CHARMS Software

Control of **H**ardware **A**ttitude using **R**eliable **M**agnetorquers **S**atellite

Computing and Controls Team: Andrew Gaylord, Michael Paulucci, Sophie Lama, Jackson O'Neill, Tyler Hanson, Peyton Reynolds, Daniel Burke, Rawan Shehayib, Brian Sagon, Joseph Michael Conoyer, Michael Kuczun, Nicholas Rask, William Martin, Jenna Beckley, Matthew McMahon, et al.

Senior Design Team: Isaac Brej, Sarah Kopfer, Peter Gibbons, Aidan Oblepias

University of Notre Dame – IrishSat

March 30, 2026

In Collaboration with **NearSpace Education** and the **Dream Big Program**

Contents

1 Introduction **1**
1.1 Glossary . . . 1
1.2 Problem Statement . . . 2
1.3 Process Overview . . . 3

2 Simulation **4**
2.1 Image Dataset Creation . . . 5
2.1.1 Transformation Order . . . 6

3 Detumbling **7**

4 Nadir Pointing **10**
4.1 Image Processing . . . 10
4.1.1 Tracking Over Time . . . 12
4.2 Attitude Determination . . . 13
4.3 Attitude Control . . . 16
4.3.1 2-Axis Control . . . 16
4.3.2 PD Controller . . . 17
4.3.3 Finding Gains . . . 18

5 Firmware **18**
5.1 RTOS . . . 18
5.2 Communication . . . 19
5.2.1 Uplink Packets . . . 19
5.2.2 Downlink Packets . . . 21

6 What We Learned **26**

7 Future Improvements **27**

8 Conclusion **28**

1 Introduction

CHARMS is a modular, low-cost magnetorquer-only ADCS for satellites as small as 0.5U (10cm × 10cm × 5cm). It offers a plug-and-play solution requiring only minimal power and serial communication with the host satellite bus, with no dependence on system-external inputs. Its reduced (yet sufficient) sensor suite, compact architecture, and autonomous operation make it scalable, practical, and highly adaptable. The primary mission of the CHARMS payload is to detumble with hardware constraints, while the secondary goal is to nadir point. In coordination with NearSpace Launch, the payload launched with SpaceX's Transporter 16 Mission on March 30, 2026.

CHARMS started as a 2024-2025 Electrical Engineering Senior Design Project. After the original team graduated, IrishSat's computing and controls subteams took over software development. This paper outlines our contributions to the mission's ADCS, which mostly came in the form of finishing the Earth-pointing functionality. The original team has written a report titled "CHARMS Final Report," which exhaustively outlines the project requirements and physical specifications. This paper does not attempt to emulate that; instead, it provides a technical overview of the algorithms and challenges behind **IrishSat's first flight software**.

1.1 Glossary

Table 1. Glossary of Terms

Term	Definition
Nadir	Vector from our satellite to the center of the Earth. "Nadir pointing" means the spacecraft is oriented toward the Earth.
LEO	Low Earth Orbit. CHARMS is designed for an expected LEO altitude of 300–600 km.
ADCS	Attitude Determination and Control System.
Quaternion	A four-dimensional method of representing orientation and rotations that is more robust than Euler angles.
Magnetorquers	Actuators that use induction coils to generate torque by interacting with the Earth's magnetic field. See Section 4.2 for more information.
EHS	Earth Horizon Sensor. Infrared camera used for Earth detection (see Section 2.1 for hardware information).

We define two coordinate systems (see Figure 1):

- Base frame: z is down, x is the axis along which the cams are mounted. See Section 2.1 for information on our EHS placement.
- Nadir frame: moving coordinate frame that moves with the cubesat, but z vector always pointed in nadir direction and x-axis always points along the plane that our cameras are in.

1.2 Problem Statement

When most satellites nadir point, they use their onboard GPS to easily determine which direction to face. However, to see if this is possible with cheaper hardware, **CHARMS is not equipped with a GPS**. The only sensors equipped are magnetometers, EHS's, and gyroscopes (two of each for redundancy). The payload must use the cameras to find the error quaternion (q_e) that aligns the base coordinate frame with the nadir frame. In other words, the ADCS must find how to rotate from our current orientation to nadir pointing (bottom of the payload pointing towards Earth). The power budget for this task is 0.5 Watts.

The lack of GPS clashes challengingly with another hardware constraint: the lack of reaction wheels/flywheels. Magnetorquers are the only way for CHARMS to orient itself in space. Unfortunately, they suffer from 2-axis control (see Section 4.3.1 for more info). With a GPS, we could establish an absolute (inertial) orientation and use that to find the vector that we lack control over; without one, however, it is impossible for the magnetometer to determine the actual direction of the Earth's magnetic field. Sole reliance on finicky magnetorquers is the largest challenge for our ADCS–we must constantly grapple with an unknown, uncontrollable axis.

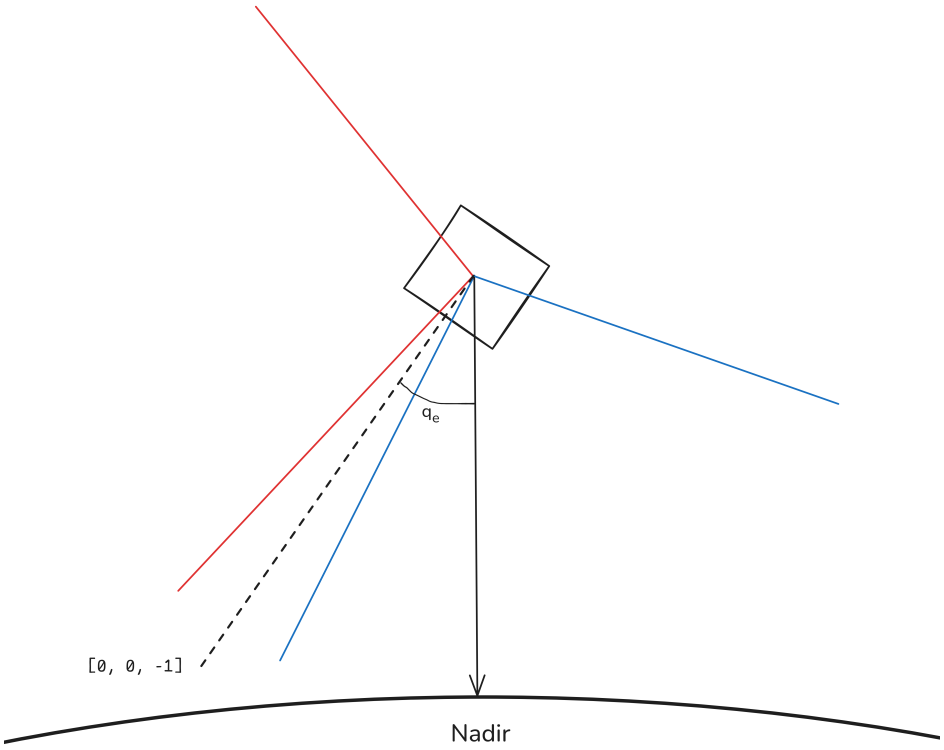

Figure 1. CHARMS inertial frames (EHS FOVs shown in color)

The NearSpace communication system allows for 200 bytes per downlink. CHARMS must be considerate of this restricted throughput while sending relevant data back to ground.

On the flip side, a primary mission is to remain as autonomous as possible. Besides some manual commands from the ground to switch modes, CHARMS should operate with minimal external input or dependencies.

1.3 Process Overview

Figure 2 gives an overview of the final control flow of the CHARMS payload. The process starts with image processing (4.1) tested on a custom EHS dataset (2.1). That information is passed to our Attitude Determination method (4.2), which was refined using a black-box method. The Attitude Control system (4.3) uses a carefully tweaked PD controller verified by LEO simulations (2) to send signals to our magnetorquers.

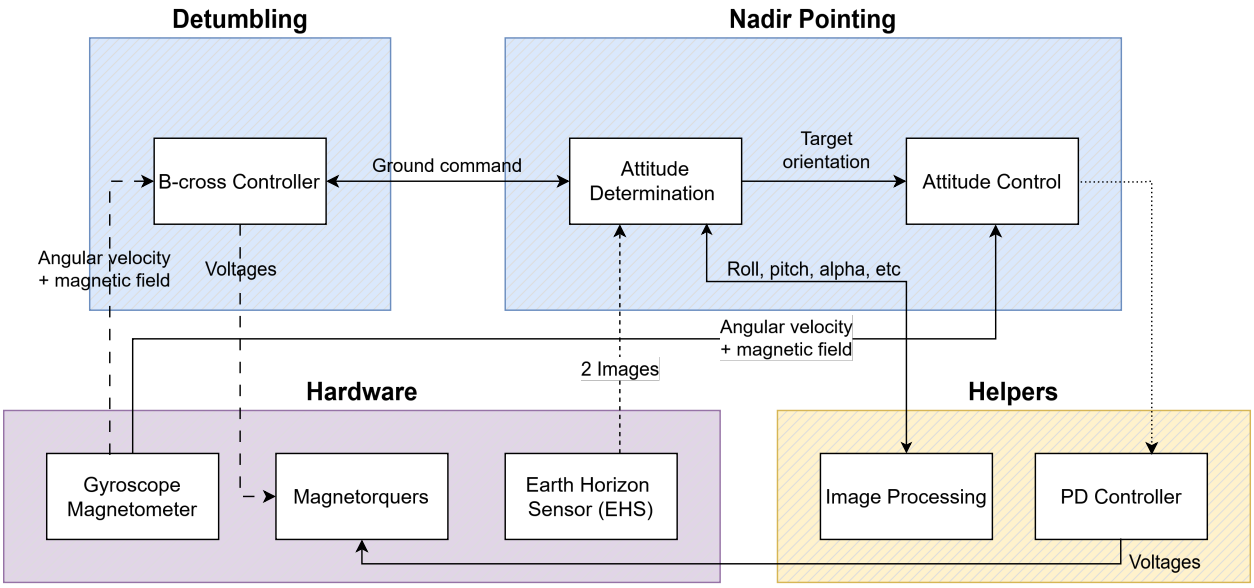

Figure 2. CHARMS process flowchart

2 Simulation

As we began brainstorming for the club's first flight software, it quickly became apparent that testing would be difficult. It would not be feasible to test abstract control methods with our limited access to testing equipment on the ground.

Our solution was to create a custom, in-house orbital simulator. Building our own testing infrastructure has many benefits: It provides good learning experience, saves cost, and grants fine-tune control over the software. 3rd party systems cannot cover all of the intricacies of our unique mission, like simulating infrared cameras.

IrishSat's closed-loop LEO simulator empowers the user to generate synthetic data, pass it to an Attitude Determination and Control System, and then propagate its state based on results. All algorithms in this paper were tested on this infrastructure. A complete overview of the simulator's capabilities can be found here. Notable features include:

- Custom Python Simulated Orbital Library (PySOL) for GPS and magnetic field generation (see Figure 3).
- Full systems modeling of our cubesat's Equations of Motion (EOM's), including magnetorquers.
- Integration with Maya (3D modeling software) for infrared camera image simulation (Figure 4).
- 100+ simulation options, including Orbital Elements, starting state, cubesat inertia, controller gains, magnetorquer specifications, image/sensor reading frequency, and many more.

- Report generation after each run, providing graphs and statistics for easy analysis (Figure 5).
- 3D cubesat visualization tool.

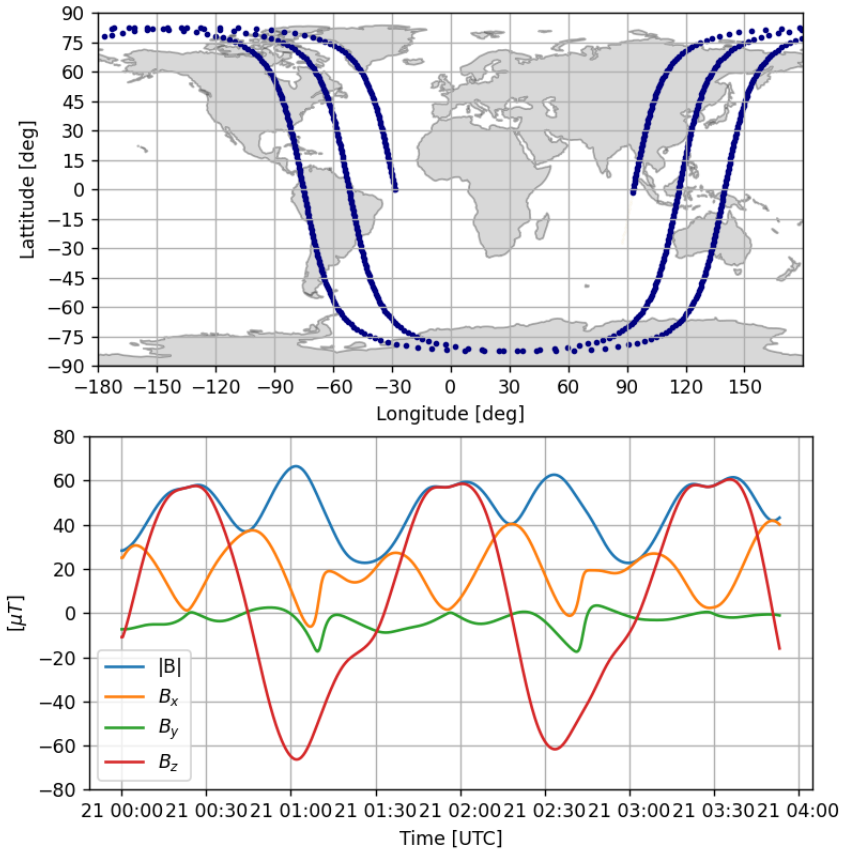

Figure 3. An example orbit generated by PySOL

2.1 Image Dataset Creation

To find and point towards the Earth, we first need to have an in-depth understanding of what our cameras will be seeing while at different angles above the Earth. CHARMS has two MLX90640 infrared (IR) cameras on opposite sides of the satellite, both pointing 30 degrees down from the x axis. They take photos with 32 vertical and 24 horizontal pixels (110° x 70° FOV). This leaves an about 10 degree blind spot on the bottom. See Figure 1 for a drawing of the setup. The challenge posed to the computing squad was to replicate realistic sensor output from these cameras for testing purposes.

EHS infrared images are artificially generated using Maya, which is 3D modeling software with raytracing capabilities. Maya's Python interfaces allow us to generate images that can be analyzed statically or fed into our control loop.

The computing squad designed a Maya script that collects data points (synthetic EHS images) from 360° rotations and stores them with the ground truth of what angle the satellite is at with respect to nadir. By associating generated images with known, correct attitude values, we could employ a black box testing methodology to validate our nadir pointing pipeline. Synthetic infrared images are generated in Maya at known spacecraft orientations, and each image is stored alongside its ground truth angle relative to nadir. These image/label pairs form a reference data set that allows attitude determination algorithms to be tested across a variety of orbital situations by comparing their outputs against the expected values. By treating the system as a black box, we have a reliable and repeatable means of verification that is independent of any method of solving for the angle of the spacecraft.

Each pair of images is generated alongside a line of metadata. See Figures 13 and 14 for an example image pair. An example line of metadata generated is:


```
{"tag": 28, "yaw": 0, "pitch": 45, "roll": 15,
"image1": "cam1_s27_roll15_pitch45_yaw0_1.png",
"image2": "cam2_s27_roll15_pitch45_yaw0_1.png",
"yaw_invarient_degrees": 45.248357}
```


In this line, the tag represents the index of the image pair, so this would be the 28th pair of images generated. The yaw, pitch, and roll numbers were the degrees in which the satellite was rotated in the simulation (this is elaborated on in 2.1.1). The image1 and image2 are simply the file names of the images generated with this iteration of rotations. Finally, the yaw invarient degrees is the minimum number of degrees between the vector of the bottom of our satellite, and nadir pointing. We do not store a direction for this angle. The yaw invarient degrees is the main parameter used to check for if our program is working.

2.1.1 Transformation Order

Each camera undergoes INTRINSIC (ie. local, changing axis after each subsequent change) rotations applied in this sequence:

1. BASE ORIENTATION: Camera 1 points directly at Earth center using look_at(). Camera 2 is always tied to camera 1 with a 2 * PAIR_ANGLE_DEGREES offset. This means that the base orientation is NOT nadir pointing, but rather just camera 1 pointing at the Earth directly, and camera 2 offset by the needed amount. It is not important that you understand this base orientation for any reason other than how the camera pairs are generated. It will not be used in any calculations. The amount that the camera pair is off from nadir pointing after the rotations will be calculated later.

2. YAW (applied around local UP axis: +Y in camera frame)

3. PITCH (applied around local RIGHT axis: -Z in camera frame)

4. TILT (applied around local FORWARD axis: +X in camera frame)

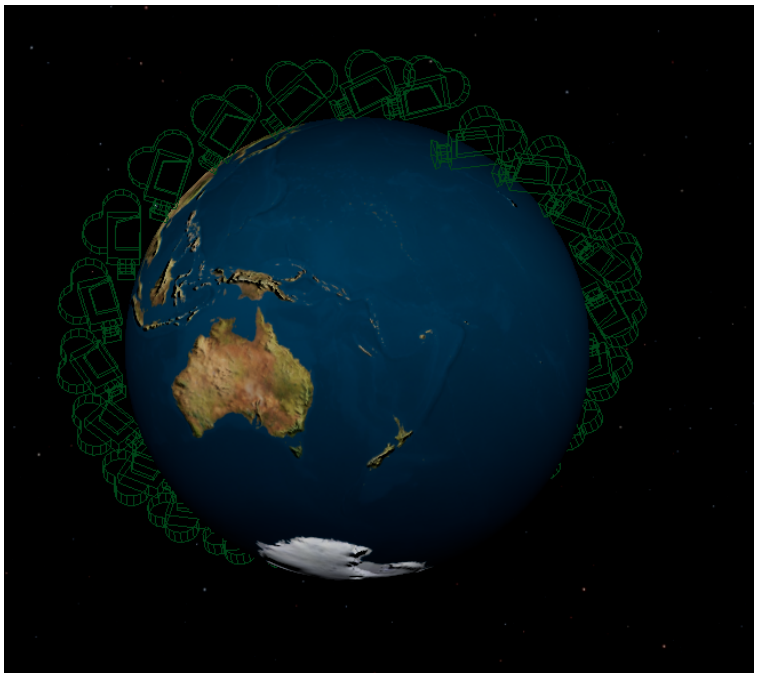

Figure 4. Example of cameras placed along orbit in Maya

The order in which these Euler transformations are applied is extremely important: yaw→pitch→tilt is not the same as tilt→pitch→yaw. On top of this, each rotation is relative to the camera's current orientation, not to the world frame. This distinction is a large reason for why we have the yaw invarient, and why the yaw invarient should be valued more than the roll/pitch/yaw for validation.

In order to reverse the translations to the original axis, one would have to thus do a negative rotation in the opposite order. The camera pairs orientation is defined as the axis bisecting the forward axis of the two cameras. In other words, in the real world, the forward axis of this is the normal vector out of the bottom of the satellite. Consider the following pair of transformations:

$$\text{T1} = \text{yaw} \rightarrow \text{pitch} \rightarrow \text{tilt}$$
$$\text{T2} = \text{-tilt} \rightarrow \text{-pitch} \rightarrow \text{-yaw}$$

Applying T1 followed by T2 would result in the original orientation.

3 Detumbling

The detumbling system on CHARMS is covered extensively in the original report. This section summarizes the computing squad's theoretical and simulation contributions.

CHARMS's real-time control system autonomously stabilizes the satellite from a state of uncontrolled spin. A counter-torque is generated by its magnetorquers according to the **B-cross Control Law**:

$$\vec{m} = -\left(k \Big/ \left\|\vec{B}_{\text{body}}\right\|^2\right) * \left(\vec{B}_{\text{body}} \times w\right)$$

$\vec{m}$ is the desired magnetic dipole moment that will oppose the current angular velocity measured by the gyroscope in rad/s (w). k is the detumbling control gain, found experimentally. $\vec{B}_{\text{body}}$ is the magnetic field measured by the the magnetometer in Teslas. The procedure for converting the desired magnetic moment to voltage is outlined in Section 4.3.2.

If gyroscopes become unavailable for any reason, CHARMS will fall back to a B-dot Control Law:

$$\vec{m} = -k * \dot{B}_{\text{body}}$$

This simpler alternative estimates the angular velocity by looking at how the local magnetic field is changing over time. It provides a redundant detumbling method in the case of sensor failure.

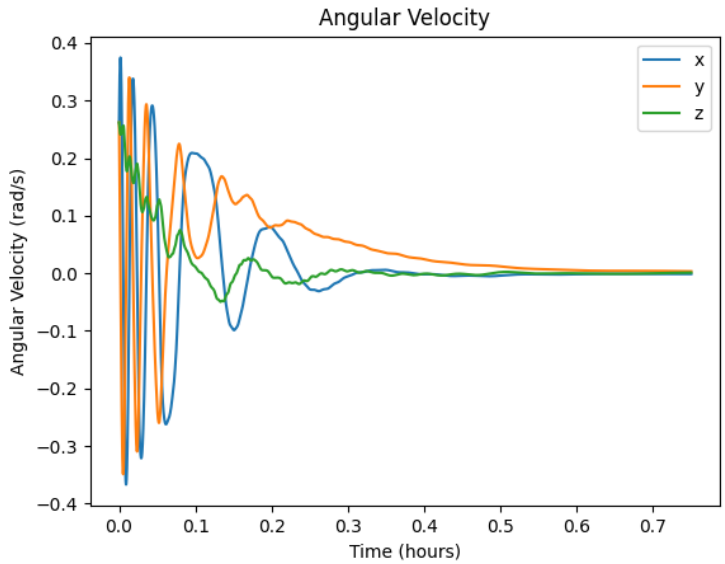

Figure 5. A simulated detumble

Using the simulation architecture described in Section 2, IrishSat thoroughly tested these control laws and verified their compliance with NearSpace's requirements. To recap, their targets were a detumble to < 0.5 degrees/s in under 75 minutes and using less than 7 wH. Our simulations predict that CHARMS will detumble well within these power and time thresholds. While starting at a spinning state of 15 degrees/s on each axis, the B-cross can detumble to ~0 degrees/s in 0.5 hours (see Figure 5). Figure 6 shows that CHARMS power consumption is within NSL's energy bus constraints.

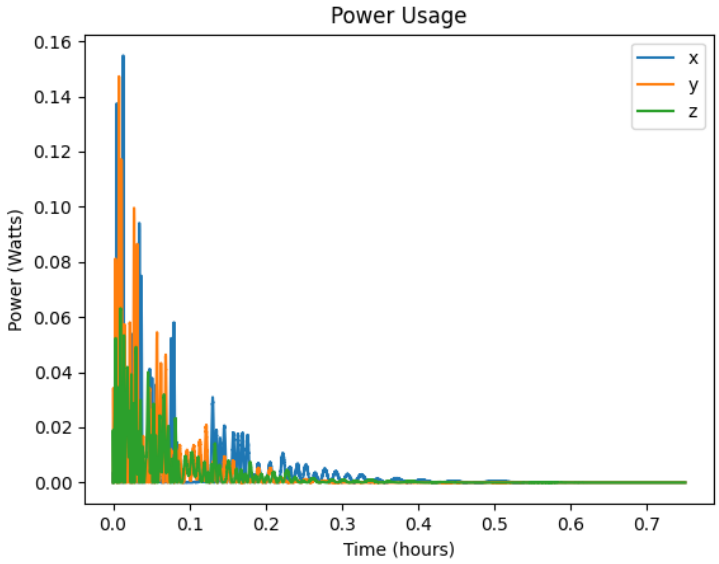

Figure 6. Simulated magnetorquer power consumption during detumbling

We used simulation testing to tweak our gains and estimate controller performance, but all results were tested on the physical system. See the original report for details about the experimental testbed.

Figure 7 highlights the smoothing effect of the B-cross algorithm; note how the orientation jitters randomly at the top of the image before stabilizing as the magnetorquers oppose its motion. This represents the path that a mission utilizing CHARMS's low power detumbling module could take after ejecting from its rocket.

Figure 7. Orientation of CHARMS during detumble represented in Maya

4 Nadir Pointing

4.1 Image Processing

The first step of nadir pointing is to obtain information from our EHS (infrared camera). We implemented a computer vision solution that discovers the Earth's horizon line. Testing on our dataset of the synthetic EHS images (see 2.1), we used the OpenCV library to extract as much information as possible that could be fed into our Attitude Determination system (Section 4.2). This library was selected because it is fast, lightweight, and extensively documented.

Terminology:

- **Pitch**: how high/low our horizon is in our photo compared to the center of the image. Rotation about y axis.
- **Roll**: how tilted our horizon line is. 0 represents a flat horizon. Rotation about x axis.
- **Alpha**: percentage of the photo filled by the Earth.

- **Average edge intensity**: tells us where Earth is on the image. Array of four values from [0-1] representing what percent of each edge is covered by Earth.

From start to finish, our image processing flow is as follows:

1. Normalize infrared temperature values from EHS based on the minimum and maximum temperatures present. Outputs 32x24 pixel byte values, each with a value 0-255.
2. Crop the image so that only uninterrupted pixels are analyzed. Our payload frame blocks some of the top of the image.
3. Smooth image by applying Gaussian blur.
4. Adjust contrast/brightness (optional).
5. Detect horizon edge with Canny edge detection. Canny was chosen for its high accuracy and noise reduction.
6. Check if we are seeing all space or all Earth (see Section 4.1.1).
7. Calculate pixel brightness threshold for space vs Earth. This is the average of the pixels along the Canny horizon line.
8. Find what percent of the total image and each edge is covered by Earth based on threshold value.
9. Fit a linear regression line ($y = ax + b$) to the found horizon line.
10. Find roll from the slope of the line.

    ```
    roll = math.degrees(math.atan2(-a, 1))
    ```

11. Find the midpoint of the image and the detected horizon line.
12. Find pitch by taking the ratio of y-direction offset between the 2 midpoints. Convert to degrees by taking into account the degrees/pixel for our camera.

    ```
    pitch_ratio = (image_midpoint_y - line_midpoint_y) / IMAGE_HEIGHT
    pitch = pitch_ratio * CAM_FOV_VERTICAL
    ```

Figure 8 shows a screenshot of matplotlib with all image transformations. It shows several visuals of different states of the image as it is processed. The blue line on the bottom right shows the edges of the image, the dotted yellow line shows a normal linear fit, and the red line shows the best quadratic fit.

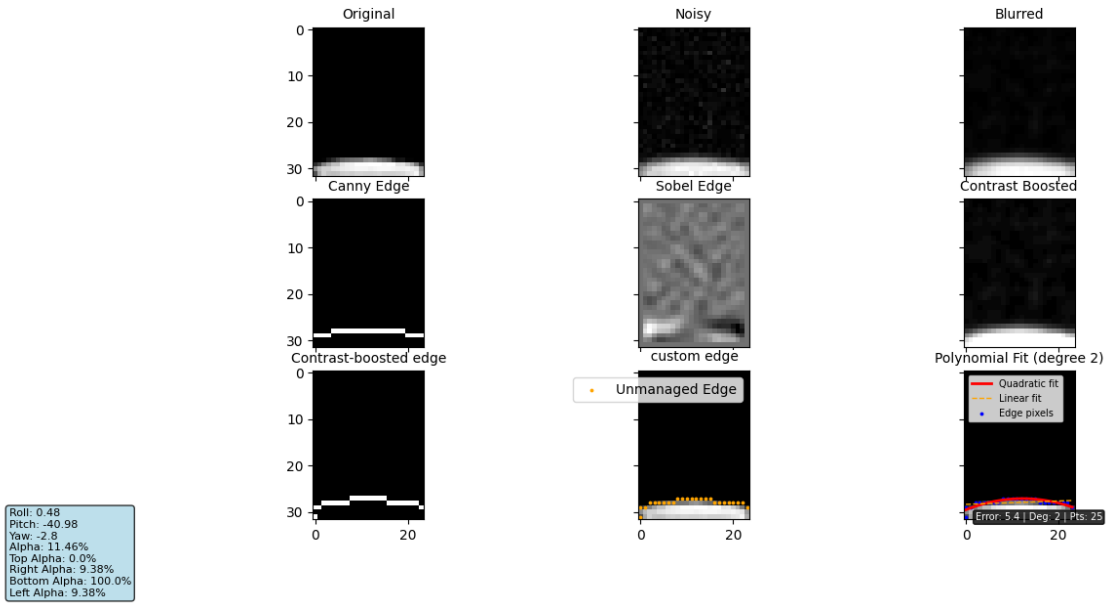

Figure 8. Image Processing Plots

4.1.1 Tracking Over Time

Using an infrared camera for horizon detection is ideal in most circumstances, where there is a clear contrast between heat from the Earth and the temperature void of space. In this case, we use the sensor's alpha value (percent of view-field with a value equal to or greater than the brightness of the detected edge) to determine where the Earth lies (see 4.1). However, when there is no clear contrast, typically in the case where the camera's entire view is either completely Earth or completely space, the lack of an available horizon edge presents a problem.

Since the satellite was developed on Earth in a simulated testing environment, the exact sensor values that correspond to either seeing space or seeing Earth were unknown. Our simulator makes its best guess about what temperature the camera will observe (see 2.1), but we wanted a system that could distinguish between Earth/space frames independently of their exact temperature values.

This method tracks the progression of alpha values over time. Once image processing is initiated, we read and store the five most recent frames from each sensor. As a new frame is read, the oldest frame is deleted and the rest are shifted. We then filter the frames and use only the frames with a detected horizon line to calculate a linear regression with alpha values as the range and with frame number as the domain. We take the slope of that line to determine whether the readings are increasing or decreasing. Once the sensor loses sight of the horizon line, we use the slope reading to determine whether the sensor is likely seeing all Earth or all space. If the value is increasing, we determine that we are likely facing the Earth, and if not, we are likely facing space. For example, in Figures 9-12 below, as the Earth moves out of the sensor's view from Image 1 to Image 4, the algorithm will detect that the sensor is now viewing space, rather than the Earth. This solution allows robust sensor data collection and handles edge cases where the

horizon line cannot be detected.

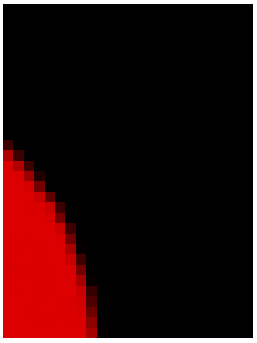

Figure 9. Image 1

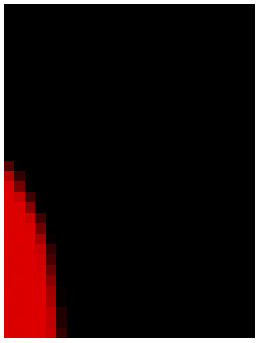

Figure 10. Image 2

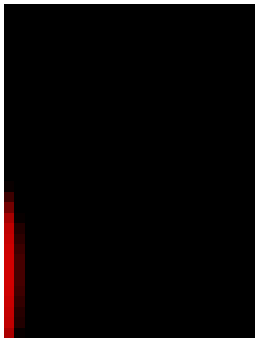

Figure 11. Image 3

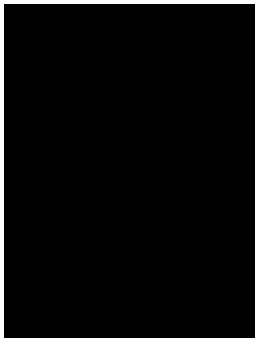

Figure 12. Image 4

4.2 Attitude Determination

After analyzing the EHS images and extracting relevant information, our Attitude Determination (AD) subsystem must use those outputs to find the Earth. Its goal is to find the error quaternion (q_e) that represents the relative rotation required to nadir point.

Over the course of testing, the team realized that there were many more images than could be individually tested. To achieve this, we developed a **case system**. Each case has unique attributes that allow us to test to see if our system works in response to each relative position of the Earth. These cases were decided by all of the different combinations on what the two cameras could see at one instance, as these would require the same transformation for image processing. For example, if one camera picks up the Earth and the other camera did not show anything, this would become a case where all images that also have these attributes were placed in that case. We combed through all of the images from the Maya generation and manually placed them in their respective cases. It allowed us to further analyze our nadir pointing and find any discrepancies when applied to our EHS image dataset (see 2.1).

The goal at this point is to sort images into cases in such a way that roll, pitch, and yaw can be calculated in the same way for images in the same sorted group. The solutions for some cases use both cameras, while others use only one. We sort images into cases numerically based on image processing characteristics, such as certain edge or alpha combinations.

Our analysis yielded 5 cases that encompassed most possible scenarios:

1. The Earth appears at the bottom of both cameras. Figures 13 and 14 show an example of EHS outputs that fall under case 1.

2. One cameras sees all Earth, while the other camera shows a sliver of Earth's curvature.

3. One camera sees only space, while the other shows the Earth upside down at the top of the frame.

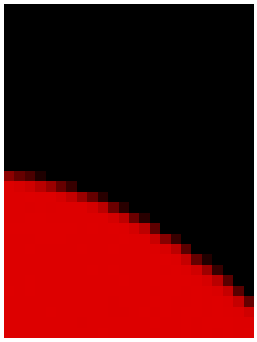

Figure 13. Camera 1 (case 1)

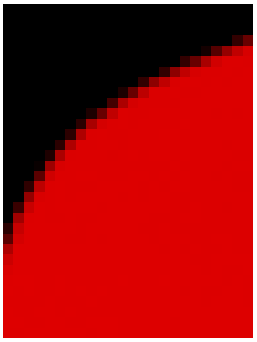

Figure 14. Camera 2 (case 1)

4. The Earth appears sideways in both cameras.
5. The Earth appears in opposite corners of the two cameras.

This case-by-case solution has a 99.65% coverage rate after testing it with all of the generated images from Maya (see Section 2.1). In other words, our system has rules in place to determine CHARMS's attitude 99.65% of the time.

We developed a GUI to easily test our solutions for different cases, as seen in Figure 15. The different features include:

1. The first feature is sorting, where the pictures are organized based on the roll, pitch, and/or yaw that was used when generating that specific image.
2. The main box (seen below) has every individual image, where the user can select specific images they want to analyze together.
3. On the far right, there is the case selector. This allows the user to automatically select all of the images that correspond to the case qualifications.
4. The Analysis Options tab houses the option for how the image processing is computed, using a polynomial degree of 1, 2, 3, or automatically choosing the best fitting one by computing each. It also allows the user to see all of the information on one screen.
5. Finally, the user can also view the quantitative computation results in text.

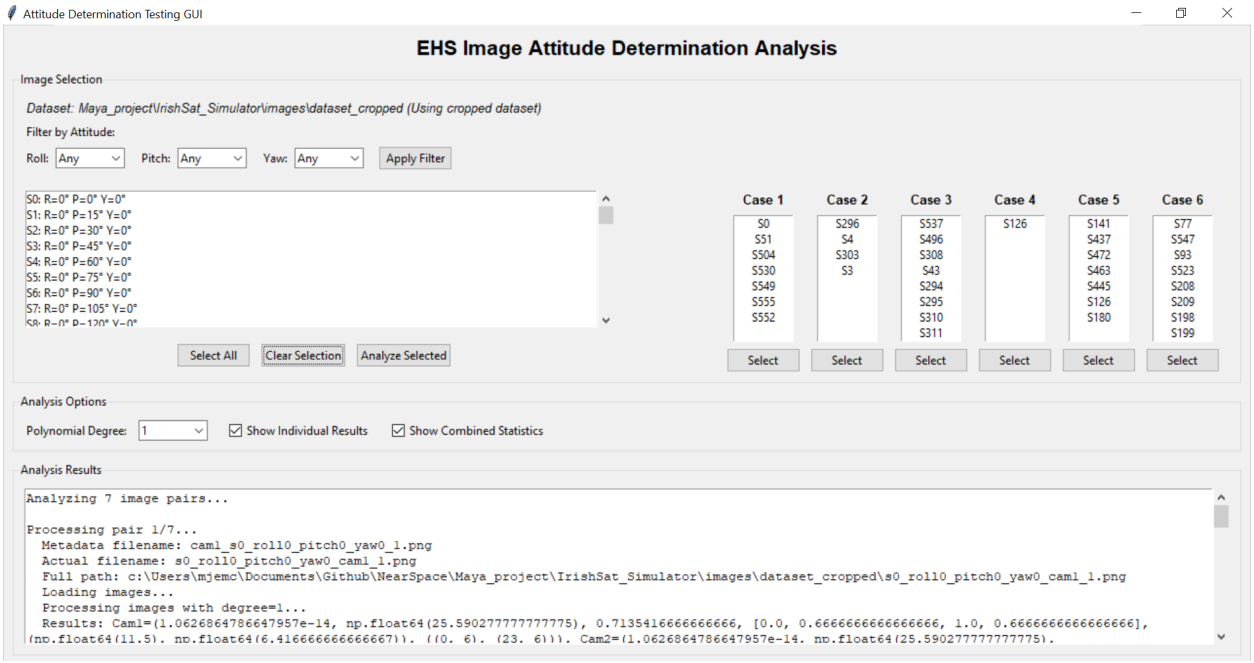

Figure 15. GUI for Attitude Determination case testing

After selecting the cases that the user wants to test, they must press the analyze selected button. This will print the results of the computations of the several images in the Analysis Results tab. It also graphs the relations between the true attitude (according to Maya) and our computed attitude as seen in Figure 16.

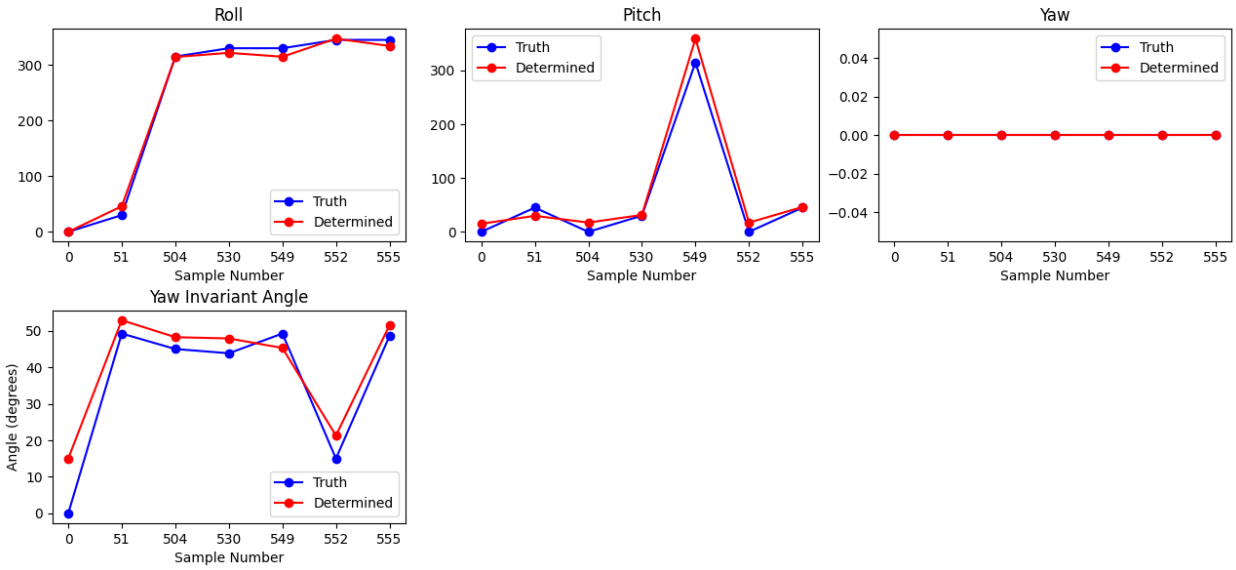

Figure 16. Case 1 Result Graphs

After finding the Euler angles (roll and pitch) that represent the rotation towards the Earth, we

convert them to an equivalent error quaternion q_e. The team decided that yaw cannot be reliably determined from the pair of 2D images, because there are infinite valid yaws that correlate to nadir pointing. There is no specific yaw to shoot for, as the satellite can spin on the Z axis while nadir pointing and the EHS output will be the same. Regardless of which case we are in and how the Euler angles were calculated, this AD system always outputs a consistent error quaternion that our Attitude Control can act on to orient us towards nadir.

4.3 Attitude Control

After our Attitude Determination subsystem (Section 4.2) has found an error quaternion that represents the rotation towards nadir, our Attitude Control subsystem uses magnetorquers to orient us towards it. This system is severely limited by the physics that govern magnetorquers.

4.3.1 2-Axis Control

The primary concern for magnetorquer-only ADCS is the lack of 3-axis control. Magnetorquers work by "pushing off" of existing magnetic fields according to the following equation.

$$\vec{L} = \vec{m} \times \vec{B}_{earth}$$

Because of cross product between magnetic moment and the magnetic field of the Earth, CHARMS cannot have control over the axis that is aligned with the magnetic field. The cross product of parallel vectors is always zero; therefore, if a torquer is parallel with Earth's magnetic field, it cannot generate any torque at that moment. Our system is inevitably underactuated; even though the satellite has three degrees of freedom rotation, we can only ever torque around the two axes that are perpendicular to the magnetic field vector.

At any given time, we have no way of knowing which axis we don't have control over. The axis that cannot be actuated around is defined in inertial coordinates, not spacecraft coordinates. As discussed in Section 1.2, the lack of GPS makes it impossible to establish an absolute (inertial) orientation. Without it, our magnetometer can only provide information about the magnetic field in the body frame.

The reality of 2-axis control means that CHARMS must be cautious about wasting power while trying to actuate on an uncontrollable axis. We cannot exceed 0.5 Watts during nadir-pointing operations. The system takes a **best-effort approach**. It constantly uses a PD controller (see Section 4.3.2) to tweak orientation towards the target, accepting the fact that such torques might be impossible to generate. Thankfully, the high orbit speed found in LEO (90 minutes for a full orbit) ensures that the Earth's magnetic field is always changing. If the magnetic field makes it impossible to rotate as we would like at any given time, we can simply try again until control over that axis is regained. Simulation results (see Section 2) indicate that control can be regained within an orbit when the Earth's magnetic field momentarily aligns with a critical axis. CHARMS uses constant, low-power, best-effort control signals to ensure that attitude control goals are met given enough time.

4.3.2 PD Controller

We elected to keep our controller simple and let the Attitude Determination system perform the heavy lifting. According to chapter 7.2 of *Fundamentals of Spacecraft Attitude Determination and Control*, the simplest feedback controller is a Proportional Derivative (PD) with the following form:

$$\vec{L} = -k_p * q_{e1:3} - k_d * \omega$$

$\vec{L}$ is the torque along each axis that will orient us towards the given error quaternion q_e. k_p and k_d are the proportional and derivative gains, respectively. The proportional gain represents how strongly to orient us towards the target (as represented by the X, Y, Z components of the error quaternion). The derivative gain opposes angular velocity (ω), preventing overshooting. Our process to tweak these values is outlined in Section 4.3.3. The integral gain was omitted because the nadir state is unlikely to be steady enough for steady state error to be relevant.

This PD controller was chosen because it could constantly make best-guess nudges towards our nadir while preventing excessive angular velocity. It is a robust, tried-and-tested solution that we could rely on to provide our magnetorquers with best-effort control inputs.

After finding the desired torque that would orient CHARMS towards nadir, we must calculate the associated magnetic moment that will generate that torque. That relationship is defined below.

$$\vec{m} = \vec{B}_{\text{body}} \times \vec{L}$$

$$\vec{m} = \vec{m} \Big/ \left\| \vec{B}_{\text{body}} \right\|^2$$

Different B-fields will produce different torques from the same magnetic moment (m). The inverted cross product with the magnetic field in the body frame (from our magnetometer) accounts for this. However, cross products cannot be inverted perfectly. The magnetic moment will be accurate when the magnetic field and desired torque are orthogonal, but it will be off by some factor when they are not. This method has inherent variability based on the magnetic field, but we could not figure out a more stable way to find the desired magnetic moment. Sometimes during operation, our outputted torque will be different than the desired torque calculated by the PD controller.

From the desired magnetic moment, we can find the actuator input (voltage) based on the physical characteristics of our magnetorquers.

$$I = \vec{m} \,/\, (n \vec{A} \epsilon)$$

$$V = IR$$

ϵ is a magnetizing factor that depends on the physical characteristics of our air or ferromagnetic-core magnetorquers. I is current, n is the number of wire turns/wraps, $\vec{A}$ is area (and orientation of the torquer), V is voltage, and R is resistance.

4.3.3 Finding Gains

Our PD controller gains were found experimentally. First, we pre-generated PySOL setups (see 2) at a wide variety of starting locations and orientations. We ran about 50 simulations with their own Kp and Kd values on each. We iteratively narrowed down which combination of gains was optimal by seeing which had the smallest magnitude of the error vector on average. Since we were not sure where the satellite would be in LEO, we tested across a variety of orbits in the 300-600 km range to avoid any assumptions in that regard. Our ADCS was system was still under development at the time, so we tested the controller with perfect Attitude Determination knowledge (supplied by the simulator). We were forced to assume that this generalizes to our AD method outlined in Section 4.2. Although time consuming, we were able to find suitable error values (Kp= 1e-2, Kd= 1e-3).

5 Firmware

5.1 RTOS

CHARM's Real Time Operating System (RTOS) is overviewed exhaustively in the original report, so please refer to it for most details. Below is a list of Computing Squad's clarifications, updates, and observations from our time with the project.

- A new task (`taskNadirPoint`) that implements the ADCS for nadir pointing as described in Section 4. See Table 2 for more information.
- Cameras are polled every 7 seconds, as we do not need updated information for Attitude Determination very often.
- We split camera data into its own struct and queue, as it does not need to be polled in tandem with IMU data.
- All downlinks are queued until detumbling is complete (which will always take 30 minutes unless a ground command is sent).
- For uplinks, sequence numbers and data length are handled by the NearSpace bus. We do not check for any specific start number (or data length), and we also only check that the current sequence number is not equal to the previous one seen. NSL pads the data length to a minimum of 8.
- The state machine task does not request Iridium messages (see Section 5.2.2).
- The second byte of the "send downlink" ground command is only checked to be `IMAGE` (0x34), and everything else is treated as a health and safety request (not 0x31 specifically).

5.2 Communication

CHARMS is designed for autonomous ADCS, but the NearSpace mission is proctored as a demonstration of functionality. For this reason, command and control is required from the ground. For example, NearSpace must tell us when they have induced a spin and are ready for us to test our detumble. For an overview of the proprietary communication protocol between the NSL bus and our flight computer, please see the original report.

5.2.1 Uplink Packets

See notes above about how the uplink header is handled by the NearSpace radio/bus. Our uplink sequences should start with one of the following ground commands in Table 2.

Table 2. Supported Commands

Ground Command	Function Byte	Description	Secondary Byte
Downlink	0x30	Manually request a downlink packet from the CHARMS payload. It can be one of two types: Health and Safety or Image. Health and Safety contains state information, the most recent IMU data, and a timestamp. Note that these are also downlinked automatically once an hour.	0x31 (health and safety)
		While in nadir pointing mode, request an image from camera 1. Pixel data from our 24x32 cam will be split across 4 downlink packets.	0x34 (image)
Detumble	0x31	This will change the payload from any state into the DETUMBLE state. The control algorithm will begin sending actuation signals which will drive magnetorquers to reduce angular velocity. Detumble packets accumulate during operations and will send automatically once completed (120 packets after 30 minutes). Note that no iridium downlinks can be sent while in detumble mode (including HS).	

Continued on next page

Ground Command	Function Byte	Description
Nadir Point	0x32	Activates a separate control algorithm which points the satellite's normal vector directly at Earth. This is the only way to start polling our IR cams. While nadir pointing, images (across 4 packets) will be downlinked automatically every 15 mins. Lasts for 3 hours.
Safety	0x33	This puts the payload into the SAFETY state, turning everything off and waiting to be power cycled by the bus for a full-system reset. We stop polling sensors and checking for uplinks from the NSL bus.
Stop	0x34	This takes the payload out of any state and places it in the IDLE state. This will manually turn off the magnetorquers, cameras, and control scripts.
Flip Voltage X	0x35	In case of backwards hardware installation, it is possible that our magnetorquers speed up the spin of the satellite instead of slow it down. This allows for a software fix in case there are hardware problems.
Flip Voltage Y	0x36	Same as 0x35, different axis.
Flip Voltage Z	0x37	Same as 0x35, different axis.
Flip IMU	0x38	There are 2 redundant IMUs on the CHARMS PCB. This allows the operator to toggle between either IMU in case one malfunctions due to radiation.
Reset Queues	0x39	This resets all RTOS data transfer queues in case they become misconfigured during operation.
Toggle Data Frequency	0x40	Not implemented. Supposed to toggle between high and low data rate packets.
Disable X Voltage	0x41	Turn off the magnetorquer on the X axis by zeroing out its command signals. Sending this command again will turn it back on.
Disable Y Voltage	0x42	Same as 0x41, different axis.

Continued on next page

Ground Command	Function Byte	Description
Disable Z Voltage	0x43	Same as 0x41, different axis.
Set Detumble Packet Creation Time	0x44	How often detumble packets (which contains data from 8 timesteps) are generated. Note that they are all sent after detumbling, so this only effects their creation frequency (and total number). 4 bytes (unsigned) should be sent. 179900 ms by default.

5.2.2 Downlink Packets

In accordance with the NearSpace Interface Control Document (ICD), our payload can make requests to the bus. See Table 3 for NSL request packet headers.

- CHARMS makes a "Check Buffer for Uplink Data" request every 5 seconds, which is how our payload queries NearSpace's bus for ground commands. This communication also prevents our payload from being power-cycled.
- Our payload does not check for Iridium uplinks ("Check Network for Uplink Data"), electing to let the bus check for new messages and place them in the buffer on its own timeline.
- To construct a downlink, our payload makes a "Send Payload to Ground" request. Our Health and Safety (Table 5), image (Table 7), and detumble results (Table 6) packets all give a payload length followed by 200 payload/filler bytes.

Table 3. NearSpace Request Packet Structure

Byte Range	Field Name	Description
Bytes 0–2	Sync Bytes	Three fixed synchronization bytes (0x50 0x50 0x50)
Byte 3	Request Type	Ask the bus to check for new messages (0x48), query the Iridium network for uplinks (0x47), or downlink payload data through Iridium (0xF5).

Continued on next page

Byte Range	Field Name	Description
Byte 4	Payload Length	Number of bytes following the length field. All 200 subsequent bytes are sent serially to the NearSpace bus. For downlink requests, only the amount specified here will be sent over Iridium to the ground.

Each IrishSat downlink packet begins in the same way (Table 4). The 4 NSL header bytes are not included in downlinks, so this is the start of the info that is received on the ground.

Table 4. IrishSat Packet Header

Byte Range	Field Name	Description
Bytes 5–6	Packet Sequence Number	16-bit global packet sequence counter (big-endian). Increments for every downlink packet.
Byte 7	Packet Type	Type of data contained in payload. • 0x31: health and safety. • 0x32: detumble results (operational data). • 0x34: infrared image data.

Health and safety packets are downlinked automatically every hour and on command. However, see Section 7 for complications.

Table 5. Health and Safety Packet Structure

Byte Range	Field Name	Description
Byte 4	Payload Length	0x6A = 106 bytes of Health and Safety data.
Byte 7	Packet Type	0x31
Bytes 8–11	Timestamp	Sensor data timestamp (milliseconds).
Bytes 12–17	Acceleration (X,Y,Z)	IMU accelerometer readings (mg, signed 16-bit).
Bytes 18–23	Gyroscope (X,Y,Z)	IMU angular rate data, scaled to 0.01 deg/s.

Continued on next page

Byte Range	Field Name	Description
Bytes 24–29	Magnetometer (X,Y,Z)	Magnetic field vector measurements (microteslas).
Bytes 30–31	State Machine Delay	State machine delay duration (ms).
Bytes 32–33	State Machine Delay Ticks	RTOS tick count for state delay.
Bytes 34–35	Sensor Poll Delay	Sensor polling interval (ms).
Bytes 36–37	Sensor Poll Delay Ticks	RTOS ticks between sensor polls.
Bytes 38–39	Camera Poll Delay	Camera polling interval (ms).
Bytes 40–41	Camera Poll Delay Ticks	RTOS ticks between camera polls.
Bytes 42–43	Uplink Interpret Delay	Delay before uplink interpretation.
Bytes 44–45	Uplink Interpret Delay Ticks	RTOS ticks for uplink delay.
Bytes 46–47	Detumble Delay	Detumble task delay duration.
Bytes 48–49	Detumble Delay Ticks	RTOS ticks for detumble delay.
Bytes 50–51	Send/Ack Delay	Delay between send and acknowledge.
Bytes 52–53	Send/Ack Delay Ticks	RTOS ticks for send/ack delay.
Bytes 54–55	Command Monitor Delay	Delay between command monitoring cycles.
Bytes 56–57	Command Monitor Delay Ticks	RTOS ticks for command monitoring.
Bytes 58–61	Iridium Request Time	Timestamp of last Iridium request.
Bytes 62–65	Uplink Request Time	Timestamp of last uplink request.
Bytes 66–69	Iridium Packet Frequency	Configured Iridium downlink frequency.
Bytes 74–75	Packet Sequence (Repeat)	Repeated packet sequence number.

Continued on next page

Byte Range	Field Name	Description
Byte 76	Detumble Sequence Number	Detumble operation counter.
Bytes 77–80	RTOS Start Time	System start timestamp.
Bytes 81–83	State Variables	Current, previous, and incoming system state.
Bytes 84–91	Control Flags	Boolean flags to dictate our state (sensor polling, detumbling, nadir pointing, voltage flips, which IMU is active).
Bytes 92–95	HS Downlink Time	Timestamp of last HS downlink.
Bytes 96–99	Detumble Start Time	Timestamp when detumble began.
Bytes 100–103	Detumble Desired Runtime	Target detumble duration.
Bytes 104–106	Voltage Disable Flags	3 per-axis voltage disable flags.
Bytes 107–110	Status Flags	High-frequency mode, previous packet sent, retransmission, and sensor update flags.

Operational data gathered during detumbling is downlinked after the mode has concluded. 8 timesteps are packaged into a single packet. The frequency of their creation can be changed by an uplink command (see Table 2). Which of the two IMU measurements are taken from can also be changed on command.

Table 6. Detumble Results Packet Format

Byte Index	Field	Description
4	Packet Length	0xB3 (179 bytes).
7	Packet Type	0x32
8–183	Detumble Sensor Payload	Eight detumble sequences, each containing 22 bytes of time-stamped IMU data (176 bytes total). The packet is transmitted only after all eight sequences have been populated.
Detumble Sequence Structure (22 bytes each)		

Continued on next page

Byte Index	Field	Description
0–3*	Timestamp	Timestamp corresponding to the sensor measurements in this sequence.
4–9*	Accelerometer X, Y, Z	A signed 16-bit integer from accelerometer for every axis.
10–15*	Gyroscope X, Y, Z	A signed 16-bit integer from gyroscope for every axis. Scaled by 100 (deg/s).
16–21*	Magnetometer X, Y, Z	A signed 16-bit integer of magnetic field on each axis (microteslas).

Images are sent across 4 packets. They are downlinked every 15 minutes during nadir pointing and on command.

Table 7. Image Downlink Packet Structure

Byte Range	Field Name	Description
Byte 4	Payload Length	0xC5 = 197 bytes of image-related data.
Byte 7	Packet Type	0x34
Byte 8	Image Info	Bit-packed image metadata: • Bit 7: Camera select (0 = camera 1, 1 = camera 2) • Bits 6–5: Image chunk index (0–3) • Bits 4–0: Reserved
Byte 9	Image ID	8-bit image counter incremented for each newly captured image. Used to associate the four packets belonging to the same image. Wraparound is permitted.
Bytes 10–201	Pixel Data	192 bytes of grayscaled pixel intensity (0–255) that make up a quarter of an image. Pixels are flattened in row-major order and segmented deterministically based on the chunk index.

6 What We Learned

As IrishSat's first time writing flight software, the NearSpace project proved to be very informative. This section outlines some of the roadblocks that we ran into and how the team will be better prepared for the next mission.

Time management played a major role in the project's outcome. As a hobbyist club of college students, working hours are not guaranteed. This was compounded by nebulous and shifting deadlines. Our workflow schedule was upended several times as software integration testing due dates were changed. Unfortunately, our theory and implementation teams were forced to publish work with an unsatisfactory level of polish. Section 7 outlines the features that were cut due to time constraints. For future missions, IrishSat must confirm 3rd party deadlines as early as possible in order to schedule internal checkpoints accordingly.

IrishSat faced many issues with both external and internal inherited dependencies. NearSpace's integration testing was a multi-day process where we learned how to interface with their proprietary communication protocol. We had to grapple with differences between their provided simulator and their Engineering Model (EM), which ended up being due to a typo in the interface control document (ICD). We learned that some level of friction is inevitable while interfacing with external organizations.

Other internal issues arose from the fact that this project was spread across multiple years. Seniors who started the project graduated, leaving the computing squad with a "legacy" codebase of sorts. IrishSat underestimated the time it took to refactor and debug this inherited dependency. We ran into issues like a lack of proper timeouts, RTOS tasks with incorrect priorities, debugging tasks left in production, and malformed packet creation. In mission critical systems, it's dangerous to trust that your predecessor caught every bug. In a college setting (with a rapidly turning-over student body), this burden is prevalent and cannot be overlooked. IrishSat's solution going forward is thorough internal integration testing.

Similarly, the NearSpace project showed the importance of communication during transition periods. For example, we showed up to integration testing unaware that our payload's IR cameras were sitting back in our workshop after being dismantled during a previous visit. We also saw that the aircore had been disssasembled improperly. Graduating members leave more than software; they're forced to abandon half-finished plans and ideas. If they don't properly document their progress, it's lost to the next generation. As the club matures, it must forge strong transition protocols that keep its members and outgoing leadership on the same page.

In addition to the software and project management complexities, lessons were learned from the mechanical aspects of the project. The following sections will detail these findings. During integration and testing the wires that connected the magnetorquers to the printed circuit board (PCB) were routinely getting snagged and failing. The solution was to connect stronger wires to these and clamp the stronger wires. While this fulfilled the primary mission goal, the proper solution to this problem would have been to make a wire harness and restrain that wire harness with clamps.

During the design phase of the payload structure a vibration and loading analysis should have been conducted. The team did not have the technical background nor the time to conduct these simulations but it would have been a large asset in proving that the structure would not fail during launch. The suggested process for this would be to use Ansys Mechanical, and contacting Professor Maria Warren for assistance as needed, to verify that the structure would sustain worst-case loading and not have fundamental frequencies close to the launch vehicle forcing frequencies.

In addition to this, it was recommended by NearSpace Launch personnel that all fasteners be staked with an epoxy compound (only when the assembly is completely finished). This will limit the chance that fasteners come loose due to vibration and become foreign object debris (FOD) within the spacecraft.

The largest hardware takeaway is the importance of triple checking payload assembly and functionality. At integration testing, we found a magnetorquer bolted down incorrectly and a slew of issues with the IR cameras' assembly. They were soldered incorrectly, lacking protective tape, and not both being picked up by our ESP32's I2C scanner. We had tested connectivity with one camera at a time, but not both at once; they actually shared an unchangeable I2C address. We were saved by EEPROM editing expertise within the NearSpace engineering team. The team has resolved to add several layers of unit testing and verification to every step of future payload assembly processes.

Lastly, the NearSpace mission illustrated the importance of good parts selection. Future iterations would benefit from thicker magnetorquer wires and higher quality sensors. COTS components with more space-grade temperature ratings would also be ideal. The noise levels of the chosen IMU provides a major obstacle for our ADCS that could be avoided with only marginally higher costs. Low cost hardware was a major goal of the project, but we believe the team sacrificed too much mission effectiveness for cheap sensors. For future missions, IrishSat must invest their limited resources wisely to balance cost and usefullness.

7 Future Improvements

There are many flight software upgrades that we were unable to implement due to time constraints.

- Various **packet structure fixes**. The Health and Safety length was set too low, causing us to lose some data. It also has some redundant data and lacks some flags that were added later in development. The detumble results length was set too high. A timestamp could be added to image packets. While parsing uplinks, data length received should be checked to prevent out of bounds reads. See Section 5.1 for other details about CHARMS's RTOS implementation.
- Improve the **theoretical backbone of our ADCS**. With more time, our team would ensure the correctness of Euler-to-quaternion conversions, improve determination methods for each case, generate and test upon noisier EHS images, filter data more thoroughly

(Kalman filter instead of low pass filter), and explore better 2-axis controllers. The PD controller (see Section 4.3.2) is simplistic and doesn't consider previous states to guide future actuation. Future work would also involve verifying CHARMS's nadir pointing accuracy and power budget in simulation.

- Thorough **unit testing** for C++ image processing, image cropping/rotations, moment to voltage conversion, and quaternion math.
- Option that **disables either or both cameras**. If one becomes disconnected (or we uplink a command to disable one), backup attitude determination algorithms that use one camera could be enabled. Initial sensor connection (and subsequent polling) should have watchdog timeouts added to prevent infinite waiting on an unresponsive peripheral. If one cam is unresponsive, camera polling should still populate queue with partial data.
- **Pull sensor data from queue correctly** when auto-creating the periodic health and safety packet. Such an issue would be circumvented by better function modularization.
- Turn **magnetorquers off while polling magnetometer** for more accurate readings.
- Better **infrared temperature normalization**. The current temperature-to-brightness conversion normalizes by the highest and lowest seen temperatures in every frame (see 4.1). We hypothesize that this will result in a splotchy mess for images that are all Earth or all space (although the metal edge at the top of the frame may provide enough stability during normalization). To improve this, the system could dynamically determine a "typical" min/max range to normalize by. It could track temperatures seen over time to understand what brightness Earth and space pixels should be. This would also remove the need for alpha tracking over time (see 4.1.1).
- Check **sensor health** more accurately during detumble (Section 3) runtime. This would allow us to better detect gyroscope failure and switch to our backup control law.
- Uplink command that **flips the orientation and axis of either camera**. This would allow us to fix any discrepancy between code and actual mounting configuration. This is already somewhat covered by the `flipVoltageX` command.
- Adhere to NASA's space computing guidelines (like **The Power of 10** and **Test Like You Fly** (TLYF).
- Implement **fault tolerant** software and hardware mechanisms to detect and correct Single-Event Upsets (SEU's).

8 Conclusion

The CHARMS (Control of Hardware Attitude using Reliable Magnetorquers Satellite) payload represents a major milestone in accessible, autonomous attitude control for small satellite platforms. CHARMS demonstrates a novel magnetorquer-only ADCS that operates within the low

size, weight, power, and cost (SWaP-C) constraints of a 0.5U Cubesat. IrishSat's computing squad demonstrated that software solutions can achieve detumble and nadir pointing without a GPS or reaction wheels.

A robust orbital simulation suite was developed to test IrishSat's proposed control algorithms. The B-cross controller's success on physical testbed setups was informed by systems modeling. The testing architecture will be invaluable as the club develops future flight software and is open sourced here.

In addition to simulation-verified detumbling, the computing squad seeks to demonstrate a novel method for nadir pointing. By only using an IMU and two cheap IR cameras, CHARMS hopes to simplify the hardware requirements for Earth communications. Our black-box Attitude Determination method leverages comprehensive feature extraction to find the orientation of CHARMS with respect to the Earth. The Attitude Control system uses a low power PD controller (with gains optimized in simulation) that combats the underactuation of magnetorquers.

As a plug-and-play ADCS built on commercial, off-the-shelf (COTS) components, CHARMS seeks to democratize space for educational programs, low-budget smallsats, and commercial tech demonstrators seeking low-cost, effective detumbling and nadir pointing.