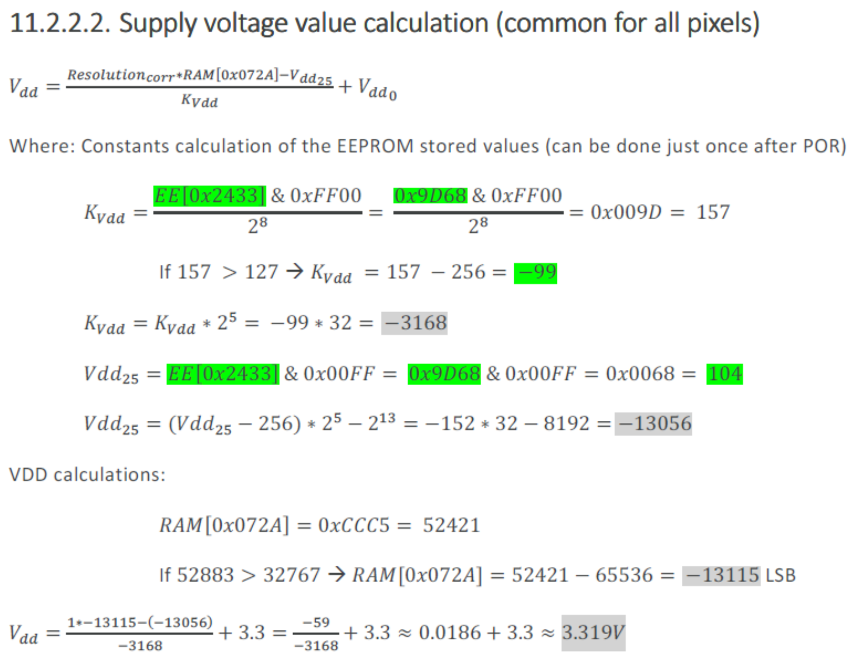
Model	RAM	Max Clock Frequency	PLL	UART	I^2C
PIC16F877A	368 B	20 MHz	No	Yes	Yes
PIC18F46K22	3896 B	64 MHz	Yes	Yes	Yes
PIC18F47K42	8 kB	64 MHz	Yes	Yes	Yes

XII.4. Temperature Calculation Process

The temperature calculation process for the MLX 90640 cameras requires that data be read from 832 calibration registers and 768 pixel raw data registers. This section of the appendix serves to give an overview of this process and examples of some of the calculations required. The Melexis datasheet explains the process in detail starting on page 30 [7].

After the PIC writes to the configuration registers of the camera, all calibration data is read from the camera and saved as 16-bit unsigned integers in the PIC RAM. This raw data takes up 20% of the PIC18F47K42's memory. Next, the PIC calculates all of the calibration constants that are common for all pixels or will only need to be calculated once. These calibration variables can be in the form of 8-bit signed or unsigned characters, 16-bit signed or unsigned words, or 32-bit floating point numbers. There is not a 1-1 relationship between the number of calibration data I^2C

reads and the number of calibration variables. For some calibration variables, only part of the 16-bit raw calibration data is needed. In some cases, multiple parts from multiples data constants must be operated on to calculate a single calibration variable. Additionally, calibration variables can be inputs for other calibration variables. For example, Figure 15 is a screen clipping from the Melexis



11.2.2.2. Supply voltage value calculation (common for all pixels)

$$V_{dd} = \frac{Resolution_{corr} * RAM[0x072A] - V_{dd_{25}}}{K_{Vdd}} + V_{dd_0}$$

Where: Constants calculation of the EEPROM stored values (can be done just once after POR)

$$K_{Vdd} = \frac{EE[0x2433] \,\&\, 0xFF00}{2^8} = \frac{0x9D68 \,\&\, 0xFF00}{2^8} = 0x009D = 157$$

If $157 > 127 \rightarrow K_{Vdd} = 157 - 256 = -99$

$$K_{Vdd} = K_{Vdd} * 2^5 = -99 * 32 = -3168$$

$$Vdd_{25} = EE[0x2433] \,\&\, 0x00FF = 0x9D68 \,\&\, 0x00FF = 0x0068 = 104$$

$$Vdd_{25} = (Vdd_{25} - 256) * 2^5 - 2^{13} = -152 * 32 - 8192 = -13056$$

VDD calculations:

$RAM[0x072A] = 0xCCC5 = 52421$

If $52883 > 32767 \rightarrow RAM[0x072A] = 52421 - 65536 = -13115$ LSB

$$V_{dd} = \frac{1 * -13115 - (-13056)}{-3168} + 3.3 = \frac{-59}{-3168} + 3.3 \approx 0.0186 + 3.3 \approx 3.319V$$

Figure 15. *Example calculation of V_{dd}, the calibration variable for common voltage for all pixels in the camera [7].*

datasheet that shows the example calculation of V_{dd}. First the MCU takes the higher byte of the EEPROM data from the camera register *0x2433*, and then divides it by 2^8. If the value is greater than 127, the it is subtracted from 256 to yield a new value: -99. This is the same as saving the raw 8-bit data from the $(0x9D68 \,\&\, 0xFF00)/2^8$ operation into a signed 8-bit character *without* changing the bits. Next, the value is multiplied by 2^5 to yield -3168. Then, the Vdd_{25} calibration variable is created by the bitwise *AND* of the camera raw data from register *0x2433* and *0x00FF* to yield 104. 256 is then subtracted from Vdd_{25} and it is then multiplied by 2^5 and added to -2^{13}. Next, the raw data from the camera RAM address *0x072A* is read and interpreted as a 16-bit signed integer with a value of -13115. Finally, V_{dd} can be calculated with all the parameters for the given equation calculated. This is one of the shorter of many of the calibration variable calibration sections and it alone contains 2 bitwise operations, 5 multiplication/division operations, 6 addition/subtraction, and two logical *if* statements. It requires operation on signed, unsigned, and floating-point variables.

Once all of the calibration that are common for all pixels have been calculated, the PIC calls the function that reads the pixel raw data and performs the rest of the necessary calculations to output pixel temperature. This means that all of the variables created in this function, including the 16-bit unsigned integer that hold the I^2C pixel raw data value, are local variables and will be erased once the function returns it's pixel temperature. This function uses the some of the calibration variables that are common for all pixels mentioned earlier, and creates many of its own new variables in the process. All of the new variables must be recalculated with the new pixel raw data every time the function is called. The function returns the calculated pixel temperature and that value is saved into an array of 32-bit floating point numbers. When all 768 pixel temperatures are stored in this array, its contents are output through UART to the computer serial port. Figure 16 below illustrates this process. The 850 lines of code for these functions were excluded from the appendix in the interest of shortening the report.

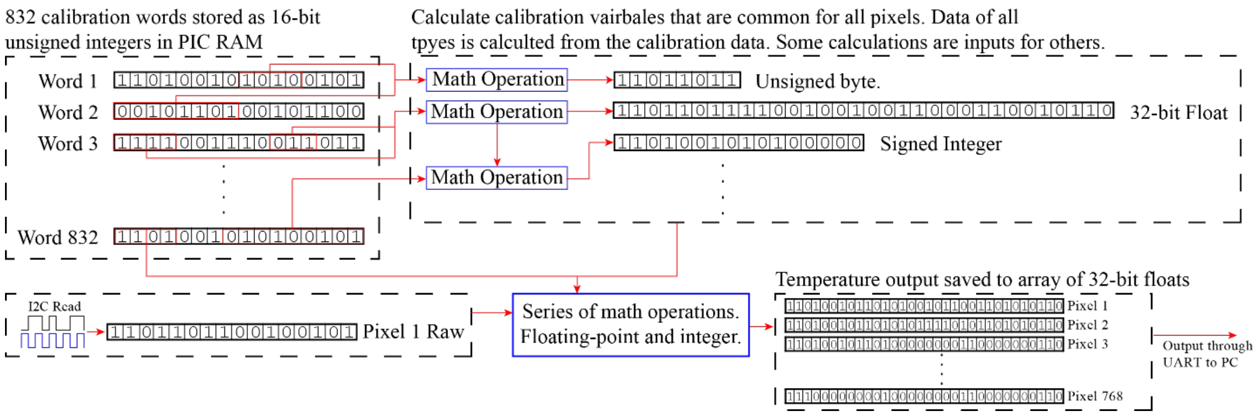
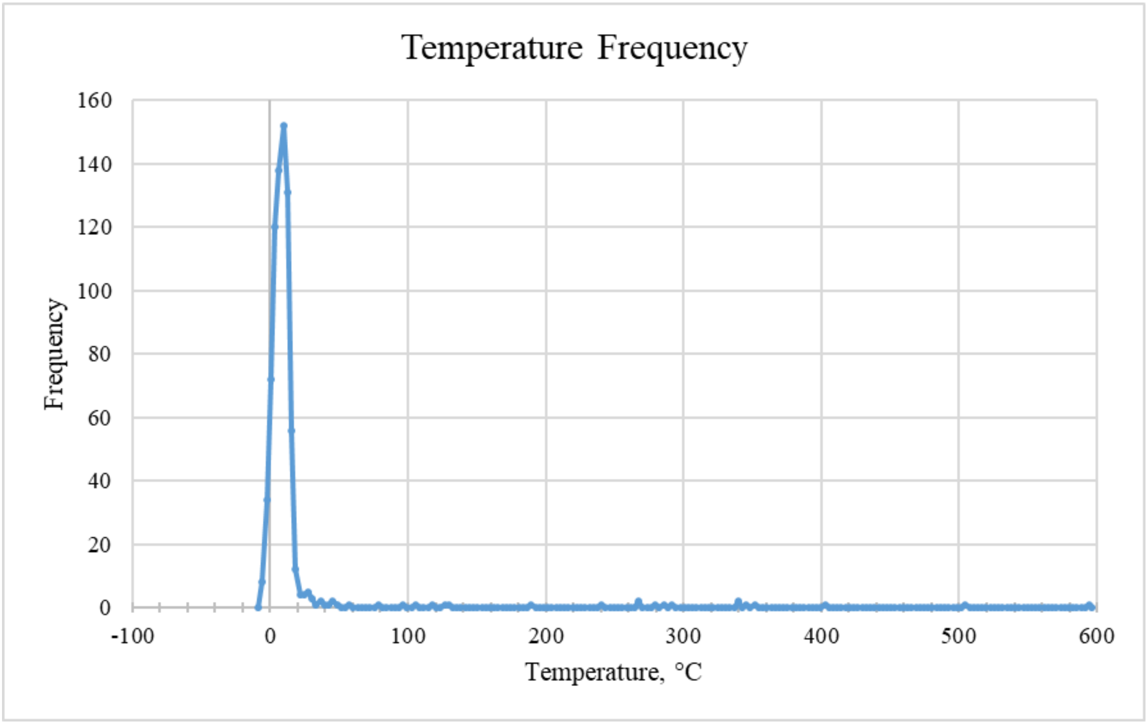

832 calibration words stored as 16-bit unsigned integers in PIC RAM
Calculate calibration vairbales that are common for all pixels. Data of all tpyes is calculted from the calibration data. Some calculations are inputs for others.
Word 1
Word 2
Word 3
Word 832
Math Operation
Math Operation
Math Operation
Unsigned byte.
32-bit Float
Signed Integer
I2C Read
Pixel 1 Raw
Series of math operations. Floating-point and integer.
Temperature output saved to array of 32-bit floats
Pixel 1
Pixel 2
Pixel 3
Pixel 768
Output through UART to PC

Figure 16. *Diagram explaining the process of calculating pixel temperatures. Note that the binary values in this image are examples and their values have no significance.*

XII.5. MATLAB Image Processing Code

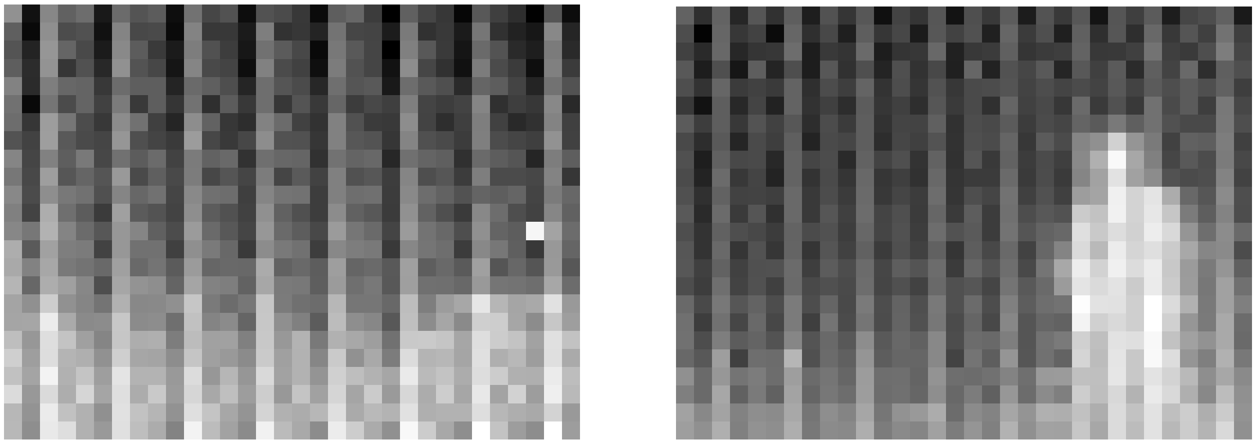
```
function [out] = process_data(File)
%This function processes the serial output stream
%from the PIC and converts it into a grayscale image.
%The input argument 'File' is a string containing the
%input.txt file name, located in the same fodler as
%this function file.

%This section opens and reads the specified file
fileID = fopen(File,'r');               %Open 'File'
formatSpec = '%f';                      %Interpret input as float
sizeA = [1 Inf];                        %Initialize matrix
A = fscanf(fileID,formatSpec,sizeA);    %Scan the file line by line
A = A';                                 %Transpose the matrix

%This section normalizes the data between 0 and 1
av = mean(A(:));        %Average the data (not used in computation)
maxi = max(A)           %Get the maximum of the entire data
mini = min(A)           %Get the minimum of the entire data
%Scale each value, one at a time
for i = 1:768
    A(i) = (A(i)*1/(maxi-mini))+(mini/(mini-maxi));
end

%This section moves the normalized data into an array with the
%correct dimensions
B = [24 32];                    %Initilaize the array
count = 1;
for y = 1:24
    for x = 1:32
        B(x,y) = A(count);
        count = count + 1;
    end
end

imwrite(B,'img.png');           %write to and save the final image
out = B;                        %output the B matrix to the console

end
```

XII.6. Sun and Moon Contrast Data



Frequency that pixels of different temperatures appear in the image of the sun (Figure 12, top right). The highest temperature is separated by approximately 12.5 standard deviations from the mean.

XII.7. Additional IR Camera Images



Left: Image of Fayetteville skyline, 11°C ambient temperature. Right: Fayetteville skyline, Zach Tolar standing in right side of image, 11°C ambient temperature.