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

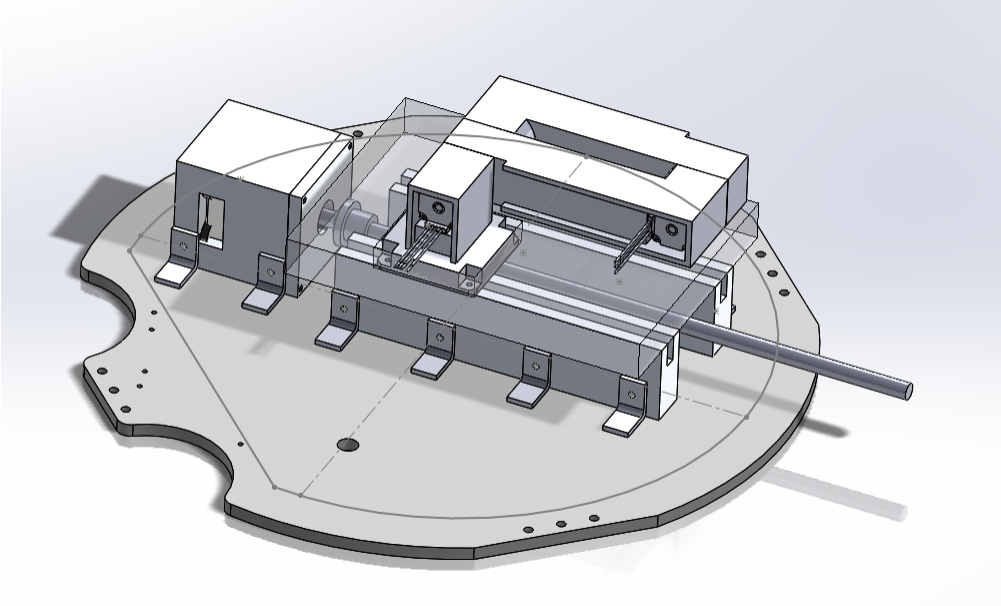


Figure 1: Model of Hephaestus Payload

- 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

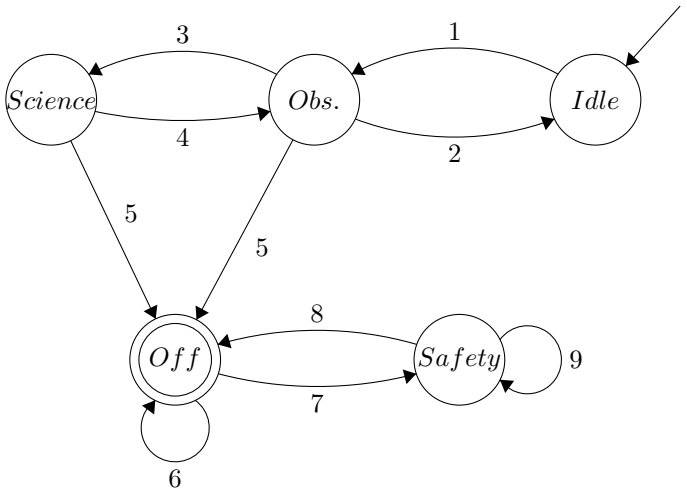
Priority 1: The software shall report all telemetry data (as defined in 2.5) to the ground station.

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

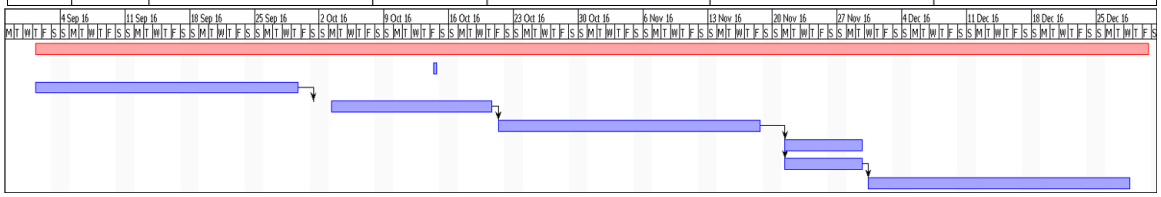
Priority 1: The system shall be secure. Since it is a closed system, the device will be programmed such that it cannot be accessed remotely and will only output sanitized data.

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

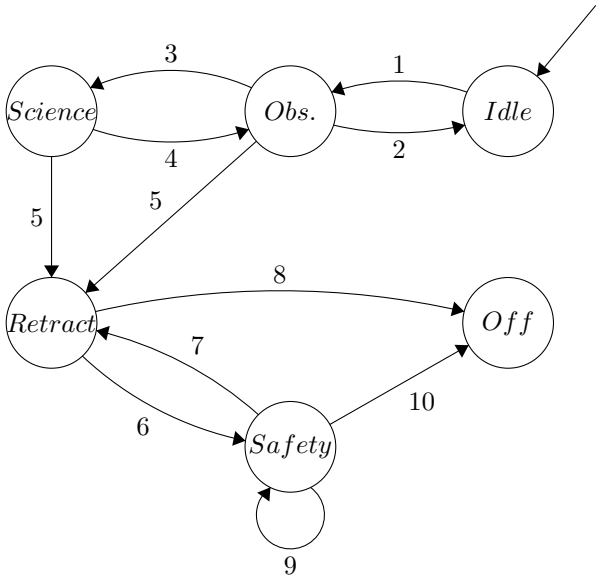


Diagram of software states of operation and transition between states [2].

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.

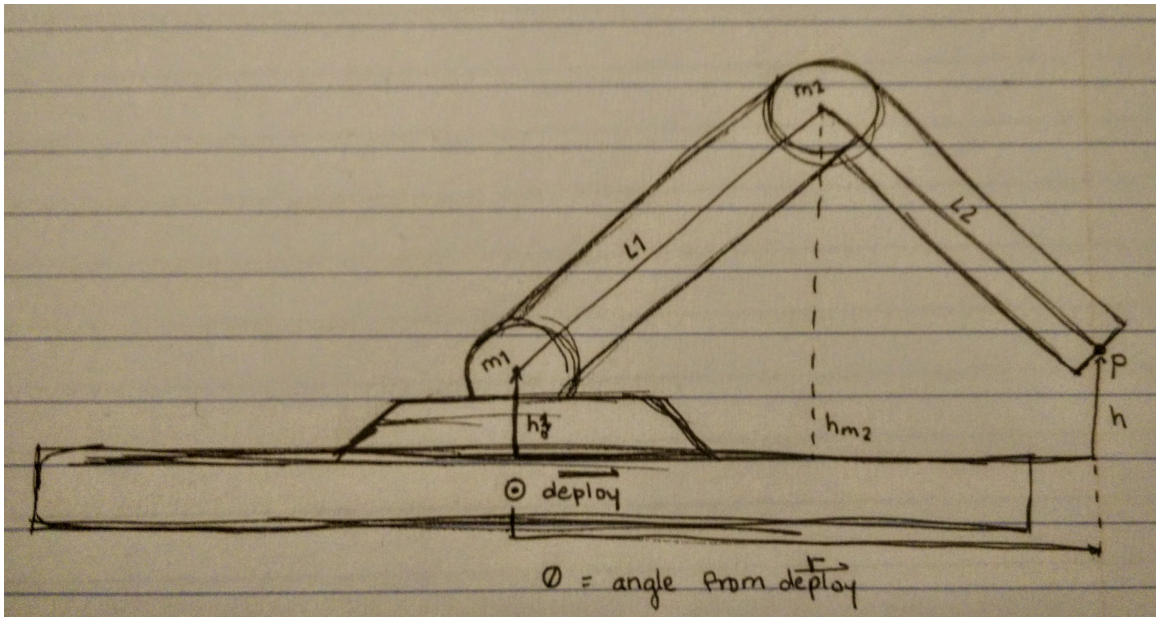


Figure 1 – Arm Movement Design

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

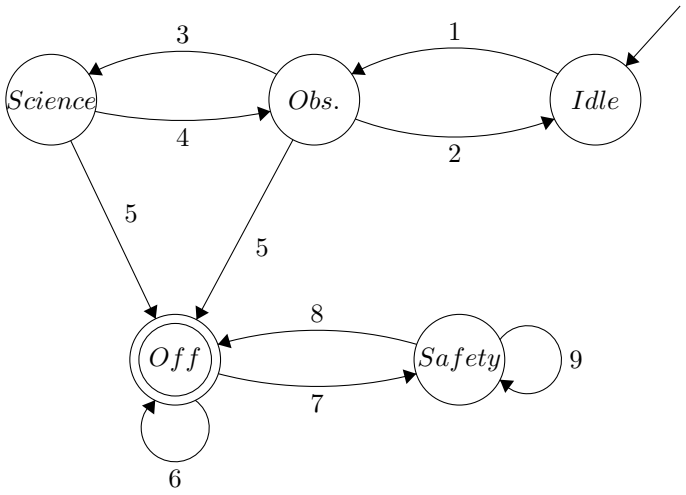


Diagram of software states of operation and transition between states [2].

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 4

6.1.1.1 Helena Bales

6.1.1.2 Amber Horvath

6.1.1.3 Michael Humphrey

6.1.2 Week 5

6.1.2.1 Helena Bales

6.1.2.2 Amber Horvath

6.1.2.3 Michael Humphrey

6.1.3 Week 6

6.1.3.1 Helena Bales

6.1.3.2 Amber Horvath

6.1.3.3 Michael Humphrey

6.1.4 Week 7

6.1.4.1 Helena Bales

6.1.4.2 Amber Horvath

6.1.4.3 Michael Humphrey

6.1.5 Week 8

6.1.5.1 Helena Bales

6.1.5.2 Amber Horvath

6.1.5.3 Michael Humphrey

6.1.6 Week 9

6.1.6.1 Helena Bales

6.1.6.2 Amber Horvath

6.1.6.3 Michael Humphrey

6.1.7 Week 10

6.1.7.1 Helena Bales

6.1.7.2 Amber Horvath

6.3.1.2 Amber Horvath

6.3.1.3 Michael Humphrey

6.3.2 Week 2

6.3.2.1 Helena Bales

6.3.2.2 Amber Horvath

6.3.2.3 Michael Humphrey

6.3.3 Week 3

6.3.3.1 Helena Bales

6.3.3.2 Amber Horvath

6.3.3.3 Michael Humphrey

6.3.4 Week 4

6.3.4.1 Helena Bales

6.3.4.2 Amber Horvath

6.3.4.3 Michael Humphrey

6.3.5 Week 5

6.3.5.1 Helena Bales

6.3.5.2 Amber Horvath

6.3.5.3 Michael Humphrey

6.3.6 Week 6

6.3.6.1 Helena Bales

6.3.6.2 Amber Horvath

6.3.6.3 Michael Humphrey

6.3.7 Week 7

6.3.7.1 Helena Bales

6.3.7.2 Amber Horvath

6.3.7.3 Michael Humphrey

6.3.8 Week 8

6.3.8.1 Helena Bales

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

11.1.2 parser.cpp

11.1.3 convert.cpp

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, 44

apogee The point at which the rocket has finished its ascent and payloads are allowed to deploy. 26, 44

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. 44

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, 44

binary string An ordered sequence of 1's and 0's. 25, 44

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. 44

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. 44

deployable Any portion of the payload that is expanded from its original configuration once in a space-like environment. 12, 44

GUI Graphical User Interface 26, 44

matplotlib A Python library for drawing and manipulating graphs. 26, 44

npm npm is a package manager for NodeJS, a javascript library. It is used to install various 'npm' packages for NodeJS servers. 37, 44

OBC On-Board Computer 22–25, 44

OSU Oregon State University 1, 44

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, 44

plot An interactive window generated by the Python library matplotlib to display some dataset on an x and y axis. 26, 44

port To transfer software from one system or machine to another. 35, 44

PSAS Portland State Aerospace Society 35, 44

PSU Portland State University 35, 44

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. 44

TE-1 The TE-1 line is an electrical input to the system that enables at a predetermined time during flight. 44

TE-R The TE-R line is a redundant electrical input to the system that enables at a predetermined time during flight. 44

WFF Wallops Flight Facility 44