Final Report For RockSat-X Payload - Hephaestus

Helena Bales, Amber Horvath, and Michael Humphrey

CS463 - Spring 2017

June 8, 2017

Abstract

The Oregon State University (OSU) RockSat-X team shall be named Hephaestus. The progress of our project shall be outlined in this document. The mission requires that the payload, an autonomous robotic arm, perform a series of motions to locate predetermined targets. The hardware shall be capable of performing the motions to reach the targets. The software shall determine the targets and send the commands to the hardware to execute the motion. The combination of the hardware controlled by the software shall demonstrate Hephaestus's ability to construct small parts on orbit.

Hephaestus Mission Logo

Approved By - Dr. Nancy Squires _____ Date _____

Approved By - Helena Bales _____ Date _____

Approved By - Amber Horvath _____ Date _____

Approved By - Michael Humphrey _____ Date _____

Contents

1 Introduction

The Hephaestus Payload is a rocketry payload that will fly onboard the 2016-2017 RockSat-X rocket. The rocket will be launched from Wallops Flight Facility filled with student-made payloads. The Hephaestus payload will be made up of a deployable arm and a video camera. The arm will perform a series of motions that will be recorded by the video camera and sensors. Following the experiment, the arm will retract back into the rocket. The Hephaestus mission will be Oregon State University's first space mission and will prove not only our ability to develop a space-ready payload, but also the viability of construction in space using a robotic arm.

1.1 Document Overview

2 Project Overview

2.1 Project Purpose

The Oregon State University RockSat-X team will demonstrate that an autonomous robotic arm can locate predetermined targets around the payload under microgravity conditions by using precise movements. The technical actions performed by this demonstration will illustrate a proof of concept for creating assemblies, autonomous repairs, and performing experiments in space.

2.2 Mission Success Criteria

2.2.1 Minimum Mission Success Criteria

2.2.2 Maximum Mission Success Criteria

2.3 Concept of Operations

2.4 Programmatics

2.4.1 Organizational Chart

2.4.2 Sponsors

3 Requirements Document

3.1 Original Requirements Document

3.2 Introduction

3.2.1 Purpose of Document

This document shall describe in detail the Hephaestus RockSat-X payload. It shall specify the software behavior of the payload. This document will not discuss the specific implementations of the hardware or the software. It will specify the behavior by describing the Functional and Non Functional requirements of the software. This document will be updated throughout the project and should be considered a living document.

3.2.2 Overview of Document

This document will first cover the functional requirements of the project, then the non functional requirements. The Functional Requirements will include descriptions of the main behavior, target generation, movement, operation modes, and telemetry. Each of these topics will include descriptions of the behavioral requirements for each. The Non Functional requirements will cover the performance, security, and telemetry. Each of the non functional topics covered will include the requirements for the quality of each of the topics.

3.2.3 Overview of Payload

The Hephaestus RockSat-X payload is a deployable rocketry payload that will fly on the 2016 RockSat-X launch. The payload's main function is to provide a proof of concept for delicate construction in a space environment. The Hephaestus payload shall perform the following operations:

- Remain retracted with power off for duration of launch
- Power on at apogee

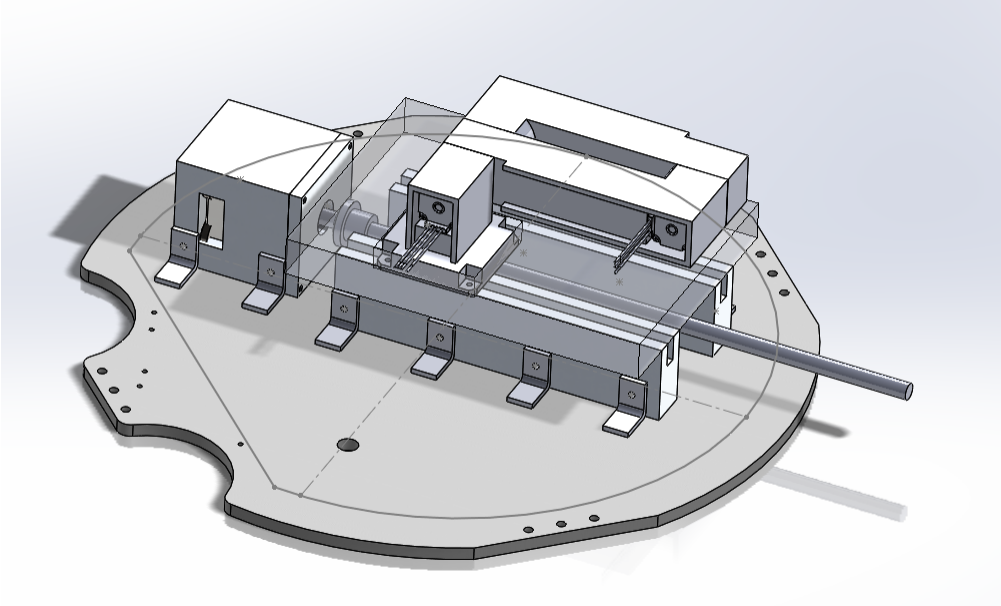


Figure 1: Model of Hephaestus Payload

- Deploy arm assembly body
- Deploy arm
- Perform 360 degree sweep with video camera
- Generate targets for arm motions
- Perform arm motions
- Record each arm motion with video camera
- Retract arm
- Retract arm assembly body
- Power off

3.2.4 Overview of Physical Payload

While this document focuses on the software of the Hephaestus payload, the project also includes hardware and electrical systems. Understanding the physical appearance of the payload will help with understanding the software system. As such, Figure 1 should serve as a reference for the physical appearance of the payload.

3.2.5 Mission Success Criteria

The following criteria determine if the Hephaestus mission will be considered successful post-flight. The minimum mission success criteria represent the lowest criteria to be met in order for the mission

to be considered successful. If the minimum mission success criteria are not met, then the mission may not be considered successful. The maximum success criteria define the highest goals for the mission. Fulfilling any or all of these criteria, in addition to the minimum success criteria, would constitute a highly successful mission. The success of the mission shall be evaluated by means of video recordings recovered post-flight and telemetry data received during the flight.

3.2.5.1 Minimum Mission Success Criteria

- The arm assembly body shall deploy and a video sweep is successfully recorded.
- The arm assembly body shall be fully retracted after data collection.

3.2.5.2 Maximum Mission Success Criteria

- The arm assembly body shall deploy and a video sweep is successfully recorded.
- The arm shall make contact with predetermined targets around the payload.
- The camera shall record all instances of contact between the arm and the targets.
- The arm assembly body shall be fully retracted after data collection.

3.2.6 Requirements Apportioning

3.2.6.1 Priority 1

This is the highest priority level. In order for the software system to be considered complete and ready for launch, all requirements of this level must be met. The completion of only Priority 1 requirements marks the completion of Minimum Mission Success criteria, as defined in subsection 1.4.

3.2.6.2 Priority 2

Requirements of Priority 2 are not required for the release of the software system. Not completing these requirements must not present a risk to mission success. The completion of these requirements and successful performance on orbit marks completion of part of the Maximum Success Criteria, as defined in subsection 1.4.

3.2.6.3 Priority 3

Requirements of Priority 3 are not required for the release of the software system. Not completing these requirements must not present a risk to mission success. Completion of all priority 3 requirements and those of higher priority, with successful performance on orbit, marks the completion of the Maximum Mission Success Criteria, as defined in subsection 1.4.

3.3 Functional Requirements

3.3.1 Main Behavior

Priority 1: The software shall control the movement of the arm assembly body to make contact with the payload base at locations generated by the Software (subsection 2.2).

3.3.2 Target Generation

Priority 1: The software shall generate points to be used in testing the Hephaestus arm. The points will constitute the total test of the arm, and should therefore include points representative of standard and edge cases. The points shall be generated in polar form, including an angle from normal, a radius, and a height. The angle shall be in the range of 0 and 359 degrees. An angle of zero degrees shall be in the direction of payload deployment. The radius shall be the distance from the arm's attachment to the base to the generated point. The height of the point, for the purpose of target generation, shall be constant. However the points will always be stored in a triple of angle from normal (θ), radius (r), and height (h). These points shall be used as targets for the arm body.

3.3.3 Movement

The software shall control the movement of the arm body assembly. The position of the tip of the arm shall be tracked in the coordinate notation described in subsection 2.2 above.

Priority 1: The software shall rotate the arm body assembly in a full 360 degrees.

Priority 2: The software shall additionally control the movement the height of the arm body assembly. The arm should descend and touch the baseplate of the payload at any rotation.

3.3.4 Modes

During the course of the flight, the software will progress though several different modes of operation.

3.3.4.1 Launch

Priority 1: The software shall remain idle during launch.

3.3.4.2 Deployment

Priority 1: The software shall power on the arm assembly body and video camera. The software shall begin saving the video feed from the camera to a persistent storage location. The software shall generate target points, as defined in subsection 2.2.

3.3.4.3 Science

Priority 1: The software shall collect data to serve as a proof-of-concept for construction of structures in flight.

3.3.4.4 Safety

Priority 1: The software shall ensure that the arm assembly body can be fully retracted after completing the mission. The software shall, in case of a failure, eject the arm to prevent damage to the arm assembly body and the rocket during descent.

3.3.4.5 Observation

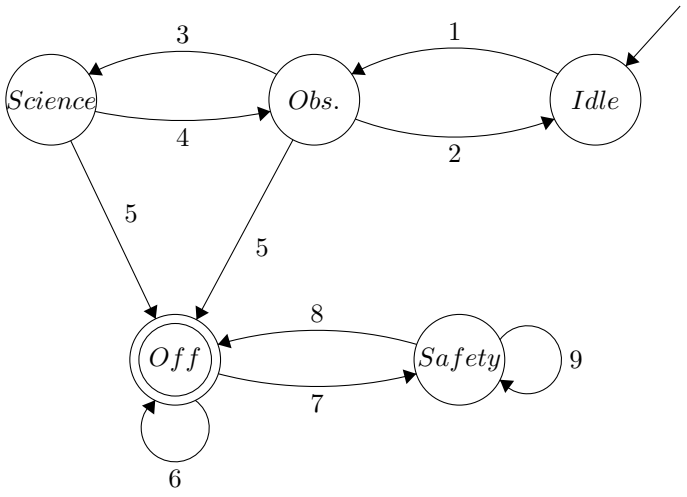
Priority 1: The software shall report all telemetry data (as defined in 2.5) to the ground station.

Priority 2: The software shall be responsible for turning the camera on and off.

3.3.4.6 Power Off

Priority 1: The software shall power down all subsystems of the payload in preparation for descent.

3.3.4.7 State Diagram



State diagram for transition between operational modes.

3.3.5 Telemetry

Let the telemetry interface be defined as 5 of the ten analog pins provided by Wallops Flight Facility. Let telemetry be defined as the data transmitted from the payload to the ground station via the telemetry interface. The software shall report all telemetry data to the ground station.

Priority 1: The software shall report via telemetry all the target points it generates, as defined in subsection 2.2. The software shall also report which code branch it takes to facilitate debugging and post-mortem analysis, if necessary.

3.4 Non Functional Requirements

3.4.1 Performance

Priority 1: The system shall perform efficiently. The maximum response service time should be long enough for the robotic arm to move from one target to another. The system should have a maximum throughput that allows for processing of input arguments about the arm's actions and processing for the telemetry data output. Resource usage should be limited to account for the storing of telemetry data. Power consumption must be limited to 28V.

3.4.2 Security

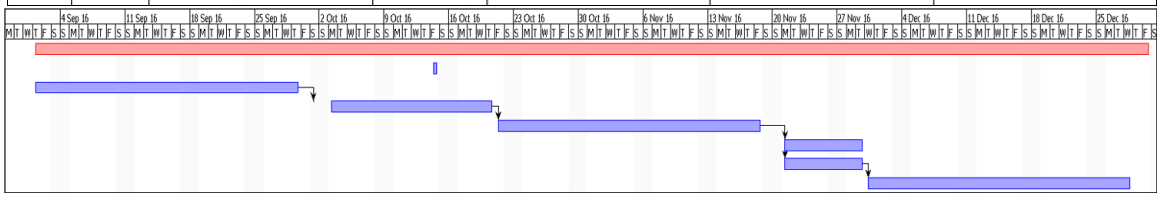
Priority 1: The system shall be secure. Since it is a closed system, the device will be programmed such that it cannot be accessed remotely and will only output sanitized data.

3.4.3 Telemetry

Priority 1: The system will perform telemetry. The data will be transmitted with a delay of up to 10 seconds.

3.5 Gantt Chart

		Name	Duration	Start	Finish	Predecessors
1		Design	87 days	9/1/16 8:00 AM	12/30/16 5:00 PM	
2		CoDR	1 day	10/14/16 8:00 AM	10/14/16 5:00 PM	
3		Design Direction	21 days	9/1/16 8:00 AM	9/29/16 5:00 PM	
4		Concept Design	14 days	10/1/16 8:00 AM	10/20/16 5:00 PM	3
5		Preliminary Design	21 days	10/21/16 8:00 AM	11/18/16 5:00 PM	4
6		Simulations	7 days	11/21/16 8:00 AM	11/29/16 5:00 PM	5
7		Prototype	7 days	11/21/16 8:00 AM	11/29/16 5:00 PM	5
8		Final Design	21 days	11/30/16 8:00 AM	12/28/16 5:00 PM	7



3.6 Changes Since Original Requirements Document

3.7 Final Gantt Chart

4 Design Document

4.1 Original Design Document

4.2 Introduction

4.2.1 Document Overview

4.2.1.1 Helena Bales

1. Target Generation
2. Arm Movement
3. Arm Position Tracking

4.2.1.2 Amber Horvath

1. Emergency Payload Expulsion
2. Program Modes of Operation
3. Target Success Sensors

4.2.1.3 Michael Humphrey

1. Telemetry
2. Video Camera
3. Data Visualization and Processing

4.3 Technologies

4.3.1 Target Generation

4.3.1.1 Requirement Overview

The software shall generate points to be used in testing the Hephaestus arm. The points will constitute the total test of the arm, and should therefore include points representative of standard and edge cases. These points shall be used as targets for the arm body.

4.3.1.2 Solution Design

The points shall be generated in 3-D polar form, including an angle from normal, a radius, and a height. The angle shall be in the range of 0 and 359 degrees. An angle of zero degrees shall be in the direction of payload deployment. The radius shall be the distance from the arm's attachment to the base to the generated point. The height of the point, for the purpose of target generation, shall be constant. However the points will always be stored in a triple of angle from normal (θ), radius (r), and height (h).

The test points that are generated shall represent a sample of points over the range of motion required of the arm. As such the points should be at the extremes of where the arm can reach, in the middle of the arm's range, and close to the arm base. Showing this full range of motion and the accuracy with which the range can be achieved will show the viability of construction on orbit.

The test points shall be generated prior to the launch. The test points will be generated by using a random number generator to pick a number in a range defined by which case the point is designed to test. For example, a point intended to test the arm's ability to reach near the base would generate an angle around the normal, a radius close to zero, and a height of zero. In this way, the generated test point will test a functionality of the arm. The test points will be generated prior to launch in order to insure that the points adequately cover the desired tests.

4.3.2 Arm Movement

4.3.2.1 Requirement Overview

The software shall control the movement of the arm body assembly. The position of the tip of the arm shall be tracked in the coordinate notation described in subsection 2.2 above. The software shall rotate the arm body assembly in a full 360 degrees. The software shall additionally control the movement the height of the arm body assembly. The arm should descend and touch the baseplate of the payload at any rotation.

4.3.2.2 Solution Design

The movement of the arm shall follow a path through a 4-degree of freedom (dof) configuration space. The path of the arm shall be generated using the A* pathfinding algorithm. The configuration space shall be in $\mathbb{R}^4$. Valid points in the configuration space will be represented by a 0, while invalid points will be represented by a 1. A point in the configuration space represents the angles at which the four arm actuators are bent. In this way, the position of the arm can be uniquely represented. An area in the configuration space maps to a single point in real space.

In order to move from one point to the next, a path will be generated using A* from the starting position to the final position. The final position will be converted from Real Space to the C-Space using Inverse Kinematics. Once the path has been generated, the arm will be moved through the path from the initial configuration through the list of configurations given by the path. In moving from one configuration to the next, the motors will be rotated to the new configuration starting at the base of the arm towards the tip of the arm.

The movement of the arm shall be constrained in several ways in order to prevent damage to the hardware. The first constraint on movement is in the height of the arm. The movement shall be limited by the heights of the arm such that it will not collide with the top or base plates. This

means that at no point should the height of the deployed arm exceed the height of the half can. This measure is meant to protect the hardware in case the payload gets stuck in any position and must be retracted. The second limit to the movement is in the rotation of the arm. The arm should never be allowed to perform more than a single full rotation. This safety measure is meant to keep the wiring of the arm from becoming tangled. The final safety measure that limits the movement of the arm is in the speed and torque allowed for the motors. Both of these values must be limited in order to insure the safety of our payload and the rocket. The velocity of the arm must be limited in case of collision to limit damages. The torque is limited to prevent damage to the arm, the payload, the rocket, and the motors. If the arm gets stuck, we will be able to detect it by measuring the torque that the motor must apply in order to move the arm. If the torque increases dangerously, we can stop, unstick the arm, and continue with the operations.

4.3.3 Arm Position Tracking

4.3.3.1 Requirement Overview

The position of the arm shall be tracked using the same coordinate system described in the Target Generation requirement. The position of the arm shall be calculated using the known start position and the rotation of the motors.

4.3.3.2 Solution Design

The position of the arm shall be tracked using the motor movement to calculate p and p_{m2}. The initial position of the arm shall be defined in advance, and a sensor will be placed at that location. From there, the position will be tracked from the movement of the motors. The arm will recalibrate by returning to the initial position in order for the error to not increase over time. The position of the arm shall be denoted p, the location of the tip of the arm. From the coordinate p, the location of p_{m2}, the center of the middle joint of the arm, will be calculated. The height of p_{m2} will be calculated from the triangle made of the two arm subsections, L1 and L2, and the radius of point p. From there the radius of the point p_{m2} can be calculated using the triangle of L1, h_{m2} and the radius of m2. Finally, the σ of p_{m2} shall be the same as that of p. Using this method will allow for the extra condition that point p_{m2} should never exceed the height of the can. Constrain rotation to not go all the way around.

The position of the arm will be verified after the flight by using visual confirmation from the video camera. The purpose of tracking the position of the arm is to verify the accuracy of the arm on orbit. Since this will determine our ability to determine mission success, it is important that we have several methods of verifying out results. This design allows us to know where we want to be by storing the values of p, the position of the tip of the arm, that occur during the motion of the arm. We can also know where we are compared to where we started by storing the motion applied by the motors. We can know where we actually are through the triggering of sensors. Finally, we can verify the sensor data using the video camera.

4.3.4 Emergency Payload Expulsion

Author: Amber Horvath

4.3.4.1 Requirement Overview

The software shall eject the arm upon system failure. System failure in this case is defined as the arm becoming lodged or stuck in a state where it is unable to retract. The software will enter Safety mode (defined in subsection 2.5.2) and attempt to retract the arm. If it is unable to complete this step, the system will continue attempting to eject the arm until ejection is completed.

4.3.4.2 Solution Design

Upon entering the Shutdown state, the system should succeed in closing the arm, the Arm Assembly Body should be retracted, and the On-Board Computer (OBC) should be powered off. The system shall determine shutdown was not completed correctly (as seen in state 7 defined in subsection 2.5.2) by determining that one of these requirements was not met. The system shall determine the arm is not contracting properly by the amount of torque that the motor is applying, as failure to contract will require more torque. The system will fire an interrupt signal from the AVR interrupt library, notifying the system to transition to Safety mode. Safety mode will attempt to contract the arm once more by calling the arm movement function. The arm movement function will take a coordinate to move the end of the arm to. The arm is equipped with sensors that can determine if the arm is folded or not so if the sensors determine that the arm is folded, then safe shutdown should be possible. The emergency retracting operation is completed by turning off all the motors in the arm except for the motor pushing the whole metal plate the arm is attached to in and out of the payload. With those motors turned off, the joints of the arm will be flimsy and can be pulled into the payload by retracting the metal plate. In the case of contracting the arm, the tip should point inwards to the center of the canister. If it is unable to do so, it shall continue attempting to eject. The system shall initiate the arm ejection sequence by turning on the motor in control of ejecting part of the arm and turning off all other motors. The system shall also clean up any memory leaks and ensure all telemetry ports are closed upon sending the data that an emergency ejection was required. In the post-mortem analysis, information regarding the arm's expulsion will be useful. The system shall, upon receiving a signal that ejection is required, send a log description of the current polar coordinates of the arm, the time elapsed since last arm movement request, and what state the system was in prior to being sent to the Safety state. The system will continue attempting to eject the arm until the system detects that the metal plate has successfully returned into the payload. The system shall determine this by a pin being set from low to high upon entry into the payload. If the arm is unable to be ejected safely, the arm will be stuck outside the canister and the mission shall be counted as a failure.

4.3.5 Program Modes of Operation

Author: Amber Horvath

4.3.5.1 Requirement Overview

The software shall have the Modes of Operation necessary to insure the mission success. The software shall first deploy the payload, then the arm. Next the software shall activate the camera and perform a video sweep. The software shall then perform the science experiment. If the experiment fails, it shall return to observation mode. If observation mode fails, it shall return to idle. Once the experiment time has been exhausted, the payload shall shut down. If it shuts down correctly,

everything will poweroff. If not, the payload shall attempt to retract again, or expel the payload from the rocket.

4.3.5.2 Solution Design

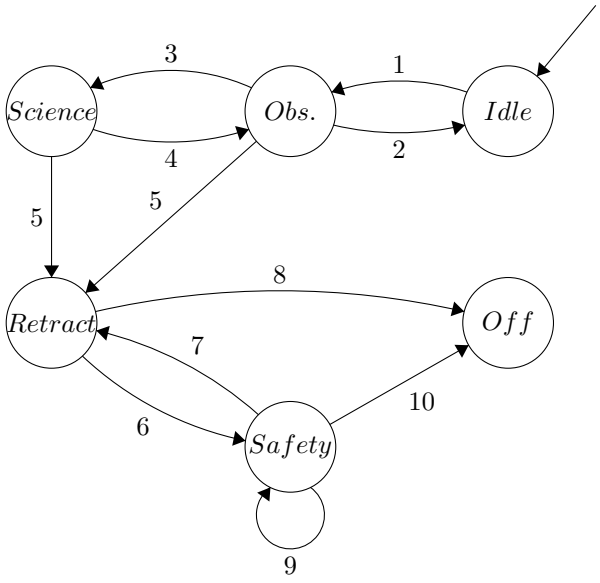


Diagram of software states of operation and transition between states [2].

Transitions between states occur as numbered:

1. **Appogee is reached.** The software shall activate when the power line goes to high at 28V. Observation mode shall be triggered when the OBC turns on. Observation mode will collect a sweep of the payload with the camera. This mode will ensure that the camera is operational during the more crticial parts of the mission.

2. **Error: Return to Idle.** If an error is encountered in entering Observation mode, the software shall fallback to Idle mode and retry. An error may occur if the payload fails to deploy correctly or if the camera fails to turn on. The system shall send a signal using the AVR interrupt library if the arm is not fully extended, as the arm is equipped with sensors to determine whether it is extended or folded. If the camera fails to turn on, the system shall be notified as the telemetry line will be sending empty data.

3. **Payload Assembly and Camera have been deployed.** The software shall enter Science mode once the payload assembly and arm have deployed and the camera has performed an observation sweep. Science mode will consist of the arm touching the sensors in the payload canister, and collecting data via the telemetry line. The whole mode shall be captured with the camera.

4. **Error: Return to Observation** The software shall return to observation mode if any error occurs in Science mode. An error may occur in Science mode if the arm fails to operate correctly and must return to default position. An error may also occur if the camera stops working. The system shall know if the arm fails as a timer can keep track of the time between an arm movement request and the arm actually completing the movement request. If too

much time has elapsed between the request and the movement, the arm may be stuck. If the telemetry line stops receiving data from the camera, then the camera has stopped working and the system shall be notified via an interrupt.

5. **Timer switches to end appogee period.** Once the time period for observation has ended, the timer line will go to low and trigger to Shutdown state. This state can be reached from either Observation or Science mode.

6. **Error: Shutdown not completed successfully.** If an error occurs in the shutdown sequence, the software shall enter Safety mode. An error that could occur is the arm failing to close, the Body failing to retract, or the OBC not powering off. All these situations except for the OBC not powering off are handled through Safety mode.

7. **Retry: Re-attempt to retract the arm.** Attempted to resolve any errors and retry retracting the arm.

8. **Accept: Shutdown correctly** If Shutdown occurs correctly, the arm should be closed, the Arm Assembly Body should be retracted, and the OBC should be powered off. The arm will be have sensors to detect whether its closed or not, which can also be used to know whether it has been retracted into the body. Once the system has determined that this criteria has been met, it will power off.

9. **Error: Payload is still deployed.** The software shall remain in Safety mode until the payload is either retracted correctly, retracted fully with the arm in the open position, or ejected safely from the rocket. Safety mode shall first try to correctly retract the arm, then retract with the arm open, then repeat attempting ejection until the payload is ejected.

10. **Payload is Shutdown correctly.** If the payload is Shutdown through Safety mode, shutdown can be completed. In Safety mode, the payload was either shut down correctly, retracted fully into the can with the arm open, or the arm was expelled safely from the rocket. The shutdown sequence consists of the arm closing, the Body retracting, and the OBC being powered off.

4.3.6 Target Success Sensors

Author: Amber Horvath

4.3.6.1 Requirement Overview

The software shall know whether or not the arm succeeded in touching the targets generated, as described in subsection 2.1. The sensors shall report back whether or not contact was made. This data can be used in post-mortem analysis to determine whether certain targets were faulty or whether the range of motion on the arm was faulty.

4.3.6.2 Solution Design

The payload shall be equipped with pre-placed sensors that the arm shall make contact with. The arm shall have generated targets as described in subsection 2.1. These coordinates shall be stored within the system and used as inputs for the function controlling the arms' movements, with the target position being where the tip of the arm should be located. The arm shall exert force to touch the sensor, and the sensor shall go high if contact is made. The sensors high or low signal shall be

sent via the telemetry line and written to our SD card. If the arm gets stuck during this process, it will enter Safety mode, as described in subsection 2.5. The telemetry data shall later be visualized using Python's UI package, TK.

4.3.7 Telemetry

Author: Michael Humphrey

4.3.7.1 Requirement Overview

The telemetry component shall report via telemetry all error codes and test results.

4.3.7.2 Solution Design

The telemetry component is responsible for collecting and sending data through the telemetry ports on the payload.

This component shall not be responsible for transmitting data generated from the temperature sensors. The temperature sensors will be wired directly to an analog telemetry port, bypassing the OBC altogether.

For each test the software successfully completes (see subsection 4.3.6, Target Success Sensors) this component shall output a code representing the test number to the telemetry port. There shall be two tests; one for each touch point on the payload. The tests shall be designed as a joint effort between the Hephaestus Structures, Robotics, Electrical, and Software teams.

The output shall be encoded as a four character binary string and transmitted simultaneously via a four parallel port pins. The binary string shall be transmitted at a rate of no more than 5,000 Hz. There shall be a delay of no less than .2 milliseconds between transmissions of codes. When no code is being actively transmitted, the telemetry shall output a code of '0000' to the telemetry pins.

In addition to transmitting codes via the telemetry lines, the software shall also store a log file with timestamps on an onboard SD card. The log file shall consist of a series of lines of text, consisting of a timestamp and a description. The timestamp shall be the number of tenths of milliseconds since the OBC powered on. The description shall be a sequence of ASCII characters of arbitrary length, and terminated with a carriage return character.

4.3.8 Video Handling

Author: Michael Humphrey

4.3.8.1 Requirement Overview

Video footage of the payload's operations shall be recorded and saved to the SD card.

4.3.8.2 Solution Design

The Hephaestus Electrical Engineering team shall design the payload such that the camera will power on and off at the appropriate times, as well as save footage to the SD card.

4.3.9 Data Visualization and Processing

Author: Michael Humphrey

4.3.9.1 Requirement Overview

The data visualization and processing component shall provide visualizations for the collected data. This component shall be able to show whether the mission success criteria have been met or not. If the mission success criteria have not been met, this component shall show how and why they have not been met.

4.3.9.2 Solution Design

The component shall have a Graphical User Interface (GUI) written in Tkinter with graphs generated by matplotlib. The GUI shall consist of two graphs, a table, and a timeline. Each graph shall be a plot with analog data collected from each of two temperature sensors. The data from the temperature sensors shall be graphed with respect to time from apogee and actual temperature, if such a value can be determined. In the absence of a method to reliably determine actual temperature from the raw sensor data, then the data shall be graphed with respect to the raw value received from the sensor, with the graph scaled such that the lowest value recorded shall be the minimum y value, and the highest recorded value recorded shall be the maximum y value. The user shall be able to scale the graphs to view portions of the data as they see fit. The table shall consist of the name of each of a series of tests, the result of that respective test, and the time that test was completed. A result shall be either "passed", "failed", or "not completed". A result of "passed" shall be colored in green. A result of "failed" shall be colored in red. A result of "not completed" shall be colored in yellow. If the result of a test is "not completed", then the time of completion for that test may be omitted. There shall be two total tests. The tests shall be for if the payload can successfully touch each touch point sensor. The tests shall be designed as a joint effort between the Hephaestus Structures, Robotics, Electrical, and Software teams. The timeline shall be a visualization with time on the y axis, with significant events marked at various positions along the axis, according to when that event happened.

4.4 Conclusion

This concludes the design of our project. Further questions or concerns can be addressed to the authors of this document. This document may be subject to changes in the future as more design constraints are found, or designs are found to not work the way

4.5 Changes Since Original Design Document

5 Technical Review Document

5.1 Original Technical Review Document

5.2 Introduction

5.2.1 Document Overview

This is the Technical Review And Implementation Plan for the Hephaestus project. This document shall investigate possible methods of implementing our project software requirements. The nine general requirements investigated below were identified as project requirements in our Requirements document. This document will focus on the "how" of our requirements implementation.

5.2.2 Role Breakdown

Each CS Senior Design team member shall be responsible for ensuring the completion of the three items from the requirements document that are assigned to them below.

5.2.2.1 Helena Bales

1. Target Generation
2. Arm Movement
3. Arm Position Tracking

5.2.2.2 Amber Horvath

1. Emergency Payload Expulsion
2. Program Modes of Operation
3. Target Success Sensors

5.2.2.3 Michael Humphrey

1. Telemetry
2. Video Camera
3. Data Visualization and Processing

5.3 Technologies

5.3.1 Target Generation

5.3.1.1 Requirement Overview

The software shall generate points to be used in testing the Hephaestus arm. The points will constitute the total test of the arm, and should therefore include points representative of standard and edge cases. These points shall be used as targets for the arm body.

5.3.1.2 Proposed Solutions

1. **The points shall be generated in 3-D polar form**, including an angle from normal, a radius, and a height. The angle shall be in the range of 0 and 359 degrees. An angle of zero degrees shall be in the direction of payload deployment. The radius shall be the distance from the arm's attachment to the base to the generated point. The height of the point, for the purpose of target generation, shall be constant. However the points will always be stored in a triple of angle from normal (θ), radius (r), and height (h).
2. **The points shall be generated in 3-D Cartesian form**, including x position to the right or left of the $y - axis$, the y position above or below the $x - axis$, and the height, h, above the $xy - plane$. Let the $y - axis$ be the direction that the payload deploys from the can. Let the $x - axis$ be the perpendicular to the $y - axis$ at the point where the arm is mounted to the rotating plate. Let h be the height above the $xy - plane$, where the arm is attached to the rotating plate.
3. **The points shall be generated in 2-D Polar coordinates**, where the implementation is the same as described in the 3-D Polar coordinate section, with the exception of h. In this case, there shall be no h. The position can be represented in 2-D Polar coordinates on the plane of the base plate. For the purpose of compatibility with the position of the arm, the height could be assumed to be 0.

5.3.2 Arm Movement

5.3.2.1 Requirement Overview

The software shall control the movement of the arm body assembly. The position of the tip of the arm shall be tracked in the coordinate notation described in section 2.2 above. The software shall rotate the arm body assembly in a full 360 degrees. The software shall additionally control the movement the height of the arm body assembly. The arm should descend and touch the baseplate of the payload at any rotation.

5.3.2.2 Proposed Solutions

1. **The movement of the arm shall be generated by a custom system where the movement of the arm is generated based on the current position and the starting position.** The position of the tip of the arm shall be stored as decided from the list of solutions above. In the case of the selection of solution 3, the position will have an added height. The

position shall be denoted as point p and shall be the location of the tip of the arm. The arm shall generate a series of commands for the motors to perform to go from p to the target, t_n where t_n is the n-th target.

2. **The movement of the arm shall be generated by a custom system where the movement of the arm is generated based on the current position and the starting position.** The position of the tip of the arm shall be stored as decided from the list of solutions above. In the case of the selection of solution 3, the position will have an added height. The position shall be denoted as point p and shall be the location of the tip of the arm. The arm shall generate a series of commands for the motors to perform to go from p to the target, t_n where t_n is the n-th target. The movement of the arm shall be constrained by the heights of the arm so that it will not collide with the top or base plates.

3. **The movement of the arm shall be accomplished by turning the arm to the correct θ, then correct radius, then correct height.** The rotating base plate will be responsible for turning the arm to the correct θ value. The motors, labeled $m1$, and $m2$, shall be responsible for moving the arm to the correct radius and height. The position of the arm, p, and the target position t_n, shall be stored in the manner described in the section titled Arm Position Tracking.

4. **The movement of the arm shall follow a path through a 4-degree of freedom (dof) configuration space.** The path of the arm shall be generated using the A* pathfinding algorithm. The configuration space shall be in $\mathbb{R}^4$. Valid points in the configuration space will be represented by a 0, while invalid points will be represented by a 1. A point in the configuration space represents the angles at which the four arm actuators are bent. In this way, the position of the arm can be uniquely represented. An area in the configuration space maps to a single point in real space.

   In order to move from one point to the next, a path will be generated using A* from the starting position to the final position. The final position will be converted from Real Space to the C-Space using Inverse Kinematics. Once the path has been generated, the arm will be moved through the path from the initial configuration through the list of configurations given by the path. In moving from one configuration to the next, the motors will be rotated to the new configuration starting at the base of the arm towards the tip of the arm.

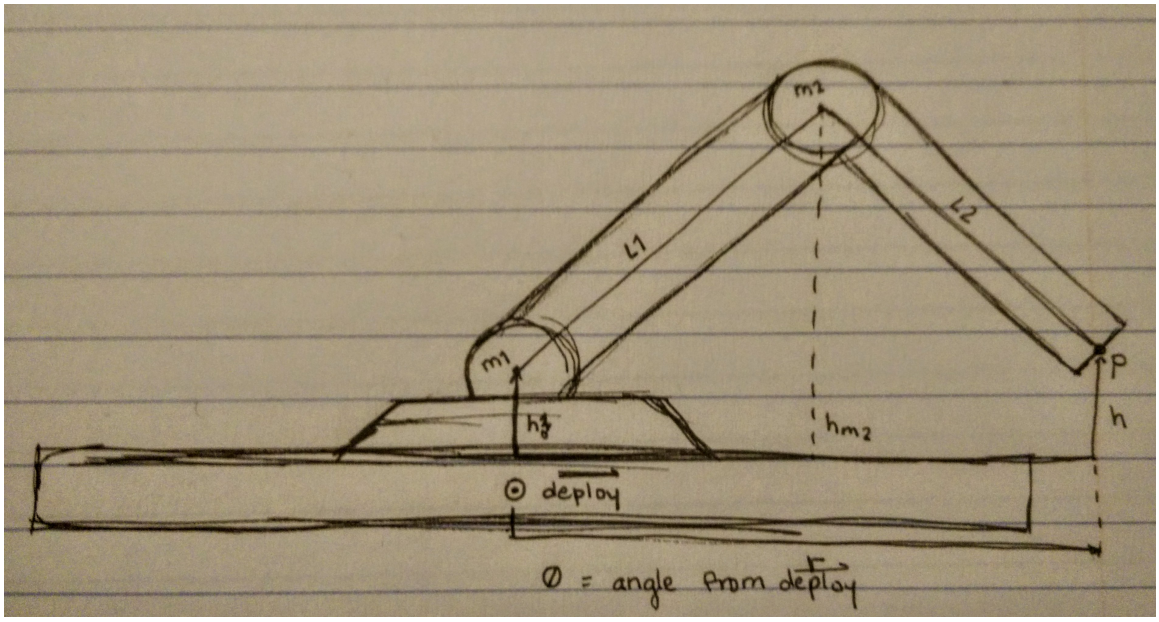


Figure 1 – Arm Movement Design

5.3.3 Arm Position Tracking

5.3.3.1 Requirement Overview

The position of the arm shall be tracked using the same coordinate system described in the Target Generation requirement. The position of the arm shall be calculated using the known start position and the rotation of the motors. The starting position shall be known due to a calibration point that will allow for a reset at any time. Resetting in this way will allow for flexibility between resetting for maximum operation time with only tolerably small error defined by the Non Functional Requirements.

5.3.3.2 Proposed Solutions

1. **The position of the arm shall be tracked using the motor movement.** The initial position of the arm shall be defined in advance, and a sensor will be placed at that location. From there, the position will be tracked from the movement of the motors. The arm will recalibrate by returning to the initial position in order for the error to not increase over time. The position of the arm shall be denoted p, the location of the tip of the arm. This shall be the only position tracked.

2. **The position of the arm shall be tracked using the motor movement to calculate p and p_{m2}.** The initial position of the arm shall be defined in advance, and a sensor will be placed at that location. From there, the position will be tracked from the movement of the motors. The arm will recalibrate by returning to the initial position in order for the error to not increase over time. The position of the arm shall be denoted p, the location of the tip of the arm. From the coordinate p, the location of p_{m2}, the center of the middle joint of the arm, will be calculated. The height of p_{m2} will be calculated from the triangle made of the two arm sections, L1 and L2, and the radius of point p. From there the radius of the point p_{m2} can

be calculated using the triangle of L1, h_{m2} and the radius of m2. Finally, the σ of p_{m2} shall be the same as that of p. Using this method will allow for the extra condition that point p_{m2} should never exceed the height of the can.

3. **The position of the arm shall be tracked using the motor movement to calculate p, with a limit on the height of the arm.** The initial position of the arm shall be defined in advance, and a sensor will be placed at that location. From there, the position will be tracked from the movement of the motors. The arm will recalibrate by returning to the initial position in order for the error to not increase over time. The position of the arm shall be denoted p, the location of the tip of the arm. This will be the only point tracked, however the values of p shall be restricted such that the height of the arm never exceeds the height of our half can.

5.3.4 Emergency Payload Expulsion

5.3.4.1 Requirement Overview

The software shall eject the arm upon system failure. System failure in this case is defined as the arm becoming lodged or stuck in a state where it is unable to retract. The software will enter Safety mode (defined in section 2.5.2) and attempt to retract the arm. If it is unable to complete this step, the system will continue attempting to eject the arm until ejection is completed

5.3.4.2 Proposed Solutions

1. **The software accepts a signal sent from the Shutdown state to the Safety state** Upon entering the Shutdown state, the system should succeed in closing the arm, the Arm Assembly Body should be retracted, and the OBC should be powered off. If any of these conditions are not met, a signal should be sent, resulting in a change of state from the Shutdown state to the Safety state, where the arm can be ejected.

2. **The software sends a signal to enter Safety state upon any failure to complete an arm-movement task** A timer should be implemented to detect whether a certain amount of time has elapsed between the last arm movement and the last request for an arm movement. If arm movement requests are not being met by arm movements, and the system stalls past a certain amount of time, the system should send a signal to enter the Safety state so that the arm can be ejected, as it is most likely caught in a bad extended position.

3. **The software shall notify via telemetry that ejection was required** In the post-mortem analysis, we will want to know whether an ejection was necessary and what caused the bad state. The system shall, upon receiving a signal that ejection is required, send a log description of the current coordinates of the arm, the time elapsed since last arm movement request, and what state the system was in prior to being sent to the Safety state.

5.3.5 Program Modes of Operation

5.3.5.1 Requirement Overview

The software shall have the Modes of Operation necessary to insure the mission success. The software shall first deploy the payload, then the arm. Next the software shall activate the camera and perform a video sweep. The software shall then perform the science experiment. If the experiment

fails, it shall return to observation mode. If observation mode fails, it shall return to idle. Once the experiment time has been exhausted, the payload shall shut down. If it shuts down correctly, everything will poweroff. If not, the payload shall attempt to retract again, or expel the payload from the rocket.

5.3.5.2 Proposed Solutions

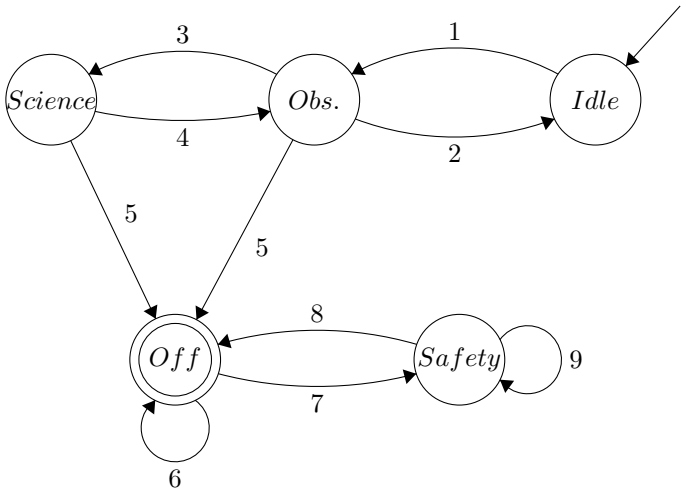


Diagram of software states of operation and transition between states [2].

Transitions between states occur as numbered:

1. **Appogee is reached.** The software shall activate when the power line goes to high at 28V. Observation mode shall be triggered when the OBC turns on.
2. **Error: Return to Idle.** If an error is encountered in entering Observation mode, the software shall fallback to Idle mode and retry. An error may occur if the payload fails to deploy correctly or if the camera fails to turn on.
3. **Payload Assembly and Camera have been deployed.** The software shall enter science mode once the payload assembly and arm have deployed and the camera has performed an observation sweep.
4. **Error: Return to Observation** The software shall return to observation mode if any error occurs in Science mode. An error may occur in Science mode if the arm fails to operate correctly and must return to default position. An error may also occur if the camera stops working.
5. **Timer switches to end appogee period.** Once the time period for observation has ended, the timer line will go to low and trigger to Shutdown state. This state can be reached from either Observation or Science mode.
6. **Accept: Shutdown correctly** If Shutdown occurs correctly, the arm should be closed, the Arm Assembly Body should be retracted, and the OBC should be powered off.
7. **Error: Shutdown not completed successfully.** If an error occurs in the shutdown sequence, the software shall enter Safety mode.

8. **Payload is Shutdown correctly.** If the payload is Shutdown through Safety mode, shutdown can be completed. In Safety mode the payload was either shut down correctly, retracted fully into the can with the arm open, or the arm was expelled safely from the rocket.

9. **Error: Shutdown not completed successfully.** If an error occurs in the shutdown sequence, the software shall enter Safety mode.

10. **Payload is Shutdown correctly.** If the payload is Shutdown through Safety mode, shutdown can be completed. In Safety mode the payload was either shut down correctly, retracted fully into the can with the arm open, or the arm was expelled safely from the rocket.

11. **Error: Payload is still deployed.** The software shall remain in Safety mode until the payload is either retracted correctly, retracted fully with the arm in the open position, or ejected safely from the rocket. Safety mode shall first try to correctly retract the arm, then retract with the arm open, then repeat attempting ejection until the payload is ejected.

5.3.6 Target Success Sensors

5.3.6.1 Requirement Overview

The software shall know whether or not the arm succeeded in touching the targets generated, as described in section 2.1. The sensors shall report back whether or not contact was made. This data can be used in post-mortem analysis to determine whether certain targets were faulty or whether the range of motion on the arm was faulty.

5.3.6.2 Proposed Solutions

1. **The software shall store the coordinates produced during the target generation stage and compare with the targets actually reported after the arm moves** The software shall have generated a coordinate (the form yet to be determined) that is sent to the arm to move to that specificied location. Post-movement, the arm can keep determine its movement using the motor and the sensor described in section 2.3. A check for equality can be performed between these two points to determine whether the points are equivalent or not. If they are equivalent, the movement was successful and resulted in the target being touched. If the equivalency fails, then the arm did not meet its target and should be set back to a pre-determined starting position to prevent further target points from being influenced by the margin of error. Both a successful target touch and a failed target touch should be stored so as to keep track of the ratio between successful and unsuccessful trials.

2. **The software shall evaluate a delta between the point generated and the actual point reported** A delta can be determined between the arm movement and the difference between that position and the calibration point, where the calibration point is a stored value. If the delta is 0, then the point generated was correct. If not, then the arm should be set back to a stored location to prevent the margin of error influencing further target generation and touches. Both a successful target touch and a failed target touch should be stored so as to keep track of the ratio between successful and unsuccessful trials.

3. **The stored video and telemetry data shall work as an oracle when evaluating our success sensors** This is the least reliable of our methods, as relying on the video capture is risky, along with the less rigorous methodology. However, if all else fails, we can comb over the telemetry data and the stored video capture to determine whether or not the arm

succeeded or failed in touching the generated targets by watching the video and comparing it to the telemetry data. We would be looking for instances where the sensor data captured via telemetry matches with video footage of the arm extending and touching a generated point to see whether the data matches up with the video feed.

5.3.7 Telemetry

5.3.7.1 Requirement Overview

The software shall report via telemetry all sensor data.

The criteria that these technologies will be evaluated on is:

- **Ease of use.** The chosen solution should let the developers focus on writing code and not encoding data for telemetry transmission. Ideally, sending data through one of the telemetry ports should be no more than one line of code.
- **Reliability.** The chosen solution should be able to relay 100% of transmitted data to the ground station without corrupting or losing any of it.
- **Documentation.** The chosen solution should be well documented. The developers should be able to quickly and easily locate supporting documentation for using the technology.
- **Compatibility.** The chosen solution should be compatible with the software and hardware of the payload.

5.3.7.2 Proposed Solutions

The three options being considered for transmitting telemetry are

1. **A custom-built solution for our own needs**. The custom-built solution is the least appealing. It would require the most amount of work to develop and maintain by the developers. The advantage of a custom-built solution is that it can be tailored to the requirements of our system, making it extremely to use. However, the benefit is offset by the huge amount of work upfront it would require to develop and test the solution. Since the developers would be coding up this solution themselves, it would require a lot of testing to ensure a reliable solution. The hand-written test cases cannot guarantee the reliability of the solution, especially given the relative inexperience of the developers with writing code for this platform. Therefore one can expect to have relatively unreliable code and encounter lots of bugs. Compatibility would not be a problem with this solution because the code would be custom-made for the hardware. However, documentation would be non-existent because the developers would be writing the code themselves. The only documentation that would be relevant would be from other projects that have written telemetry code for spacecraft. However, most of that documentation would be internal to the organizations building the spacecraft, most likely wouldn't be helpful.
2. **Open MCT developed by NASA for space-specific missions** Open MCT is a mission control framework developed and used by NASA. Because of the many requirements by NASA, Open MCT is a vast and complicated framework. It is incredibly complicated and requires a lot of code in order to do simple tasks. However, because it is supported by NASA, it is highly reliable for space applications. There is lots of documentation on the Open MCT website

for developers. However, it appears that Open MCT does not support telemetry from the spacecraft. It does, however, support data visualization out of the box. (See section 2.9)

3. **PSAS Packet Serializer developed by Portland State Aerospace Society (PSAS).** PSAS Packet Serializer is a student aerospace engineering project developed by PSAS at Portland State University (PSU). The project seeks to create a standard way to encode data for telemetry transmission between various components and the ground station. This solution would be very easy to use because of its simple interface. Only one line is required to both encode and decode data. This solution is also extremely reliable since it has been used in several flights by the PSU team. The solution is also well-documented. There is an entire website dedicated to documenting the simple API. However, the major problem with this solution is compatibility. The solution is implemented in Python, whereas the code for the payload is restricted to C. It is not feasible to run the Python implementation on the microcontroller in C, but it may be possible to port the code to C. This would require a lot of unpleasant work on the developers' part. The goal for this technology is to let the developers quickly and easily relay data to the ground station.

Despite many disadvantages, the best option for now appears to be creating a custom-built telemetry solution due to compatibility issues with the other solutions.

5.3.8 Video Handling

5.3.8.1 Requirement Overview

The software shall be responsible for controlling the camera output.

The criteria that these technologies will be evaluated on is:

- **Reliability.** The solution should guarantee that video footage is permanently recorded.
- **Ease of use.** The solution should be easy to implement and use.

5.3.8.2 Proposed Solutions

The three options being considered for controlling the camera are:

1. **Enabling and disabling a third-party camera.** This solution involves turning on and off a self-contained third-party camera. Defining what the camera will be is outside of the scope of the Hephaestus software team. The camera used will be decided by the Hephaestus electrical and robotics teams based on their design requirements. Currently, a GoPro is the most likely to be used for the camera. Self-contained shall be defined as a product that can start, stop, and store video footage without any outside input. The software shall enable video recording at the beginning of the demonstration, and stop video recording at the end.
2. **Enabling/disabling an on-board camera, and storing video output.** This solution involves turning on and off a video camera, as well as processing and storing the video output. Defining what the camera will be is outside of the scope of the Hephaestus software team. The camera used will be decided by the Hephaestus electrical and robotics teams based on their design requirements. The software shall start video recording at the beginning of the demonstration, and stop video recording at the end. Additionally, the software shall process

the output of the video camera and store it in a location so that it can be recovered after the rocket returns to earth.

3. **Enabling/disabling an on-board camera, and transmitting video output through telemetry ports.** This solution involves turning on and off a video camera, as well as processing and transmitting the video output though the telemetry ports. Defining what the camera will be is outside of the scope of the Hephaestus software team. The camera used will be decided by the Hephaestus electrical and robotics teams based on their design requirements. The software shall start video recording at the beginning of the demonstration, and stop video recording at the end. Additionally, the software shall process the output of the video camera and transmit it through the telemetry ports to the ground station. In the event of the rocket not being recovered, the video feed can still be kept from the telemetry playback.

The recommended solution for this technology is enabling and disabling a third-party camera.

5.3.9 Data Visualization and Processing

5.3.9.1 Requirement Overview

After the mission completes, the software shall provide visualizations for the collected data. The software shall be able to show whether the mission success criteria have been met or not. If the mission success criteria have not been met, the software shall show how and why they have not been met.

The criteria that these technologies will be evaluated on is:

- **Cross-platform compatibility.** The chosen solution should be able to run across any of the major computing platforms.
- **Range and variety of visualization methods.** The chosen solution should have a large variety of different visualization methods.
- **Documentation.** The chosen solution should be well documented. The developers should be able to quickly and easily locate supporting documentation for using the technology.
- **Developer proficiency.** The majority of developers should be able to comfortably develop the visualizations without needing to learn any new technologies.

5.3.9.2 Proposed Solutions

The three options being considered for visualizing the data are:

1. **Matplotlib.** Matplotlib is a Python plotting and graphing library. Matplotlib is written in Python, and will therefore run on all platforms that Python supports. Matplotlib supports both 2d and 3d graphics, and can render dozens of different types of graphs. Since Matplotlib is used and supported by thousands of developers, there is ample documentation for all aspects of the library. All core developers for the Hephaestus mission are familiar with Python.
2. **Vis.js.** Vis.js is a Javascript library for constructing graphs in a browser. Since it is rendered in a browser, it is accessible on all platforms with a web browser that runs Javascript. Vis.js lists 20 different 2d graphs on its website and 13 different 3d graphs, as well as other graphs including

timelines and networks. Vis.js has less documentation for it on its website, and because it's a smaller library there are less third-party resources for learning it online. However, there is enough documentation to start using it on its website. The API is easy enough that there should not be any significant challenges because of the lack of documentation. Only about half of the Hephaestus development team is familiar with Javascript, so that may be an obstacle going forward if this solution is used.

3. **Lighting.** Lighting provides a unique and flexible way to create graphs. Instead of using a library to render graphs, Lighting uses a web server to render the graphs. Developers can request a server to render a graph, and then retrieve it either via a RESTful web API or through one of several client libraries. Developers can either opt to run their own server, or use one of several public servers Lightning has provided for free. Because Lighting doesn't restrict what programming language you can use to create charts and graphs, the developers are free to choose whatever language they are most proficient in. Lighting also provides the ultimate level of cross-compatibility among platforms because it is completely platform agnostic. Since it runs in a server by itself, it can be accessed by any platform with a TCP/IP stack. Lighting lists 15 different graphs it can render by default; with the potential to add many more. Lighting can be extended to support more kinds of graphs though npm modules. Lightning provides a variety of documentation sources on its website. There isn't an overwhelming abundance of documentation, but it appears to be enough to successfully start developing charts and graphs using it.

The recommended solution for this technology is Lightning.

5.4 Conclusion

The Hephaestus RockSat-X Payload will continue with the implementation of one of the listed possible solutions to each of the nine requirements outlined in this document. The development of the software will begin through the end of Fall term of 2016 and continue during the Winter 2017 term. Once we have obtained the hardware for the arm, we shall begin development of the arm control software, the video recording, and the payload behavior for the duration of the flight. This development will be followed by thourough testing, which will be described in future documents.

5.5 Changes Since Original Technical Review Document

6 Weekly Blog Posts

NOTE:Follow the format and put your posts for fall week 4 in Fall 2016/Week 4/Your-Name for example. Change the weeks to have the right ranges. Delete this note.

6.1 Fall 2016

6.1.1 Week 3

6.1.1.1 Helena Bales

Progress
This week I made significant strides in the design of our project. I wrote part of the Project Definition assignment. I started the Project Description with a description of the problem, broken down into the requirements of the RockSat-X program and the payload that we decided on for the project. In order for our senior design project to be successful, we have to build the payload, meet the RockSat-X project requirements (such as testing, documentation, and design reviews), and meet the capstone class requirements. Our payload idea is a mechanical arm, and as a project it is capable of meeting all the requirements.

While the Project Definition document met our capstone class requirements for the week, there were also RockSat-X requirements to be met this week. The RockSat-X CoDR (Conceptual Design Review) was this week. As a large group (including two teams of ME's, one team of EE's, and the CS team) we developed the CoDR powerpoint that was presented yesterday to RockSat-X. This document included all of our conceptual payload designs thus far, and was our first time presenting our designs to the RockSat-X group. Following that presentation, in order to meet the RockSat-X requirements, we took a group photo.

In addition to the RockSat-X requirements and the capstone class requirements, we met the payload requirements by meeting with Nancy Squires to discuss the project, get approval of the Project Definition assignment, and discuss starting an official Space Lab at OSU.

Plans
The next week will be focusing on the development of documents for Senior Design class as well as for the RockSat-X project. Specifically, we will be revising the Project Description document and begin the Requirements Document. We will also be continuing the design process for the payload with the other teams.

Problems
The problems that we have encountered have been minimal so far.

6.1.1.2 Amber Horvath

6.1.1.3 Michael Humphrey

6.1.2 Week 4

6.1.2.1 Helena Bales

Progress
This week I was at the Grace Hopper Celebration of Women in Computing. I did not do any work directly on the RockSat-X project, but I did talk to many people about Computer Science and space exploration.

Plans
Next week will be focusing on the development of the requirements document for Senior Design and

PDR presentation. The PDR presentation is coming up and will require us to compile a powerpoint about our design, practice presenting it, and presenting it for the RockSat-X program.

Problems
I encountered a significant obstacle to completing work this week. I did not have internet access at Grace Hopper, so I was unable to work on the project or create an update.

6.1.2.2 Amber Horvath

6.1.2.3 Michael Humphrey

6.1.3 Week 5

6.1.3.1 Helena Bales

Progress
This week was focused on developing the requirements document for Hephaestus and revising the Project Description document. The revision of the Project Description document was turned in on Wednesday after adding more of a focus on the software side of the project. The first draft of the requirements document will be turned in by the end of the day today. I focused on creating the outline of the document and writing the introduction. The introduction establishes a purpose and description of the document, an overview of the mission description, the mission success criteria, and the priorities for the requirements. The rest of the document describes the functional and non functional requirements that we have established for the software that controls the Hephaestus payload.

Plans
The next week will focus on creating a solid final draft of the Requirements Document and presenting PDR. That will require meeting as a group to practice presenting PDR and meeting as a group to present PDR.

Problems
Availability has been a problem this week. It has been a challenge to fit all of the large group meetings into my schedule and still have time to catch up on homework after Grace Hopper and work.

6.1.3.2 Amber Horvath

6.1.3.3 Michael Humphrey

6.1.4 Week 6

6.1.4.1 Helena Bales

Progress
This week, work focused on the development of the Requirements Document for Senior Design and finalizing the PDR presentation for RockSat-X. I mainly focused on the Requirements Document, and did significant work on the structure and content of that document. We turned in a draft first, then flushed it out to a final document that was turned in on Friday of Week 6. I focused on the

functional requirements, introduction, and structure of the paper. For the PDR presentation, we had to develop requirements and a plan to meet the requirements. There was a lot of overlap in content between PDR and the Requirements Documents, which was ideal for finishing both of these big documents in the same week. In preparation for this presentation, we had one meeting where we all went over content and one where we practiced the presentation. The final presentation for PDR (Preliminary Design Review) was at 7am on Thursday of week 6. Finally, I revised the README for this repository, so that it was more informative regarding the structure, contents, and context of this repository.

Plans
Next week will focus on finalizing major design choices and developing the technical review. The design choices that need to be finalized include the method for assigning test points and the operational modes of the arm.

Problems
None.

6.1.4.2 Amber Horvath

6.1.4.3 Michael Humphrey

6.1.5 Week 7

6.1.5.1 Helena Bales

Progress
This week we are developing the Technical Review Document for Senior Design. As such, we have divided the requirements up between the three of us as follows:

Amber Horvath:

Emergency Expelling of Payload

Program Modes of Operation

Target Success Sensors

Helena Bales:

Target Generation

Movement of arm

Arm position tracking

Michael Humphrey:

Telemetry

Video Camera

Data visualization and processing

Each of us shall be responsible for insuring the completion of their assigned tasks. We will focus on our assigned tasks for the tech review. This week has focused on defining and assigning the requirements to each of us. We have also finalized some design choices, specifically in the modes of

operations, emergency procedures, and arm target generation.

Plans
Next week we will complete and turn in the tech review on Monday of Week 8. Before that date we will be finishing that document. After the completion of the tech review we will be going back through past documents and including all suggestions we have received as feedback throughout the course. We will be doing this to prepare for the final document to be turned in on December 4th. We will also be preparing our designs and requirements for our big RockSat-X review during weeks 10 or 11.

Problems
We mainly are encountering the issue that we have too many assignments due on or before Monday of Week 8.

6.1.5.2 Amber Horvath

6.1.5.3 Michael Humphrey

6.1.6 Week 8

6.1.6.1 Helena Bales

Progress
This week we finalized and turned in the technical review document. Preparing this document required meeting as a group to talk about potential solutions, then documenting the solutions that we came up with. This week we also talked to the Electrical Engineering group to make sure that our plans were consistent and that we would be able to work together on the software/hardware interface in the future.

Plans
Next week we plan on starting the Design Document and the presentation for the end of the term and our CDR presentation with RockSat-X.

Problems
I have been experiencing technical issues with my computers, so that is something that I will need to resolve before I can seriously start working on the Design Document.

6.1.6.2 Amber Horvath

6.1.6.3 Michael Humphrey

6.1.7 Week 9

6.1.7.1 Helena Bales

Progress
This week we started the Design Document and slides for the presentation for this class and for our RockSat-X program CDR. We met in order to discuss the solutions that we wanted to choose for each of the requirements. During that meeting I updated our Design Document to reflect the choices that we made, and created issues to reflect the tasks that we have yet to complete.

Plans
In the next week we will be finishing the Design Document, finishing our slides for the class presentation, finishing our slides for the CDR presentation, practicing the CDR presentation, and starting to compile the progress update assignment.

Problems
I am still experiencing technical issues with my computer, but less seriously than before, so progress has been made there.

6.1.7.2 Amber Horvath

6.1.7.3 Michael Humphrey

6.1.8 Week 10

6.1.8.1 Helena Bales

Progress
This week we finished up the Design Document and started the Progress Update write up and presentation. We also prepared for CDR by adding slides to the presentation. In order to finish the design document we talked about how to solve each of the issues from the Requirements document. Once we picked a solution to pursue, we each added detail to our solutions. The CDR presentation was adapted to form the start of our Progress Update presentation since it already describes the project and our work thus far.

Plans
Next week we will be finishing our progress update write up and presentation. We will do the write up first, then make sure that the presentation slides cover the content from the write up, and finally record the presentation.

Problems
None.

6.1.8.2 Amber Horvath

6.1.8.3 Michael Humphrey

6.2 Winter 2017

6.2.1 Week 1

6.2.1.1 Helena Bales

Progress
I made no progress over winter break, other than actually managing to take a vacation. I am very proud of myself.

Since the start of week 1, I met with the Software Team to discuss the schedule that we will have for the term. We decided that we would have an hour long meeting on Mondays after our TA meeting

and an hour long meeting with the whole Hephaestus team on Thursdays at 6pm. We have not yet had a Monday meeting because the past two mondays have been cancelled due to weather and MLK Day.

We accomplished some tasks for week 1, which includes adding more content to the System Architecture for the Software Subsystem. In addition to this planning I began development of the test cases that will be used to test the Software Subsystem. Specifically, I am focusing on developing experiments to test the three functional requirements that I was assigned last term.

Plans
The plan for next week is to finish up the architecture diagram and the test cases and beginning the implementation phase of the project. We need to have a prototype completed and tested by mid February, so we are planning on implementing for three weeks then testing.

Problems
My biggest problem is that I do not have a working computer right now. I have been trying to fix my computer but it has just been a time sink so far. I don't have a solution to this problem.

6.2.1.2 Amber Horvath

6.2.1.3 Michael Humphrey

6.2.2 Week 3

6.2.2.1 Helena Bales

Progress
This week we started really diving into the motion planning in the Pathing and Automation cross-functional team. I checked out books from the library to help with research into motion planning for robotics, robotics in space, and inverse kinematics.

Plans
Over the next weeks I will be doing research into the issues of path planning and motion tracking on earth and in space.

Problems
I still don't have a working computer with which to do the research.

6.2.2.2 Amber Horvath

6.2.2.3 Michael Humphrey

6.2.3 Week 4

6.2.3.1 Helena Bales

Progress
This week was continuing research into the pathing and automation portion of the software. I have found that the A* algorithm for pathfinding within a configuration space will be a good solution. Additionally, we will be breaking up the arm into its individual links in order to move the arm to a

valid configuration. Essentially, we will start at the base of the arm and move that first, then move up the arm to the next link and move that.

Plans
The next week will be finishing up the research phase for pathing and automation and starting implementations. We will be starting with building the Configuration Space.

Problems
We still haven't figured out a good way to build a C-Space.

6.2.3.2 Amber Horvath

6.2.3.3 Michael Humphrey

6.2.4 Week 5

6.2.4.1 Helena Bales

Progress
This week was focusing on figuring out how to build the configuration space (C-Space) in which to perform the path-finding algorithm for the arm's motion. We found that the C-Space need to be in R^4 because we have 4 degrees of freedom in the arm. We also know that for each possible configuration of the arms' motors, we need to know if the configuration is valid in order to map the C-Space. This week we are also starting on the slide for STR.

Plans
The next week will focus on finalizing the slides for STR. Our STR presentation will be at 6am on Friday of week 6. In addition to STR, the Pathing and Automation and Software groups will be meeting with Dr. Smart during week 6 to discuss methods for building the configuration space.

Problems
I am blocked from progressing further with the code for motion planning because we do not yet know enough about the C-Space and how to build it. This should be resolved next week after meeting with Dr. Smart.

6.2.4.2 Amber Horvath

6.2.4.3 Michael Humphrey

6.2.5 Week 6

6.2.5.1 Helena Bales

Progress
This week involved finishing the presentation for STR, a all-team social event, presenting STR, and meeting with Dr. Smart. The meeting with Dr. Smart was on Monday and provided a lot of useful information for pathfinding and automation. We discussed methods for creating the Configuration Space. Dr. Smart explained the ways in which we should limit the payload to keep the configuration space as a plane in R4. We also discussed the best way to generate the configuration space. The options that we discussed were calculating it mathematically using Inverse Kinematics, running a

simulation in solid works, or physically moving the arm to valid configurations and mapping those. Each of these methods has benefits and drawbacks. STR occured on friday. In preparation we created the slides throughout the week. On thursday, at our all-team meeting, we went through all the slides in preparation for the presentation on Friday at 6am. The presentation went very well on Friday. The project reviewer said that she was excited to see our project and that our presentation and progress were both very good. The all-team social was at the All-Team meeting on Thursday after we finished all relevant business. We ordered pizza and played board games. The Software team was divided between the two teams with Michael and I against Amber. Amber's team won the first two rounds, but Michael and I brought in a win in the last round. All in all, it was an effective evening of work and team bonding.

Plans
The next week will focus on creating and recording our presentation for the Senior Design class. We will be working on the presentation on Tuesday, finishing it on Wednesday in order to record the video on Wednesday or Thursday. We will finish the project with editing and posting the video on Thursday and Friday to have it done by Friday. I will also be updating the design documents from last term to reflect the changes we have made. I do not expect there to be significant changes, however there may be some slight modifications to the pathfinding and automation section to reflect what Dr. Smart taught us this week.

Problems
The motors have arrived, so I am no longer blocked on progressing in the code. Following the completion of the presentation for CS462, I will be able to dive into the pathfinding code.

6.2.5.2 Amber Horvath

6.2.5.3 Michael Humphrey

6.2.6 Week 7

6.2.6.1 Helena Bales

Progress
This week was focused on preparing the Midterm project update for the Senior Design course. I started the slides on Monday using our STR presentation as a base and added more details on the software side of the project and deleted many of the slides from STR that were focused on the hardware side. On Thursday we met as a group to finalize the slides and record the audio. Michael then edited the audio while I worked on updating our written documents for last term, compiling my blog posts, and starting the retrospective. Also on Thursday was the all-team meeting. We discussed funding issues because AIAA will not be contributing as much money to the project as they initially said they would, so we now have about $30,000 to fund raise. We will all be reaching out to previous employers, etc. to raise the remainder of the funds.

Plans
The next week will focus on building the configuration space. We are still trying to get motors to use for this step since they need to be back-driven and have encoders. We will be buying foam to line the payload. We will then line the payload with foam to provide a buffer of forbidden space in the C-Space before any collision occurs between the arm and payload. Next we will attach the motors to the arm and an arduino. We will use the arduino to read values from the motors as we back-drive them through every valid configuration. Finally, we will use this data to build the c-space. This will take up most of the week. The only other plan is that the Tuesday Senior Design class is required

attendance.

Problems
We have not yet found motors to use for building the C-Space.

6.2.6.2 Amber Horvath

6.2.6.3 Michael Humphrey

6.2.7 Week 8

6.2.7.1 Helena Bales

Progress
This week was focused on preparing the payload to build the configuration space. On Tuesday I went to the required Senior Design class. I wrote and practiced pitching and talking about the Hephaestus project. After class I talked to McGrath about getting motors with encoders to build the Configuration Space. He gave us three motors that we can back-drive and hook up to an arduino. Later in the day on Tuesday I worked with the Pathfinding and Automation team to prepare the payload to build the C-Space. I bought foam board and cut out pieces of it to line the payload. With this complete, we need to attach the motors to the arm and the arduino then read the values from the motor.

Plans
The next week will mostly be working on the C-Space more and the arm control software. I will be starting by writing arduino code that we can use to build the C-Space. I will be working on that this weekend. Once that is complete, we will build the C-Space. After we have the C-Space, I will start the arm control software that will plot the path through the C-Space and move the arm along the path.

Problems
None.

6.2.7.2 Amber Horvath

6.2.7.3 Michael Humphrey

6.2.8 Week 9

6.2.8.1 Helena Bales

Progress
My primary goal this week was to get an initial mapping of the C-Space. I met this goal. On Tuesday I met with Pathing and Automation to start work on the Arduino code to map the C-Space. They had a start to it with three motors on interrupts and printing their values to Serial out. Unfortunately, the Arduino UNO only has two interrupt pins, so I had to switch one of the motors to poll instead of interrupt. This took a few tries, but by Thursday, which is when we met next, I had a motor polling instead of interrupting. I had also fixed the print statements to be more useful to us. However the code still had some bugs. We ran out of time on Thursday and had to

go to the All-Team meeting, so we met up the next day as well. On Friday I finished debugging the Arduino code and we were finally ready to map the C-Space. I ran the arm all around the payload so we now have an initial data set on the shape of the C-Space to play with.

Plans
My next step in the software is to write a parser for the C-Space data. It is currently in the form of one motor angle per line, with the triples separated by a line with a semicolon on it. So I will go from:

$$theta1theta2theta3;$$

to:

$$point_n =< 0, theta1, theta2, theta3 >$$

I will have a 4D array of 1's, then fill in a 0 in every location indicated by point_n above. The 4D C-Space will then contain a 0 for every valid configuration of motors and a 1 for every invalid configuration.

Once I have an initial C-Space, I will need to do some repair on it to smooth out the C-Space. This will account for angles that are valid but were not sampled.

My other goal for Week 10 is to help the ME's test the payload. They are all finishing up their capstone class, so they need to have a bunch of tests done to show that their parts of the project work. They will need some help from CS and EE in order to test how their hardware parts all work in motion. I will be helping them to write code to move the arm and to devise test cases.

Problems
I currently have way too many projects going on. All of my classes have final projects, so I am worried that I will not have time to get it all done.

6.2.8.2 Amber Horvath

6.2.8.3 Michael Humphrey

6.2.9 Week 10

6.2.9.1 Helena Bales

Progress
This week has been an extremely stressful one. All of my classes have final projects. I have not had time to write the parser for the C-Space data. I did update the presentation for the Final Update video. I have some more visuals to add, but I have recorded most of my audio.

We are also working on the presentation today. We did not realize that it was due tonight, so we will be getting most of it done at 4pm today. Needless to say, we are are stressed by this looming deadline.

Plans
Configuration Space:
My next step in the software is still to write a parser for the C-Space data. It is currently in the

form of one motor angle per line, with the triples separated by a line with a semicolon on it. So I will go from:

$$theta1theta2theta3;$$

to:

$$point_n =< 0, theta1, theta2, theta3 >$$

I will have a 4D array of 1's, then fill in a 0 in every location indicated by point_n above. The 4D C-Space will then contain a 0 for every valid configuration of motors and a 1 for every invalid configuration.

Once I have an initial C-Space, I will need to do some repair on it to smooth out the C-Space. This will account for angles that are valid but were not sampled.

Fundraising:
My other goal for Finals Week is to contact people to ask for funding for our project. We also have ISTR on Friday of Finals week, so we will be spending some time during Finals week working on the presentation for that. We will also be finishing up our presentation and video for this class.

Expo Poster:
Finally, we will be completing the Expo poster next week as we got an extension from Kirsten. I have defined the general sections on the poster in the poster template. This will cover the sections of the poster, the images on the poster, and the assignments for text and graphics.

Poster Sections:

1. Panel 1 - Achieving Detailed Autonomous Movement in Space
2. Panel 1 - a - Building the Configuration Space
3. Panel 1 - a - i - Creating the Configuration Space from Real Space Data
4. Panel 1 - b - Pathfinding in an IR 4 Configuration Space
5. Panel 1 - b - i - Dijkstra's Algorithm
6. Panel 1 - c - Accuracy and Obstacle Avoidance
7. Panel 1 - c - i - Accuracy
8. Panel 1 - c - ii - Obstacle Avoidance
9. Panel 2 - A Rocket-Mounted Autonomous Robotic Arm for Construction in Space
10. Panel 2 - a - Hephaestus Mission
11. Panel 2 - b - Mission Success Criteria
12. Panel 2 - b - i - Minimum Success Criteria
13. Panel 2 - b - ii - Maximum Success Criteria
14. Panel 2 - c - i - Overview

15. Panel 2 - c - ii - Telemetry
16. Panel 2 - c - iii - Data Storage
17. Panel 3 - Programmatics
18. Panel 3 - a - Launch Details
19. Panel 3 - b - Sponsors

Section Assignments:

1. 1 - Helena
2. 2 - a - Amber
3. 2 - b - Amber
4. 2 - c - i - Michael and Amber
5. 2 - c - ii - Michael
6. 2 - c - iii - Amber
7. 3 - a - Michael
8. 3 - b - Michael

Images:

1. 1 - Landscape - Pathfinding through IR4 Graphic
2. 2 - Landscape - Hephaestus Payload
3. 3 - Portrait - ???
4. 4 - Landscape - Team Photo

Image Assignments:

1. 1 - Helena
2. 2 - Amber
3. 3 - Amber/Michael
4. 4 - Michael

Problems
I currently have way too many projects going on. All of my classes have final projects, so I am worried that I will not have time to get it all done. Additionally, I feel the stress of the end of the term is affecting the effectiveness of our team. Spring break will help with that.

6.2.9.2 Amber Horvath

6.2.9.3 Michael Humphrey

6.3 Spring 2017

6.3.1 Week 1

6.3.1.1 Helena Bales

Progress
This week was a light one for Senior Design. I attended the first class on Wednesday and met with my group and the TA on Tuesday. Both meetings gave me an overview of the deadlines and expectations for this term. I met with the whole RockSat-X group on Wednesday for a meeting, pizza, and games. The other team members kept calling me Amber. We did not have a pathing and automation cross-functional group meeting this week since we didn't figure out scheduling soon enough. I did some work on the Parser for the C-Space.

Plans
I plan to finish the C-Space parser next week and move on to pathfinding. I will also be making some funding requests so that we can hopefully travel to the integration testing and launch.

Problems
The main problem for this week is a funding shortage that may mean that we will not be able to send everyone that we need to the integration testing and launch at Wallops.

6.3.1.2 Amber Horvath

6.3.1.3 Michael Humphrey

6.3.2 Week 2

6.3.2.1 Helena Bales

Progress
This week I finished the parser for the C-Space. It reads the configuration space data from a file. The data is in the form of four angles separated by a semicolon from the next set of angles. The parser reads these angles and uses floor to convert them to integers then marks that location in a 37x37x37x37 array as a 0, indicating that it is a valid configuration of motors, or not blocked. There were no required attendance Senior Design classes this week. I had a TA meeting on Tuesday and an all-team meeting on Wednesday. I also had a pathing and automation meeting on thursday, but due to miscommunications, only one other person showed up. It was disappointing that no one else came but it gave me the chance to get Subret caught up on what we were talking about since he replaced Ian (who graduated) as the ME rep for Pathing. In explaining the C-Space to him I realized that I missed a bug when writing my parser in that I forgot to convert negative degrees to their positive equivalents before putting them in the array.

Plans
Next week will involve finishing up the second draft of the RockSat-X poster and continuing work on the pathing and automation tasks. I developed a plan for the pathing and automation tasks during

our meeting this week. Next week I will be fixing that bug in the parser, delegating a C-Space visualization in Matlab to James, and starting the Pathfinding portion of the code.

Problems
The problems that I am currently facing are in receiving enough support from the group at large for my pathfinding and automation tasks. I think this will improve next week as our schedules all become more normal.

6.3.2.2 Amber Horvath

6.3.2.3 Michael Humphrey

6.3.3 Week 3

6.3.3.1 Helena Bales

Progress
This week I delegated making a visualization of the configuration space to James. I had trouble explaining what I needed and what to do because we were not able to meet in person. That resulted in some frustration, but I think he will be able to get the visualization done. I also spent a lot of time on the poster this week. There were a lot of tiny edits to be made in the formatting and content. I also took another group picture of all of the Software Team and some individual photos of everyone for them to use on other things.

Plans
Next week will be continuing work on the pathing and automation tasks. I will also continue working with James on the visualization of the C-Space.

Problems I am currently having problems communicating how to make a visualization of a 4D array with James over text.

6.3.3.2 Amber Horvath

6.3.3.3 Michael Humphrey

6.3.4 Week 4

6.3.4.1 Helena Bales

Progress
This week I was able to get the final visualization from James. I was able to answer the last few questions he had, and he produced a good quality visualization for us to include on our poster. I also took updated group pictures for us as well as some individual pictures for everyone to use on their linkedin profiles. I once again continued working on Pathing and Automation tasks. Finally, I continued to make some changes to the Final Expo Poster and got it approved by McGrath and Dr. Squires.

Plans
On Monday I will submit the final expo poster for printing and turn in the photo release form.

Next week I will interview Evan on Thursday and write the WIRED-style review of his project. As always, I will be continuing my work on the Pathing and Automation code.

Problems
I am having trouble getting support from the Electrical Engineers. They are also very busy with work and classes.

6.3.4.2 Amber Horvath

6.3.4.3 Michael Humphrey

6.3.5 Week 5

6.3.5.1 Helena Bales

Progress
This week I submitted our final poster for printing and turned in my photo release form to the Engineering building. I interviewed Evan on Friday and wrote my WIRED-style review. I continued working on my pathing and automation code, and finished the code that makes the motors step through a path.

Plans
Next week I will be finishing the Pathing and Automation code. I will continue to try to work with the Electrical Engineers and continue to try to secure funding for our travels. I will also start working on my slides for the Progress update.

Problems
I am still having trouble getting the support I need from the other engineers. Hopefully things improve next week.

6.3.5.2 Amber Horvath

6.3.5.3 Michael Humphrey

6.3.6 Week 6

6.3.6.1 Helena Bales

Progress
This week I attempted to finish the pathing and automation code. I ran into some difficulties with this that will require me to finish the code next week. I was introduced to an OSU Foundation grant writer so this week I emailed with him about potential travel grants for us to apply to. He found one potential one, but it requires that someone on the team be under 21, and none of us are. He will still be able to set up a crowdfunding campaign through OSU for us. Other than that, I started working on my slides for the Midterm Progress Update due next week.

Plans
Next week I will be finishing up the code for the Pathing and Automation, as well as my slides for the progress update, the video of the progress update, the FSMR presentation for RockSat-X, setting up the crowdfunding campaign through OSU, creating print visuals in addition to our poster

for Engineering Expo, submitting security clearance and background check authorization forms to Wallops, doing the Poster Extra Credit thing on Monday at 4 with Kirsten, and Expo on Friday.

Problems
Not enough hours in the day. Also my computer has some issues with AVR so I have to find another computer to use. I have also had a hard time getting in contact with the Electrical Engineer who can help me with the Program Memory part of my code, which is something that I am not particularly skilled or practiced at, so I could really use the help.

6.3.6.2 Amber Horvath

6.3.6.3 Michael Humphrey

6.3.7 Week 7

6.3.7.1 Helena Bales

Progress
Plans
Problems

6.3.7.2 Amber Horvath

6.3.7.3 Michael Humphrey

6.3.8 Week 8

6.3.8.1 Helena Bales

If you were to redo the project from Fall term, what would you tell yourself?
If I were to redo the project from Fall term I would have us divide up the work differently. I would also want to be better about creating timelines for the whole team (including the ME's and EE's) so that we could understand our dependencies better from the beginning. We were making a lot of it up as we went along since OSU hasn't ever made a rocketry payload for space. I also would have requested that we remove one degree of freedom from the arm since it would have made things a lot easier, but the challenge has been fun.

What's the biggest skill you've learned?
I have learned a lot about configuration spaces, pathfinding, and have been able to wrap my head around 4D arrays. I have simultaneously been learning about Machine Learning (I wonder how an AI would feel about us learning to make them learn), so I have been thinking about how to incorporate some ML into this project in the future, especially in reacting to changing environments.

What skills do you see yourself using in the future?
I think that I will use my space robotics skills in the future. It's a pretty niche skill, but I think Intelligence and Space Research might be the right place to use that. I will, if nothing else, continue to use my embedded C skills to do hardware projects.

What did you like about the project, and what did you not?

I liked getting to think about pathfinding and 4D arrays, but I did not like dealing with data storage or telemetry, so I was pretty happy when Michael and Amber took the lead on those two things. I did not enjoy working with so many other engineers at first as it could be pretty frustrating when we didn't all know each other very well, but now that is one of my favorite things about it, because we are a team.

If your project were to be continued next year, what do you think needs to be working on?
I hope that this project will continue next year. There are a lot of improvements to be made. I would like to see the following improvements (in order):

1. Dynamic pathfinding on orbit - requires more processing power or longer experiment duration or both
2. Dynamic targeting on orbit - requires 1.
3. Dynamic CSpace Mapping on orbit - requires data from multiple angles and Inverse Kinematics. requires 1.
4. A second arm - requires 1, 2, 3, and a longer mission, probably on a satellite
5. Tool use with one arm
6. Tool use with two arms interacting with each other
7. Image processing and object recognition
8. ...

I could detail my whole idea for construction in space, but I think that is enough for one night.

6.3.8.2 Amber Horvath

6.3.8.3 Michael Humphrey

7 Final Poster

8 Project Documentation

8.1 Project Functionality

8.1.1 Project Structure

8.1.2 Theory of Operation

8.1.3 Block Diagram

8.1.4 Flow Diagram

8.2 Hardware Requirements

8.3 Installation Instructions

8.4 Running Instructions

8.5 User Guides and Documentation

9 Learning New Technology

9.1 Helpful Resources

9.1.1 Web Sites

1.

9.1.2 Books and Print Materials

1.

9.1.3 Faculty and Personel

1.

10 What We Learned

10.1 Helena Bales

10.1.1 Technical Information

10.1.2 Non-Technical Information

10.1.3 Project Work Information

10.1.4 Project Management Information

10.1.5 Team Work Information

10.1.6 If you could do it all over what would you do differently?

10.2 Amber Horvath

10.2.1 Technical Information

10.2.2 Non-Technical Information

10.2.3 Project Work Information

10.2.4 Project Management Information

10.2.5 Team Work Information

10.2.6 If you could do it all over what would you do differently?

10.3 Michael Humphrey

10.3.1 Technical Information

10.3.2 Non-Technical Information

10.3.3 Project Work Information

10.3.4 Project Management Information

10.3.5 Team Work Information

10.3.6 If you could do it all over what would you do differently?

11 Appendix 1: Essential Code

11.1 Pre-Processing

11.1.1 CSpace_Mapping.ino

```
#include <stdio.h>

//=========DEFINITIONS=====================
int encA_pin1 = 6;
int encB_pin1 = 7;
long encA_ticks1 = 0;
//for polling implementation
int n = LOW;
int encA_pin1_prev = LOW;

unsigned long time;
char buffer [50];
int i = 0;

int encA_pin2 = 2;
int encB_pin2 = 4;
volatile long encA_ticks2 = 0;

int encA_pin3 = 3;
int encB_pin3 = 5;
volatile long encA_ticks3 = 0;

const float pi = 3.14;

volatile int state = HIGH;

volatile float encA_degree1 = 0;
volatile float encA_degree2 = 0;
volatile float encA_degree3 = 0;

/*
2 - Motor 1: A
3 - Motor 1: B
4 - Motor 2: A
5 - Motor 2: B
6 - Motor 3: A
7 - Motor 3: B


*/
//=========SET-UP=====================
void setup() {
  // put your setup code here, to run once:
pinMode(encA_pin1, INPUT);
pinMode(encA_pin2, INPUT);
pinMode(encA_pin3, INPUT);

digitalWrite(encA_pin1, HIGH);
digitalWrite(encA_pin2, HIGH);
digitalWrite(encA_pin3, HIGH);

pinMode(13, OUTPUT);

// INIT interrups

//switch motor 1 to polling
//attachInterrupt(digitalPinToInterrupt(encA_pin1),encoderA1,RISING);
attachInterrupt(digitalPinToInterrupt(encA_pin2),encoderA2,RISING);
attachInterrupt(digitalPinToInterrupt(encA_pin3),encoderA3,RISING);

Serial.begin(9600);
}

```

```
//========== LOOP ==================
void loop() {
  // put your main code here, to run repeatedly:

  time = millis();

  //Poll innermost motor to circumvent limited number
  //of Arduino UNO interupts
  n = digitalRead(encA_pin1);
   if ((encA_pin1_prev == LOW) && (n == HIGH)) {
     if (digitalRead(encB_pin1) == LOW) {
       encA_ticks1 -= 1.0;
     } else {
       encA_ticks1 += 1.0;
     }
   }
   encA_pin1_prev = n;

  digitalWrite(13,state);

  //if enough time has passed, print a status update
  //print status update every .5 seconds

  //output the valid configurations to Serial which will then be stored in a file
  encA_degree1 = encA_ticks1*2.25;
  encA_degree2 = encA_ticks2*2.25;
  encA_degree3 = encA_ticks3*2.25;

  Serial.println("0");
  Serial.println(encA_degree1);
  Serial.println(encA_degree2);
  Serial.println(encA_degree3);
  Serial.println(";");

  delay(500);
}

//Arduino UNO only has two interupts
//Switch innermost motor to polling

//void encoderA1(){
//  state=!state;
//  if(digitalRead(encB_pin1)==HIGH){
//    encA_ticks1=encA_ticks1+1.0;
//  }else{
//    encA_ticks1=encA_ticks1-1.0;
//  }
//  }

void encoderA2(){
  state=!state;
  if(digitalRead(encB_pin2)==HIGH){
    encA_ticks2=encA_ticks2+1.0;
  }else{
    encA_ticks2=encA_ticks2-1.0;
  }
}

void encoderA3(){
  state=!state;
  if(digitalRead(encB_pin3)==HIGH){
    encA_ticks3=encA_ticks3+1.0;
  }else{
    encA_ticks3=encA_ticks3-1.0;
```

```
  }
  }
```

11.1.2 parser.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <time.h>
#include <unistd.h>
#include <math.h>

using namespace std;

/*
    ********************************************************************************************

 * Program: C-Space Data Parser
 * Author: Helena Bales
 * Date: April 12th, 2017
 * Description: A C++ implementation of a parser for the Hephaestus Configuration Space data. The
 *  parser creates a 4-D array of boolean values with a 0 representing a valid configuration
 *  and a 1 representing an invalid configuration.
 * Input: ./test1/data1 - Hephaestus C-Space data as defined in project docs
 * Output: ./cspace.out - Output file containing boolean data from 37x37x37x37 array

    ********************************************************************************************
    */

int main() {
  char* myFile;
  ifstream sourceFile;
  ofstream outfile;
  string line;

  int accuracyFactor = 10;
  int MAX_ANGLE = 360;
  int x = MAX_ANGLE/accuracyFactor;
  int a, b, c, d;
  float af, bf, cf, df;
  bool endOfFile = 0;

  unsigned char CSpace[37][37][37][37];

  //init C-Space with 1's
  for(a=0; a<x; a++) {
    for(b=0; b<x; b++) {
      for(c=0; c<x; c++) {
        for(d=0; d<x; d++) {
          CSpace[a][b][c][d] = 255;
        }
      }
    }
  }

  //open file
  myFile = "./test1/data1";
  sourceFile.open(myFile);
```

```
53
54  //loop for length of file
55  while(!endOfFile) {
56    //read 5 lines from the file
57
58    //line 1 == a
59    getline(sourceFile, line);
60    af = stof(line, NULL);
61    //cout << "af = " << af << endl; //FOR TESTING
62
63    //line 2 == b
64    getline(sourceFile, line);
65    bf = stof(line, NULL);
66    //cout << "bf = " << bf << endl; //FOR TESTING
67
68    //line 3 == c
69    getline(sourceFile, line);
70    cf = stof(line, NULL);
71    //cout << "cf = " << cf << endl; //FOR TESTING
72
73    //line 4 == d
74    getline(sourceFile, line);
75    df = stof(line, NULL);
76    //cout << "df = " << df << endl; //FOR TESTING
77
78    //line 5 == ; divider between coordinates
79    getline(sourceFile, line);
80    //cout << ";" << endl;
81
82    //convert negative degrees to corresponding positive angle
83    if(af < 0) {
84      af = 360 + af;
85    }
86
87    if(bf < 0) {
88      bf = 360 + bf;
89    }
90
91    if(cf < 0) {
92      cf = 360 + cf;
93    }
94
95    if(df < 0) {
96      df = 360 + df;
97    }
98
99    //convert to ints
100   a = floor(af/10);
101   b = floor(bf/10);
102   c = floor(cf/10);
103   d = floor(df/10);
104
105   //check floored values against original FOR TESTING
106   cout << a << "/" << af << " - " << b << "/" << bf << " - " << c << "/" << cf << "
      - " << d << "/" << df << endl;
107
108   //mark array[a][b][c][d] = 0
109   CSpace[a][b][c][d] = 0;
110
111   //check if eof has been reached
112   if(sourceFile.eof()) {
113     cout << "Reached eof - check 3" << endl;
114     endOfFile = 1;
115   }
```

```
116
117     //check if eof has been reached
118     if(sourceFile.peek() == 10) {
119       cout << "Reached eof - check 2" << endl;
120       endOfFile = 1;
121     }
122   }
123
124   //close source file
125   if(sourceFile.is_open()) {
126     sourceFile.close();
127   }
128
129   /*
130   //print array
131   for(a=0; a<1; a++) {
132     for(b=0; b<37; b++) {
133       for(c=0; c<37; c++) {
134         for(d=0; d<37; d++) {
135           cout << CSpace[a][b][c][d];
136         }
137         cout << endl;
138       }
139       cout << endl;
140       cout << endl;
141     }
142   }
143   */
144
145   //store array in file
146   outfile.open("cspace");
147   for(a=0; a<x; a++) {
148     for(b=0; b<x; b++) {
149       for(c=0; c<x; c++) {
150         for(d=0; d<x; d++) {
151           outfile << (int)CSpace[a][b][c][d];
152         }
153       }
154     }
155   }
156
157   /*
158   //store 0 locations in file for plotting
159   outfile.open("cspacePlotData");
160   for(a=0; a<x; a++) {
161     for(b=0; b<x; b++) {
162       for(c=0; c<x; c++) {
163         for(d=0; d<x; d++) {
164           if(CSpace[a][b][c][d] == 0){
165             outfile << b << " " << c << " " << d
166               << endl;
167
168           }
169         }
170       }
171     }
172     outfile << "\n";
173   }
174   */
175
176 }
```

11.1.3 convert.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string.h>
#include <cstring>
#include <time.h>
#include <unistd.h>
#include <math.h>

using namespace std;

/*
    ********************************************************************************************

 * Program: Target point converter
 * Author: Helena Bales (Inverse Kinematics by Brett Moffatt)
 * Date: May 17th, 2017
 * Description: For the RockSat-X Hephaestus payload. Converts our target points from cartesian
 *  form into a representation within the configuration space. This is accomplished through
 *  Inverse Kinematics with theta0 isolated to four discrete values, allowing us to resolve
 *  the results of IK from a 4D area in the configuration space to four possible solutions,
 *  of which at least one must be marked as 0 in the cspace.
 * Input: ./targets - A text document containing one target (in the form: x y z) per line, in
 *    cartesian form
 * Output: ./thetaTargets - A text document containing one target per line (in the form:
 *  theta0 theta1 theta2 theta3), where theta0-3 refers to the rotational angle from the
 *  defined normal

    ********************************************************************************************
    */

/**********************************
 * theta1 = roll = {0, 90, 180, 270}
 * theta2 = yaw = tan-1(yo/xo)
 * theta3 = pitch1
 * theta4 = pitch2
 **********************************/

int main() {
  char* myFile;
  ifstream sourceFile;
  ofstream outfile;
  char line[24];
  char* tokens;

  int xo, yo, zo; //real target point
  int theta0, theta1, theta2, theta3; //cspace target point
  double l1, l2; //arm segment lengths
  double k1, k2, gamma; //variables for calculating pitch
  double t, b, fraction1, pt1, pt2; //some more variables for calculations
  bool endOfFile = 0; //marker for eof

  //define arm lengths
  l1 = 3.99;
  l2 = 4.49;

```

```
54  //open the file of targets
55  myFile = "./targets";
56  sourceFile.open(myFile);
57
58  //open the file for theta targets
59  myFile = "./thetaTargets";
60  outfile.open(myFile);
61
62
63  while(!endOfFile) {
64    //read a line
65    sourceFile.getline(line, 24);
66
67    //break the line on spaces
68    tokens = strtok(line, " ");
69
70    //store the values as integers
71    xo = stoi(tokens);
72    tokens = strtok(NULL, " ");
73
74    yo = stoi(tokens);
75    tokens = strtok(NULL, " ");
76
77    zo = stoi(tokens);
78    tokens = strtok(NULL, " ");
79
80    //check if it is eof
81    if(sourceFile.eof() || sourceFile.peek() == 10) {
82      endOfFile = 1;
83      cout << "end of file reached" << endl;
84    }
85
86    //calculate variables
87    t = (xo*xo) + (yo*yo) - (l1*l1) - (l2*l2);
88    b = 2 * l1 * l2;
89    fraction1 = t/b;
90    pt1 = sqrt(1 - (fraction1 * fraction1));
91    pt2 = fraction1;
92
93    //calcualte theta2
94    theta2 = atan2(zo, sqrt((yo*yo) + (xo*xo)));
95
96    //calculate k1, k2, and gamma
97    k1 = l1 + (l2 * cos(theta2));
98    k2 = l2 * sin(theta2);
99    gamma = atan2(k1, k2);
100
101   //calculate theta0, theta1, theta3
102   theta1 = atan(yo/xo);
103   theta3 = atan2(yo, xo) - gamma;
104   theta0 = 0; //TODO move this motor to change the plane
105
106   //print the thetas to the file
107   outfile << theta0 << " " << theta1 << " " << theta2 << " " << theta3 << " " <<
      endl;
108
109 }
110
111 //close the file
112 if(sourceFile.is_open()) {
113   sourceFile.close();
114 }
115
116 //close the file
```

```
  if(outfile.is_open()) {
    outfile.close();
  }


  return 0;
}
```

11.1.4 pathing.cpp

11.2 Data Storage

11.2.1 SDRead.py

```
import matplotlib.pyplot as plt
import sys

if len(sys.argv) != 2:      # Make sure filename is specified on command line
  print("No file specified.")
  print("USAGE:", sys.argv[0], "<filename>")
  sys.exit(1)

temps = []              # Create an empty list of temperatures

with open(sys.argv[1], "br") as f:  # Open output file for binary reading
  data = bytes(f.read(3))     # Read three bytes
  while data != bytes(b""):   # Make sure we have input (i.e. not EOF)
    num = (data[0] * pow(2, 8)) # Load the most significant byte
    num += (data[1])      # Load the least significant byte
    temps.append(num)     # Store the number in the array
    data = bytes(f.read(3))   # Get thee next three bytes

scaled = [(i/(2**10)) for i in temps] # Scale the temperature values to between 0-1
plt.plot(scaled)          # Create the plot
plt.ylabel("Relative temperature")  # Always label your axises
plt.xlabel("Sample number")
plt.show()              # Display the plot
```

11.2.2 telemetry.c

```
/*
 * telemetry.c
 *
 * Created: 1/18/2017 10:44:49 AM
 *  Author: Michael Humphrey
 */

#include "telemetry.h"
#include "RSXAVRD.h"
#include <avr/eeprom.h>
#include <string.h>

uint16_t current_address;

void telemetry_init(void) {
  current_address = eeprom_read_word(0);
  if (current_address == 0xFFFF) {
    current_address = 2;
  }
```

```
}

// Sends a code with the predefined TELEMETRY_TIME constant.
void inline telemetry_send_code(uint8_t code) {
  send_code(code, TELEMTRY_TIME);
}

// Log a message to the EEPROM
void eeprom_log(char* message) {
  eeprom_update_block(message, (void *) (uint16_t) current_address, strlen(message)
    +1);
  current_address += strlen(message)+1;
  eeprom_update_word(0, current_address);
}


// Log a message to the SD card - DEPRECATED
/*
void SD_log(char* message) {
  // Write elapsed time to SD card
  char *result = (char*) malloc(6*sizeof(char)); // Max value of get_time() is 65535,
     which is 5 characters, plus 1 for null terminator

  if (result == NULL)
    return;

  itoa(get_time(), result, 10);
  // Write result string to SD card
  free(result);

  // Write separator to SD card

  // Write message to SD card

  // Write line terminator to SD card
}
*/
```

11.3 Main

11.3.1 main.c

```
/*
 * Hephaestus.c
 *
 * Created: 1/17/2017 6:05:07 PM
 * Author : Michael Humphrey
 */

#include <avr/io.h>
#include "phases.h"
#include "RSXAVRD.h"
#include "retract.h"


int main(void)
{
  // Pin configuration
  AVR_init();

  uint8_t status = 0;

```

```
  //timer_counter_enable(1,1); // code that enables timer event - Jon says Michael wanted this?

  timer_event_enable(0,1); // enables timer event line 0

  timer_event_enable(1,1); // enables timer event line 1

    /* Replace with your application code */
    while (1)
    {
    // Phase 1: Idle
    status = idle();

    // Phase 2: Observation
    status = observation();

    // Phase 3: Science
    status = science();
    if (status != 0) { // if our arm is not calibrated i.e. collapsed and in home position...
      retract(); // turn off all motors and retract into a safe position
    }

    // Phase 4: Off
    status = off();
    // Under normal circumstances, off never returns.
    // Off only returns if there was an error.

    // Phase 5: Safety
    status = safety();
    }
}
```

11.3.2 phases.h

```
/*
 * phases.h
 *
 * Created: 1/20/2017 3:36:58 PM
 *  Author: Michael Humphrey
 */


#ifndef PHASES_H_
#define PHASES_H_

int idle(void);      // Phase 1
int observation(void);  // Phase 2
int science(void);    // Phase 3
int off(void);       // Phase 4
int safety(void);   // Phase 5
void retract(void);  // retract phase

#endif /* PHASES_H_ */
```

11.3.3 Modes of Operation

11.3.3.1 idle.c

```
/*
 * idle.c
 *
```

```
 * Created: 1/20/2017 3:36:32 PM
 *  Author: Michael Humphrey
 */

#include "RSXAVRD.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include "telemetry.h"

uint8_t ready = 0;

ISR(INT6_vect) {
  ready = 1;
}

 // First phase of payload deployment. Wait for TE-R lines to go high, then return.
 int idle(void) {

  // Infinite loop until receive signal to start


  while (!ready) {}

  eeprom_log("idle phase complete");

  return 0;
}
```

11.3.3.2 observation.c

```

/*  observation.c
 *  Created: 1/22/2017 5:12:25 PM
 *  Author: Amber Horvath
 */

#include <avr/io.h>
#include "phases.h"
#include "RSXAVRD.h"
#include "telemetry.h"
#include "MOTOR_DEF.h"

int observation(){


      camera_enable(POWER_ON); // turns on camera

      motor_pwr(MOTOR_CAMERA, POWER_ON); // power on the motor for the camera

      motor_dir(MOTOR_CAMERA, CLOCKWISE); // set the camera to move clockwise

      motor_step(MOTOR_CAMERA, DEGREES_TO_STEPS(180), 1, 85);

      _delay_ms(500);

      motor_dir(MOTOR_CAMERA, COUNTER_CLOCKWISE);

      motor_step(MOTOR_CAMERA, DEGREES_TO_STEPS(360), 1, 85);

      _delay_ms(500);

      motor_dir(MOTOR_CAMERA, CLOCKWISE);

```

```
34        motor_step(MOTOR_CAMERA, DEGREES_TO_STEPS(180), 1, 85);
35
36        _delay_ms(500);
37
38        telemetry_send_code(OBSERVATION_PHASE); // let us know we finished Observation
      mode
39
40        eeprom_log("observation phase complete");
41
42        return 0;
43
44
45 }
```

11.3.3.3 science.c

```
1  #include <avr/io.h>
2  #include "phases.h"
3  #include "RSXAVRD.h"
4  #include "telemetry.h"
5  #include <avr/interrupt.h>
6  #include <util/delay.h>
7  /*
8  * Design of science phase: arm will move to touch sensor and press it. Upon moving
       the arm to the location, we will
9  * check whether the touch sensor was pressed and change our status to represent
       whether or not it was pressed.
10 * We will also write a code over the telemetry lines to represent whether or not the
       touch sensor was pressed.
11 */
12
13 int science(){
14   int status = 0;
15   int M1_POS, M2_POS, M3_POS, M4_POS;
16   int M1_NXT, M2_NXT, M3_NXT, M4_NXT;
17   int M1_DIF, M2_DIF, M3_DIF, M4_DIF;
18   int M1_STP, M2_STP, M3_STP, M4_STP;
19   int scale_factor = 0.225;
20   char path_step;
21
22   //TODO reference CSpace in program memory (the address of a 4D array in program
       memory)
23   char**** cspace;
24
25   // init motor values to 0
26   M1_POS = 0;
27   M2_POS = 0;
28   M3_POS = 0;
29   M4_POS = 0;
30
31   // turn on camera
32   camera_enable(1);
33
34   // power on motors
35   motor_pwr(1, 1);
36   motor_pwr(2, 1);
37   motor_pwr(3, 1);
38   motor_pwr(4, 1);
39
40   //repeat for length of path
41   while(1) {
42     //set motor direction to be forward
43     motor_dir(1, 0);
```

```
    motor_dir(2, 0);
    motor_dir(3, 0);
    motor_dir(4, 0);

    //find the next point in the path
    path_step = cspace[M1_POS][M2_POS][M3_POS][M4_POS];

    //set the lower and upper bounds for the loops so only indeces
    //in range are checked
    if(M1_POS == 0) { sw = 0; } else { sw = -1; }
    if(M2_POS == 0) { sx = 0; } else { sx = -1; }
    if(M3_POS == 0) { sy = 0; } else { sy = -1; }
    if(M4_POS == 0) { sz = 0; } else { sz = -1; }

    if(M1_POS == 36) { ew = 1; } else { ew = 2; }
    if(M2_POS == 36) { ex = 1; } else { ex = 2; }
    if(M3_POS == 36) { ey = 1; } else { ey = 2; }
    if(M4_POS == 36) { ez = 1; } else { ez = 2; }


    //check each of the neighboring points
    for(int w=sw; w<ew; w++) {
      for(int x=sx; x<ex; x++) {
        for(int y=sy; y<ey; y++) {
          for(int z=sz; z<ez; z++) {
            if(cspace[M1_POS+w][M2_POS+x][M3_POS+y][M4_POS+z] == path_step + 1) {
              M1_NXT = M1_POS + w;
              M2_NXT = M2_POS + x;
              M3_NXT = M3_POS + y;
              M4_NXT = M4_POS + z;
              break;
            }
          }
        }
      }
    }

    if(M1_NXT == M1_POS && M2_NXT == M2_POS && M3_NXT == M3_POS && M4_NXT == M4_POS)
    {
      //end of path has been reached
      //TODO send eop telemetry signal
      return 0;
    }

    //change _NXT values to be less than 360
    if(M1_NXT > 359) {
      M1_NXT = M1_NXT - 359;
    }

    if(M2_NXT > 359) {
      M2_NXT = M2_NXT - 359;
    }

    if(M3_NXT > 359) {
      M3_NXT = M3_NXT - 359;
    }

    if(M4_NXT > 359) {
      M4_NXT = M4_NXT - 359;
    }

    // calculate change in each motor (in degrees)
    M1_DIF = M1_NXT - M1_POS;
    M2_DIF = M2_NXT - M2_POS;
```

```
107     M3_DIF = M3_NXT - M3_POS;
108     M4_DIF = M4_NXT - M4_POS;
109
110     // reverse motor direction for negative difference
111     if(M1_DIF < 0) {
112       motor_dir(1, 1);
113       M1_DIF = M1_DIF * -1;
114     }
115
116     if(M2_DIF < 0) {
117       motor_dir(2, 1);
118       M2_DIF = M2_DIF * -1;
119     }
120
121     if(M3_DIF < 0) {
122       motor_dir(3, 1);
123       M3_DIF = M3_DIF * -1;
124     }
125
126     if(M4_DIF < 0) {
127       motor_dir(4, 1);
128       M4_DIF = M4_DIF * -1;
129     }
130
131     // convert degress to steps
132     M1_STP = M1_DIF / scale_factor;
133     M2_STP = M2_DIF / scale_factor;
134     M3_STP = M3_DIF / scale_factor;
135     M4_STP = M4_DIF / scale_factor;
136
137     // move each motor that many steps
138     motor_step(1, M1_STP, 28, 99);
139     motor_step(2, M2_STP, 28, 99);
140     motor_step(3, M3_STP, 28, 99);
141     motor_step(4, M4_STP, 28, 99);
142
143     // set the current position
144     M1_POS = M1_NXT;
145     M2_POS = M2_NXT;
146     M3_POS = M3_NXT;
147     M4_POS = M4_NXT;
148
149     if(touch_sensor_check() == 0x01){
150       telemetry_send_code(TOUCH_SENSOR_1_ENGAGED); // For now, always send
      TOUCH_SENSOR_1_ENGAGED.
151     }
152
153     // code to return arm to callibrated "home" position
154     if(get_calibration_status() != 0x01){ // or whichever motor refers to the home
      position
155       status = 1;
156     }
157
158   }
159
160   return status;
161
162 }
```

11.3.3.4 retract.c

```
1 #include <avr/io.h>
2 #include <avr/interrupt.h>
```

```
#include <util/delay.h>
#include "RSXAVRD.h"
#include "phases.h"
#include "retract.h"
#include "MOTOR_DEF.h"
#include "telemetry.h"

uint8_t plate_retracted_flg; // flag to keep track of plate's position

ISR(INT5_vect){

  if(plate_retracted_flg == 0x00){
    retract();
  }

}

void retract(){

  motor_pwr(MOTOR_DECK_PLATE, POWER_ON);

  _delay_ms(500); // delay for motor after powering on

  motor_pwr(MOTOR_CAMERA, POWER_OFF); // turn off all other motors
  motor_pwr(MOTOR_DECK_ARM, POWER_OFF);
  motor_pwr(MOTOR_PAN, POWER_OFF);
  motor_pwr(MOTOR_SHOULD, POWER_OFF);
  motor_pwr(MOTOR_ELB, POWER_OFF);

  camera_enable(POWER_OFF);

  motor_dir(MOTOR_DECK_PLATE, COUNTER_CLOCKWISE); // rotates the deck plate to

  motor_step(MOTOR_DECK_PLATE, 1650, 28, SPEED + 19); // the amount of steps needed
    to pull the arm back in

  eeprom_log("deck plate has been retracted");

  plate_retracted_flg = 0x01;


}

void extend(){

  motor_pwr(MOTOR_DECK_PLATE, POWER_ON); // powers on deck plate motor

  _delay_ms(500); // delays to allow motor to power on

  motor_dir(MOTOR_DECK_PLATE, CLOCKWISE); // push the deck plate out

  motor_step(MOTOR_DECK_PLATE, 1650, 28, SPEED + 19); // the amount of steps needed
    to move the deck plate at a good speed

  plate_retracted_flg = 0x00; // plate is NOT retracted


}
```

11.3.3.5 safety.c

```
/* safety.c
 *
```

```
 * Created by: Amber Horvath
 * Date Created: 1/22/17 5:36:47
 *
 *
 */

#include <avr/io.h>
#include "phases.h"
#include "retract.h"
#include "telemetry.h"

int safety(void){

  eeprom_log("in safety - attempting retract");

    retract();

    while(1){}; // abort mission, we are hanging here

}
```

11.3.3.6 off.c

```
#include "telemetry.h"

int off(void) {

  eeprom_log("power off");

  while(1){};

  // This return statement should never be reaches, so return error.
  return 1;
}
```

12 Appendix 2: Other Documents

12.1 Mission Logo

12.2 Team Photos

12.3 CAD Models

12.4 Launch Compliance

13 Glossary

Glossary

API Application Programming Interface. The set of functions and classes that a given library exposes for other programs to make use of its provided functionality. 35, 37, 73

apogee The point at which the rocket has finished its ascent and payloads are allowed to deploy. 26, 73

Arm Assembly The Hephaestus Arm Assembly includes the arm, the rotating arm base, the camera, and base. It is the portion of the payload that is extended during Science mode. 73

ASCII American Standard Code for Information Interchange. Each alphabetic, numeric, or special character is represented with a 7-bit binary number. 128 possible characters are defined. 25, 73

binary string An ordered sequence of 1's and 0's. 25, 73

can A can is a segment of the rocket in which payloads can be placed. A can constitutes a standard length of rocket, defined by the RockSat-X program. 73

configuration space The Configuration Space (or C-Space) is a 4 dimmentional space with a mapping to 3D real space that is used to represent the possible configurations of the arm's motors. The C-Space is stored in a 4D array of characters. Possible valid configurations are marked as such in the C-Space and this data is used to plot the arm's path. 73

deployable Any portion of the payload that is expanded from its original configuration once in a space-like environment. 12, 73

GUI Graphical User Interface 26, 73

matplotlib A Python library for drawing and manipulating graphs. 26, 73

npm npm is a package manager for NodeJS, a javascript library. It is used to install various 'npm' packages for NodeJS servers. 37, 73

OBC On-Board Computer 22–25, 73

OSU Oregon State University 1, 73

payload A subsection of a rocket that is not essential to the rocket's operation. A payload is placed in a can, mounted on a standard base plate. A payload completes some specific task. 1, 12, 25, 26, 34, 35, 73

plot An interactive window generated by the Python library matplotlib to display some dataset on an x and y axis. 26, 73

port To transfer software from one system or machine to another. 35, 73

PSAS Portland State Aerospace Society 35, 73

PSU Portland State University 35, 73

replay To replay a dataset is to reproduce preexisting data in a manner that simulates how it was generated, e.g. output the data in the same timeline that each data point was generated. 73

TE-1 The TE-1 line is an electrical input to the system that enables at a predetermined time during flight. 73

TE-R The TE-R line is a redundant electrical input to the system that enables at a predetermined time during flight. 73

WFF Wallops Flight Facility 73