Final Report For RockSat-X Payload - Hephaestus

Helena Bales, Amber Horvath, and Michael Humphrey

CS463 - Spring 2017

June 12, 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 **3**

1.1 Document Overview . . . 3

2 Project Overview **4**

2.1 Project Purpose . . . 4

2.2 Mission Success Criteria . . . 4

2.2.1 Minimum Mission Success Criteria . . . 4

2.2.2 Maximum Mission Success Criteria . . . 4

2.3 Concept of Operations . . . 5

2.4 Programmatics . . . 6

2.4.1 Organizational Chart . . . 6

2.4.2 Sponsors . . . 6

3 Requirements Document **6**

3.1 Original Requirements Document . . . 6

3.1.1 Introduction . . . 6

3.1.1.1 Purpose of Document . . . 6

3.1.1.2 Overview of Document . . . 7

3.1.1.3 Overview of Payload . . . 7

3.1.1.4 Overview of Physical Payload . . . 7

3.1.1.5 Mission Success Criteria . . . 8

3.1.1.5.1 Minimum Mission Success Criteria . . . 8

3.1.1.5.2 Maximum Mission Success Criteria . . . 8

3.1.1.6 Requirements Apportioning . . . 9

3.1.1.6.1 Priority 1 . . . 9

3.1.1.6.2 Priority 2 . . . 9

3.1.1.6.3 Priority 3 . . . 9

3.1.2 Functional Requirements . . . 9

3.1.2.1 Main Behavior . . . 9

3.1.2.2 Target Generation . . . 9

3.1.2.3 Movement . . . 9

3.1.2.4 Modes . . . 10
3.1.2.4.1 Launch . . . 10
3.1.2.4.2 Deployment . . . 10
3.1.2.4.3 Science . . . 10
3.1.2.4.4 Safety . . . 10
3.1.2.4.5 Observation . . . 10
3.1.2.4.6 Power Off . . . 10
3.1.2.4.7 State Diagram . . . 11
3.1.2.5 Telemetry . . . 11
3.1.3 Non Functional Requirements . . . 11
3.1.3.1 Performance . . . 11
3.1.3.2 Security . . . 11
3.1.3.3 Telemetry . . . 12
3.1.4 Gantt Chart . . . 12
3.2 Changes Since Original Requirements Document . . . 12
3.3 Final Gantt Chart . . . 12

4 Design Document **13**

4.1 Original Design Document . . . 13
4.1.1 Introduction . . . 13
4.1.1.1 Document Overview . . . 13
4.1.1.1.1 Helena Bales . . . 13
4.1.1.1.2 Amber Horvath . . . 13
4.1.1.1.3 Michael Humphrey . . . 13
4.1.2 Technologies . . . 13
4.1.2.1 Target Generation . . . 13
4.1.2.1.1 Requirement Overview . . . 13
4.1.2.1.2 Solution Design . . . 13
4.1.2.2 Arm Movement . . . 14
4.1.2.2.1 Requirement Overview . . . 14
4.1.2.2.2 Solution Design . . . 14
4.1.2.3 Arm Position Tracking . . . 15

4.1.2.3.1 Requirement Overview . . . 15
4.1.2.3.2 Solution Design . . . 15
4.1.2.4 Emergency Payload Expulsion . . . 15
4.1.2.4.1 Requirement Overview . . . 15
4.1.2.4.2 Solution Design . . . 16
4.1.2.5 Program Modes of Operation . . . 16
4.1.2.5.1 Requirement Overview . . . 16
4.1.2.5.2 Solution Design . . . 17
4.1.2.6 Target Success Sensors . . . 18
4.1.2.6.1 Requirement Overview . . . 18
4.1.2.6.2 Solution Design . . . 18
4.1.2.7 Telemetry . . . 19
4.1.2.7.1 Requirement Overview . . . 19
4.1.2.7.2 Solution Design . . . 19
4.1.2.8 Video Handling . . . 19
4.1.2.8.1 Requirement Overview . . . 19
4.1.2.8.2 Solution Design . . . 19
4.1.2.9 Data Visualization and Processing . . . 20
4.1.2.9.1 Requirement Overview . . . 20
4.1.2.9.2 Solution Design . . . 20
4.1.3 Conclusion . . . 20
4.2 Changes Since Original Design Document . . . 20
4.2.1 Target Generation . . . 20
4.2.2 Arm Movement . . . 21
4.2.3 Arm Position Tracking . . . 21
4.2.4 Emergency Payload Expulsion . . . 21
4.2.5 Program Modes of Operation . . . 21
4.2.6 Target Success Sensors . . . 21
4.2.7 Telemetry . . . 22
4.2.8 Video Camera . . . 22
4.2.9 Data Visualization and Processing . . . 22

5 Technical Review Document **23**

5.1 Original Technical Review Document . . . 23

5.1.1 Introduction . . . 23

5.1.1.1 Document Overview . . . 23

5.1.1.2 Role Breakdown . . . 23

5.1.1.2.1 Helena Bales . . . 23

5.1.1.2.2 Amber Horvath . . . 23

5.1.1.2.3 Michael Humphrey . . . 23

5.1.2 Technologies . . . 24

5.1.2.1 Target Generation . . . 24

5.1.2.1.1 Requirement Overview . . . 24

5.1.2.1.2 Proposed Solutions . . . 24

5.1.2.2 Arm Movement . . . 24

5.1.2.2.1 Requirement Overview . . . 24

5.1.2.2.2 Proposed Solutions . . . 24

5.1.2.3 Arm Position Tracking . . . 26

5.1.2.3.1 Requirement Overview . . . 26

5.1.2.3.2 Proposed Solutions . . . 26

5.1.2.4 Emergency Payload Expulsion . . . 27

5.1.2.4.1 Requirement Overview . . . 27

5.1.2.4.2 Proposed Solutions . . . 27

5.1.2.5 Program Modes of Operation . . . 27

5.1.2.5.1 Requirement Overview . . . 27

5.1.2.5.2 Proposed Solutions . . . 28

5.1.2.6 Target Success Sensors . . . 29

5.1.2.6.1 Requirement Overview . . . 29

5.1.2.6.2 Proposed Solutions . . . 29

5.1.2.7 Telemetry . . . 30

5.1.2.7.1 Requirement Overview . . . 30

5.1.2.7.2 Proposed Solutions . . . 30

5.1.2.8 Video Handling . . . 31

5.1.2.8.1 Requirement Overview . . . 31

5.1.2.8.2 Proposed Solutions . . . 31
5.1.2.9 Data Visualization and Processing . . . 32
5.1.2.9.1 Requirement Overview . . . 32
5.1.2.9.2 Proposed Solutions . . . 32
5.1.3 Conclusion . . . 33
5.2 Changes Since Original Technical Review Document . . . 33

6 Weekly Blog Posts **34**

6.1 Fall 2016 . . . 34
6.1.1 Week 3 . . . 34
6.1.1.1 Helena Bales . . . 34
6.1.1.2 Amber Horvath . . . 34
6.1.1.3 Michael Humphrey . . . 35
6.1.2 Week 4 . . . 35
6.1.2.1 Helena Bales . . . 35
6.1.2.2 Amber Horvath . . . 36
6.1.2.3 Michael Humphrey . . . 36
6.1.3 Week 5 . . . 36
6.1.3.1 Helena Bales . . . 36
6.1.3.2 Amber Horvath . . . 37
6.1.3.3 Michael Humphrey . . . 37
6.1.4 Week 6 . . . 38
6.1.4.1 Helena Bales . . . 38
6.1.4.2 Amber Horvath . . . 38
6.1.4.3 Michael Humphrey . . . 38
6.1.5 Week 7 . . . 39
6.1.5.1 Helena Bales . . . 39
6.1.5.2 Amber Horvath . . . 40
6.1.5.3 Michael Humphrey . . . 40
6.1.6 Week 8 . . . 41
6.1.6.1 Helena Bales . . . 41
6.1.6.2 Amber Horvath . . . 41

6.1.6.3 Michael Humphrey . . . 41
6.1.7 Week 9 . . . 42
6.1.7.1 Helena Bales . . . 42
6.1.7.2 Amber Horvath . . . 42
6.1.7.3 Michael Humphrey . . . 42
6.1.8 Week 10 . . . 43
6.1.8.1 Helena Bales . . . 43
6.1.8.2 Amber Horvath . . . 43
6.1.8.3 Michael Humphrey . . . 44
6.2 Winter 2017 . . . 44
6.2.1 Week 1 . . . 44
6.2.1.1 Helena Bales . . . 44
6.2.1.2 Amber Horvath . . . 44
6.2.1.3 Michael Humphrey . . . 45
6.2.2 Week 2 . . . 45
6.2.2.1 Amber Horvath . . . 45
6.2.3 Week 3 . . . 45
6.2.3.1 Helena Bales . . . 45
6.2.3.2 Amber Horvath . . . 46
6.2.3.3 Michael Humphrey . . . 46
6.2.4 Week 4 . . . 47
6.2.4.1 Helena Bales . . . 47
6.2.4.2 Amber Horvath . . . 47
6.2.4.3 Michael Humphrey . . . 47
6.2.5 Week 5 . . . 48
6.2.5.1 Helena Bales . . . 48
6.2.5.2 Amber Horvath . . . 48
6.2.5.3 Michael Humphrey . . . 49
6.2.6 Week 6 . . . 49
6.2.6.1 Helena Bales . . . 49
6.2.6.2 Amber Horvath . . . 50
6.2.6.3 Michael Humphrey . . . 50

6.2.7 Week 7 . . . 51
6.2.7.1 Helena Bales . . . 51
6.2.7.2 Amber Horvath . . . 51
6.2.7.3 Michael Humphrey . . . 52
6.2.8 Week 8 . . . 52
6.2.8.1 Helena Bales . . . 52
6.2.8.2 Amber Horvath . . . 52
6.2.8.3 Michael Humphrey . . . 53
6.2.9 Week 9 . . . 53
6.2.9.1 Helena Bales . . . 53
6.2.9.2 Amber Horvath . . . 54
6.2.9.3 Michael Humphrey . . . 54
6.2.10 Week 10 . . . 55
6.2.10.1 Helena Bales . . . 55
6.2.10.2 Amber Horvath . . . 57
6.2.10.3 Michael Humphrey . . . 57
6.3 Spring 2017 . . . 58
6.3.1 Week 1 . . . 58
6.3.1.1 Helena Bales . . . 58
6.3.1.2 Amber Horvath . . . 58
6.3.1.3 Michael Humphrey . . . 59
6.3.2 Week 2 . . . 59
6.3.2.1 Helena Bales . . . 59
6.3.2.2 Amber Horvath . . . 59
6.3.2.3 Michael Humphrey . . . 60
6.3.3 Week 3 . . . 60
6.3.3.1 Helena Bales . . . 60
6.3.3.2 Amber Horvath . . . 60
6.3.3.3 Michael Humphrey . . . 61
6.3.4 Week 4 . . . 61
6.3.4.1 Helena Bales . . . 61
6.3.4.2 Amber Horvath . . . 62

6.3.4.3 Michael Humphrey . . . 62
6.3.5 Week 5 . . . 63
6.3.5.1 Helena Bales . . . 63
6.3.5.2 Amber Horvath . . . 63
6.3.5.3 Michael Humphrey . . . 63
6.3.6 Week 6 . . . 64
6.3.6.1 Helena Bales . . . 64
6.3.6.2 Amber Horvath . . . 64
6.3.6.3 Michael Humphrey . . . 65
6.3.7 Week 7 . . . 65
6.3.7.1 Helena Bales . . . 65
6.3.7.2 Amber Horvath . . . 66
6.3.7.3 Michael Humphrey . . . 67

7 Final Poster **68**

8 Project Documentation **69**

8.1 Project Functionality . . . 69
8.1.1 Project Structure . . . 69
8.1.2 Theory of Operation . . . 69
8.1.3 Block Diagram . . . 70
8.1.4 Flow Diagram . . . 71
8.1.4.1 Pathing and Automation Subsystem Flow Diagram . . . 71
8.2 Hardware Requirements . . . 71
8.3 Installation Instructions . . . 72
8.4 Running Instructions . . . 72
8.5 User Guides and Documentation . . . 72
8.5.1 Program Overview . . . 72
8.5.2 Mission Overview . . . 72
8.5.3 Repository Contents . . . 72
8.5.3.1 Deliverables . . . 72
8.5.3.2 Design Documents . . . 73
8.5.3.3 Photos . . . 73

8.5.3.4 Presentations . . . 73
8.5.3.5 Code . . . 73
8.5.4 Instructions to build . . . 73
8.5.4.1 Before you build . . . 73
8.5.4.2 Building . . . 74

9 Learning New Technology **74**
9.1 Helpful Resources . . . 74
9.1.1 Web Sites . . . 74
9.1.2 Books and Print Materials . . . 74
9.1.3 Faculty and Personnel . . . 74

10 What We Learned **74**
10.1 Helena Bales . . . 74
10.1.1 Technical Information . . . 74
10.1.2 Non-Technical Information . . . 75
10.1.3 Project Work Information . . . 75
10.1.4 Project Management Information . . . 75
10.1.5 Team Work Information . . . 76
10.1.6 If you could do it all over what would you do differently? . . . 76
10.2 Amber Horvath . . . 76
10.2.1 Technical Information . . . 76
10.2.2 Non-Technical Information . . . 77
10.2.3 Project Work Information . . . 77
10.2.4 Project Management Information . . . 77
10.2.5 Team Work Information . . . 78
10.2.6 If you could do it all over, what would you do differently? . . . 78
10.3 Michael Humphrey . . . 78
10.3.1 Technical Information . . . 78
10.3.2 Non-Technical Information . . . 79
10.3.3 Project Work Information . . . 79
10.3.4 Project Management Information . . . 79
10.3.5 Team Work Information . . . 79

10.3.6 If you could do it all over what would you do differently? 79

11 Appendix 1: Essential Code **80**

11.1 Pre-Processing 80

11.1.1 CSpace_Mapping.ino 80

11.1.2 parser.cpp 82

11.1.3 convert.cpp 85

11.1.4 pathing.cpp 87

11.2 Data Storage 90

11.2.1 SDRead.py 90

11.2.2 telemetry.c 90

11.3 Main 91

11.3.1 main.c 91

11.3.2 phases.h 92

11.3.3 Modes of Operation 93

11.3.3.1 idle.c 93

11.3.3.2 observation.c 93

11.3.3.3 science.c 94

11.3.3.4 retract.c 97

11.3.3.5 safety.c 98

11.3.3.6 off.c 98

12 Appendix 2: Other Documents **99**

12.1 Mission Logos 99

12.2 Team Photos 100

12.2.1 All Team Photo 100

12.2.2 Software Team Photo 100

12.3 CAD Models 100

12.3.1 Front View 101

12.3.2 Back View 101

12.3.3 Right View 102

12.3.4 ISO 1 View 102

12.3.5 ISO 2 View 103

12.3.6 ISO 3 View . . . 104
12.3.7 ISO 4 View . . . 104
12.3.8 Left View . . . 105
12.3.9 Left Fully Deployed View . . . 105
12.3.10 Top View . . . 106
12.3.11 Top Deployed View . . . 107
12.3.12 Top Deployed Angle View . . . 108
12.3.13 Top Fully Deployed View . . . 109
12.4 Electrical Schematics . . . 109
12.4.1 Voltage Regulator . . . 109
12.4.2 Motor Driver . . . 110
12.4.3 AVR . . . 110
12.5 Power Budget . . . 111
12.6 Mass Budget . . . 112
12.7 Pin Assignments . . . 112
12.7.1 Power Pin Assignment . . . 112
12.7.2 Telemetry Pin Assignment . . . 113
12.8 Timer Events . . . 113
12.9 Payload Placement . . . 114
12.10 Launch Compliance . . . 114
12.11 Budget . . . 115

13 Glossary **115**

1 Introduction

The Hephaestus project is a 2016-2017 Senior Design project involving seniors from the Computer Science, Electrical Engineering, and Mechanical Engineering disciplines. The project is a proof-of-concept for construction in a space-like environment. The concept that we have chosen to pursue will be described over the course of this document but is, at its core, a high degree-of-freedom robotic arm mounted on a rocketry payload. At the next level of detail, the Hephaestus payload is an autonomously controlled 4 degree-of-freedom steel arm capable of precise, collisionless operations in microgravity and vacuum conditions.

This concept will be proven in August, following the 2017 RockSat-X launch from Wallops Flight Facility (WFF) in West Virginia. RockSat-X is an educational ride-sharing program for student exploration of Low Earth Orbit. The RockSat-X program is run by the Colorado Space Grant Consortium and provides students with a standard rocketry platform for space experiments. This will be Oregon's first participation in the program, and OSU's first space mission. This project is also supported by the OSU American Institute of Aeronautics and Astronautics (AIAA) club.

The success of this project will be determined following the launch in August. Due to gravity, we are unable to test the function of the arm on earth. The weight of the motors prohibits the arm from moving, even using a plastic version of the arm segments to reduce weight. Success of the mission will be measured in several ways. The most direct way that we will see our success is by recording a video of the payload's operation on orbit using a GoPro camera. We will also receive telemetry data during the flight about the experiment's state and status of the on-board touch sensors. Finally, we will be storing data throughout the experiment to the on-board computer for evaluation after the rocket's recovery.

The success of this mission will show the success of the Hephaestus Senior Design groups, take a step towards proving the viability of construction on orbit using a small autonomous robotic arm, and show OSU's ability to put an experiment in space. The goal for this project is mission success and technical and professional development.

1.1 Document Overview

This document will cover all of the work on the Hephaestus payload that has been conducted by the software team over the course of the year, as well as some closely related materials generated by the Mechanical and Electrical Engineering majors. These included materials are necessary for understanding the full scope and nature of the project. This document includes our original design document, requirements document, and tech review document, as well has the changes that have been made since the creation of the initial documents. This document also contains documentation on the project itself and how to run the project, including the hardware required. This document contains all of the progress update blog posts that we have written over the course of the year. Lastly, this document contains useful pieces of code and other chart and diagrams.

2 Project Overview

2.1 Project Purpose

The OSU 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

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 and after the flight.

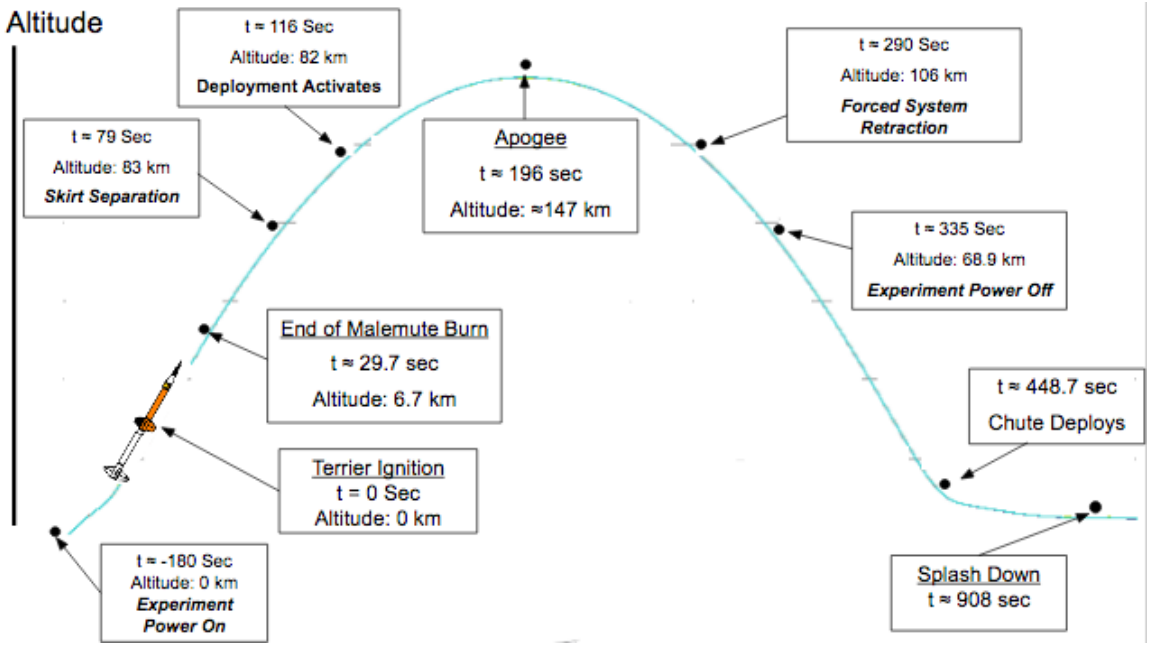
2.2.1 Minimum Mission Success Criteria

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

2.2.2 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.

2.3 Concept of Operations

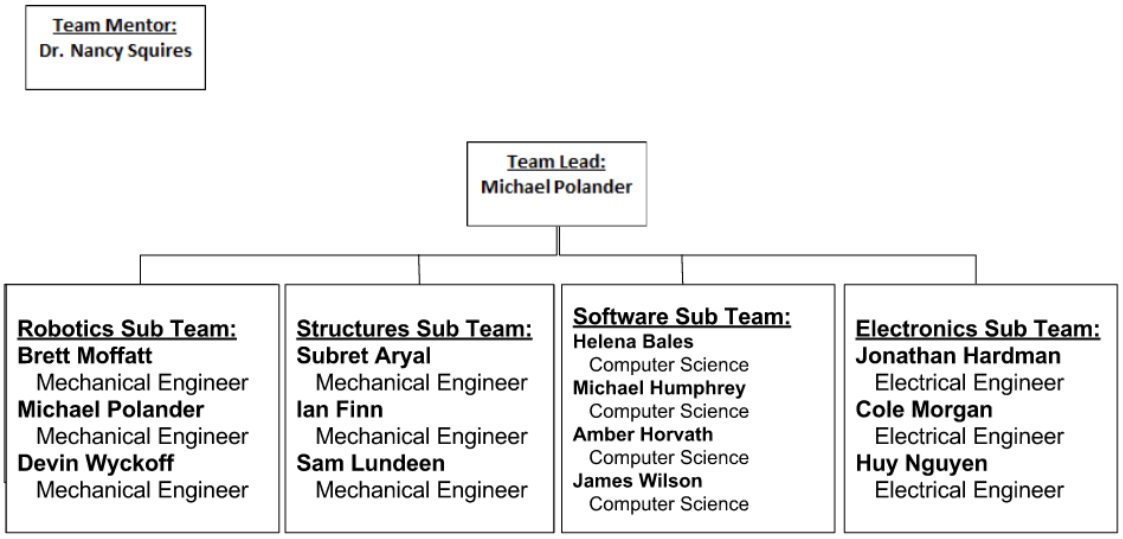


Event	Time On	Units	Dwell Time	Units	Event Description
GSE 1	T-180 Sec	(T-X) (sec)	520 Sec	(sec)	Voltage regulator and Microcontroller turn on.
GSE 2		(T-X) sec)		(sec)	
TE-R	T+116 Sec	(T+X) (sec)	5 Sec	(sec)	Activation line for experiment deployment.
TE-2	T+185 Sec	(T+X) (sec)	120 Sec	(sec)	CU: Start/Stop Video
TE-3	T+200 Sec	(T+X) (sec)	90 Sec	(sec)	CU: Open Door & Eject Boom
TE-1	T+290 Sec	(T+X) (sec)	5 Sec	(sec)	System retraction failsafe signal.

Concept of Operations for the 2017 RockSat-X Rocket Launch.

2.4 Programmatics

2.4.1 Organizational Chart



2.4.2 Sponsors

The Hephaestus project is made possible by generous donations of time and money from the following people and organizations:

- Dr. Nancy Squires, Techincal Advisor
- Oregon NASA Space Grant Consortium
- Colorado Space Grant Consortium
- AIAA

3 Requirements Document

3.1 Original Requirements Document

3.1.1 Introduction

3.1.1.1 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.

3.1.1.2 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.

3.1.1.3 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
- 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

3.1.1.4 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.

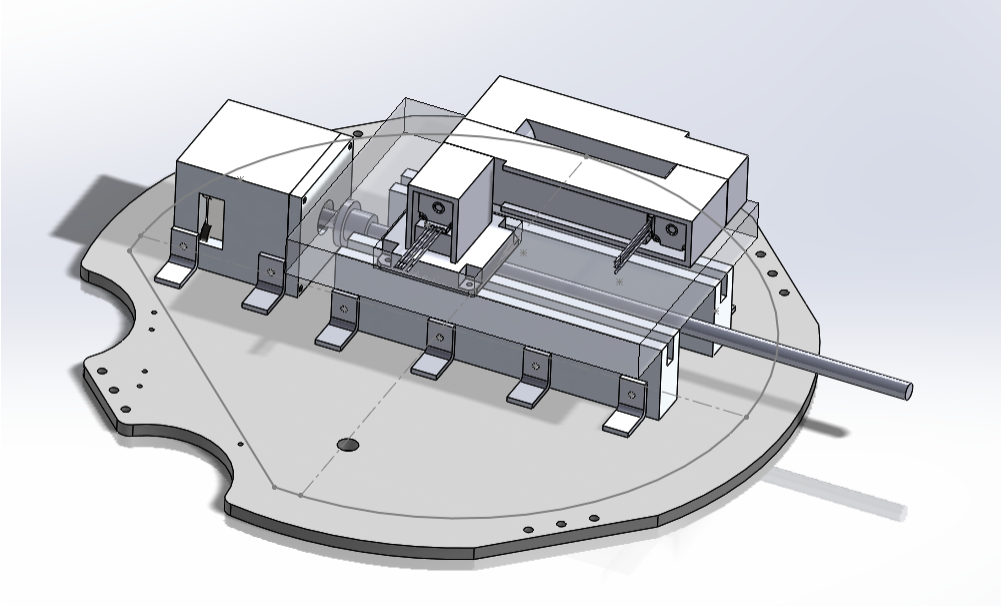


Figure 1: Model of Hephaestus Payload

3.1.1.5 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.

3.1.1.5.1 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.

3.1.1.5.2 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.

3.1.1.6 Requirements Apportioning

3.1.1.6.1 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.

3.1.1.6.2 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.

3.1.1.6.3 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.

3.1.2 Functional Requirements

3.1.2.1 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).

3.1.2.2 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.

3.1.2.3 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.

3.1.2.4 Modes

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

3.1.2.4.1 Launch

Priority 1: The software shall remain idle during launch.

3.1.2.4.2 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.

3.1.2.4.3 Science

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

3.1.2.4.4 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.

3.1.2.4.5 Observation

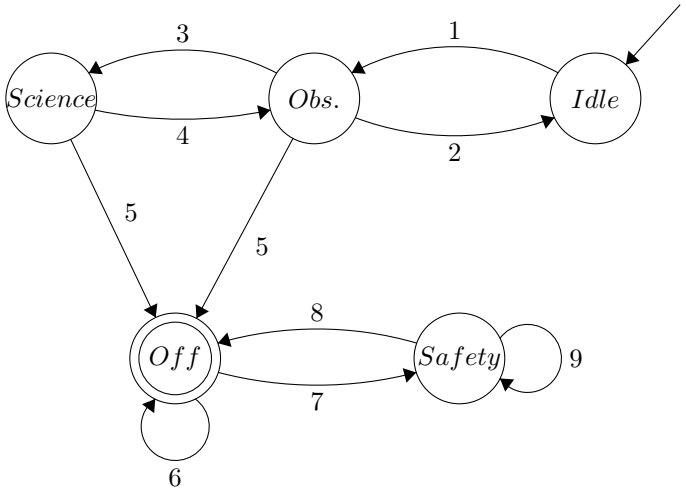
Priority 1: The software shall report all telemetry data (as defined in 2.5) to the ground station.

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

3.1.2.4.6 Power Off

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

3.1.2.4.7 State Diagram



State diagram for transition between operational modes.

3.1.2.5 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.

3.1.3 Non Functional Requirements

3.1.3.1 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.

3.1.3.2 Security

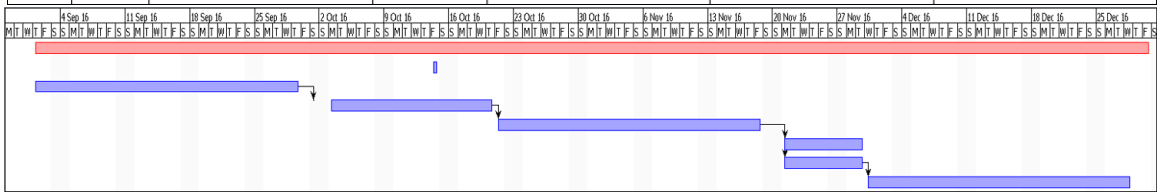
Priority 1: The system shall be secure. Since it is a closed system, the device will be programmed such that it cannot be accessed remotely and will only output sanitized data.

3.1.3.3 Telemetry

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

3.1.4 Gantt Chart

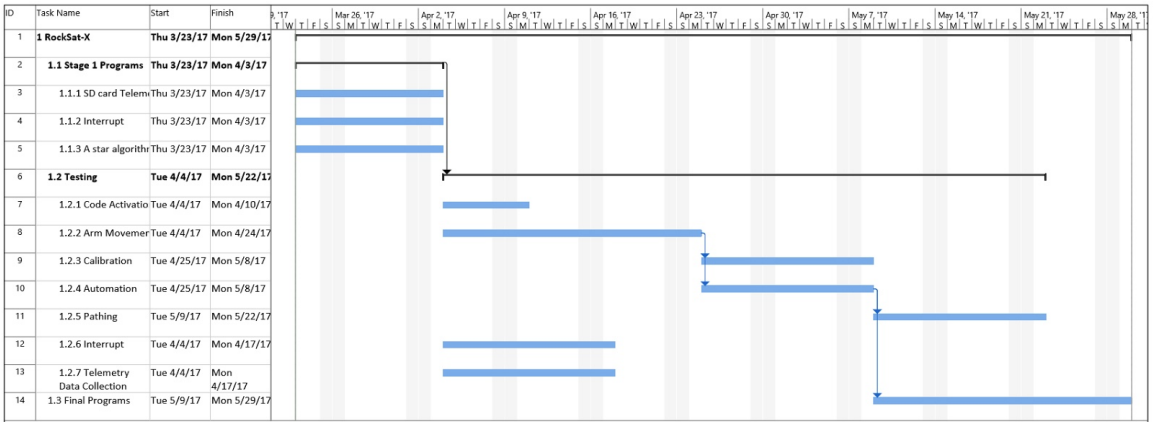
		Name	Duration	Start	Finish	Predecessors
1		Design	87 days	9/1/16 8:00 AM	12/30/16 5:00 PM	
2		CoDR	1 day	10/14/16 8:00 AM	10/14/16 5:00 PM	
3		Design Direction	21 days	9/1/16 8:00 AM	9/29/16 5:00 PM	
4		Concept Design	14 days	10/1/16 8:00 AM	10/20/16 5:00 PM	3
5		Preliminary Design	21 days	10/21/16 8:00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



3.2 Changes Since Original Requirements Document

No changes made to original requirements document.

3.3 Final Gantt Chart



4 Design Document

4.1 Original Design Document

4.1.1 Introduction

4.1.1.1 Document Overview

4.1.1.1.1 Helena Bales

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

4.1.1.1.2 Amber Horvath

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

4.1.1.1.3 Michael Humphrey

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

4.1.2 Technologies

4.1.2.1 Target Generation

4.1.2.1.1 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.

4.1.2.1.2 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.

4.1.2.2 Arm Movement

4.1.2.2.1 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.

4.1.2.2.2 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.

4.1.2.3 Arm Position Tracking

4.1.2.3.1 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.

4.1.2.3.2 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.

4.1.2.4 Emergency Payload Expulsion

Author: Amber Horvath

4.1.2.4.1 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.

4.1.2.4.2 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.

4.1.2.5 Program Modes of Operation

Author: Amber Horvath

4.1.2.5.1 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.

4.1.2.5.2 Solution Design

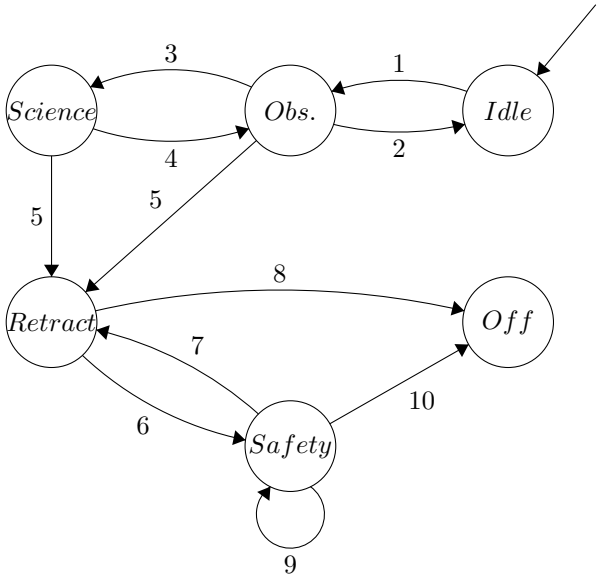


Diagram of software states of operation and transition between states [2].

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.

4.1.2.6 Target Success Sensors

Author: Amber Horvath

4.1.2.6.1 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.

4.1.2.6.2 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.

4.1.2.7 Telemetry

Author: Michael Humphrey

4.1.2.7.1 Requirement Overview

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

4.1.2.7.2 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 4.1.2.6, 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.

4.1.2.8 Video Handling

Author: Michael Humphrey

4.1.2.8.1 Requirement Overview

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

4.1.2.8.2 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.

4.1.2.9 Data Visualization and Processing

Author: Michael Humphrey

4.1.2.9.1 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.

4.1.2.9.2 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.

4.1.3 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

4.2 Changes Since Original Design Document

4.2.1 Target Generation

The target generation design has had some slight changes. The test points were going to be randomly generated, but due to our need to place sensors on the payload and the limited locations that can be wired with sensors, we had to hand-select the test points. Further, the points are not stored in the radial form that I indicated and are instead represented as configuration space points, with an angle

of roll, an angle of yaw, and two angles of pitch. These changes were made due to limits imposed by the configuration space and the hardware.

4.2.2 Arm Movement

There have been fairly limited changes to the arm movement following the development of the configuration space, however we were able to switch from using A* to using Dijkstra's algorithm thanks to the evenly spaced nodes in the configuration space, accomplished in parser.cpp.

4.2.3 Arm Position Tracking

Arm position tracking is the section of this project that has changed the most since the initial design document. The biggest change was the addition of the 4D configuration space as the storage method for information regarding the physical shape of the payload. The configuration space is now what is used to track the position of the arm and the path which it follows. These coordinates have four values, one of yaw, one of roll, and two of pitch, representing the theta values of each one of the four arm motors on Hephaestus.

4.2.4 Emergency Payload Expulsion

The design of the emergency payload expulsion changed multiple times due to changing requirements and understanding of how the payload and arm itself would fit into the rocket's limited space.

Originally, the plan was to disattach the arm and eject it into space upon encountering an issue that prevented us from being able to properly retract the arm into the body of the payload, such as the arm getting stuck on some object. However, upon request from our contacts at the RockSat-X facility in Colorado, we changed the design such that, in the event of the arm getting stuck or the experiment timing out, we power off all motors except the motor attached to the lower deck plate, and just pull the arm in. This implementation works with the inclusion of an interrupt vector that is triggered by the event timer going high, signalling the end of our time in apogee or the previous modes failing to return a value indicating that the arm is in its callibrated, home position.

4.2.5 Program Modes of Operation

The design for the program modes of operation hardly changed throughout the course of the project. The only difference from our initial design and final design was the inclusion of a state representing the retract phase. The reason for adding the retract phase is that we originally imagined retraction would only happen during the beginning and end of the flight and thus would be included within the Observation and Off phases, respectively. However, since we changed our emergency payload expulsion to essentially a retraction mechanism, we wanted to include this relationship within our design of the program structure.

4.2.6 Target Success Sensors

The design of the target success sensors has changed multiple times over the course of the year, mostly due to time constraints and evolving understanding of the problem and how to move a robotic arm with 4 degrees of motion.

The original design included the automatic generation of randomly selected points within the confines of the payload, and the on-board computer performing an automatic check to determine after motor movements whether it's made it to that point. While this could be future work for later students and would allow for a more flexible design, our team did not have the capacity to perform this dynamic moving aboard the payload. Instead, early on within the project, we changed the design to its current state where we have pre-placed sensors onboard the payload. Upon a sensor being depressed, a signal is sent and a code is sent over the telemetry lines and a message is written to EEPROM.

4.2.7 Telemetry

The design of the telemetry interface has changed multiple times over the course of the year, mainly due to the poor understanding of how telemetry systems work on the rocket.

The design was originally going to transmit the result of a series of tests that the payload would perform. However, when the arm pathing design changed from dynamically generating target points to using a set of predefined target points, we also changed the way the result of these operations would be reported via telemetry. We ultimately decided on a set of 4 digit codes to transmit through telemetry, since we had four pins to transmit data through the rocket's telemetry system. Each code would represent a milestone in the program, like transitioning to a new phase or successfully touching touch sensor.

Additionally, we did not plan on storing a log file. However, later in the design process we decided to add an SD card to the payload to store data on. To take advantage of the huge amount of data storage available to us, we decided to store a detailed log of all operations to better assist in debugging after the flight. However, after significant implementation difficulties, we were forced to abandon the SD card. Instead of aborting the log file, we changed the storage target of the logging API from the SD card to an on-board EEPROM chip. This introduced a new constraint, as the EEPROM chip had only 4 kB of storage space, as opposed to the near limitless storage capacity of the SD card. We had to reduce the total amount of logging done in order to remain under the 4 kB limit. The logging format will remain the same.

4.2.8 Video Camera

When this document was first drafted, the video was originally designed to be handled by the on-board computer. However, as the electrical system was designed and finalized, it became easier to have the video completely handled in hardware. As a result, the on-board computer does not interact with the video camera beyond turning it on at the beginning of the experiment and turning it off at the end. The electrical system manages storing the data.

4.2.9 Data Visualization and Processing

The only change to the design of the data visualization and processing component is the parsing component for the log file. Instead of opening an ASCII text file stored on the SD card, this component must parse a raw binary file copied from the EEPROM chip.

5 Technical Review Document

5.1 Original Technical Review Document

5.1.1 Introduction

5.1.1.1 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.

5.1.1.2 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.

5.1.1.2.1 Helena Bales

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

5.1.1.2.2 Amber Horvath

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

5.1.1.2.3 Michael Humphrey

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

5.1.2 Technologies

5.1.2.1 Target Generation

5.1.2.1.1 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.

5.1.2.1.2 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.

5.1.2.2 Arm Movement

5.1.2.2.1 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.

5.1.2.2.2 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.

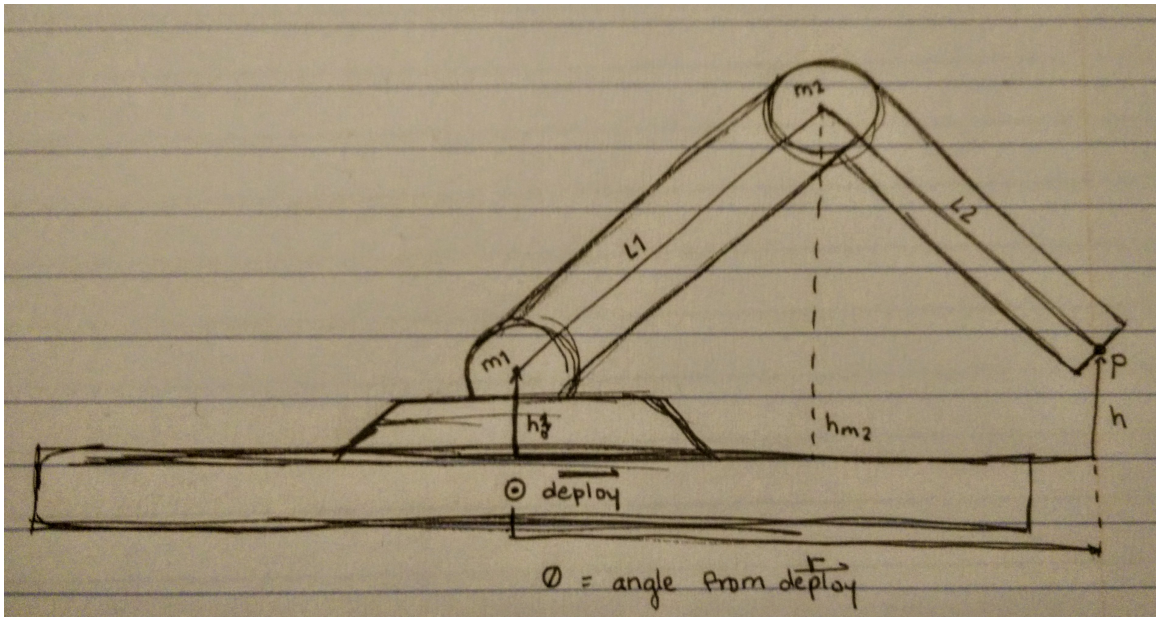


Figure 1 – Arm Movement Design

5.1.2.3 Arm Position Tracking

5.1.2.3.1 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.

5.1.2.3.2 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.

5.1.2.4 Emergency Payload Expulsion

5.1.2.4.1 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

5.1.2.4.2 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.

5.1.2.5 Program Modes of Operation

5.1.2.5.1 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.

5.1.2.5.2 Proposed Solutions

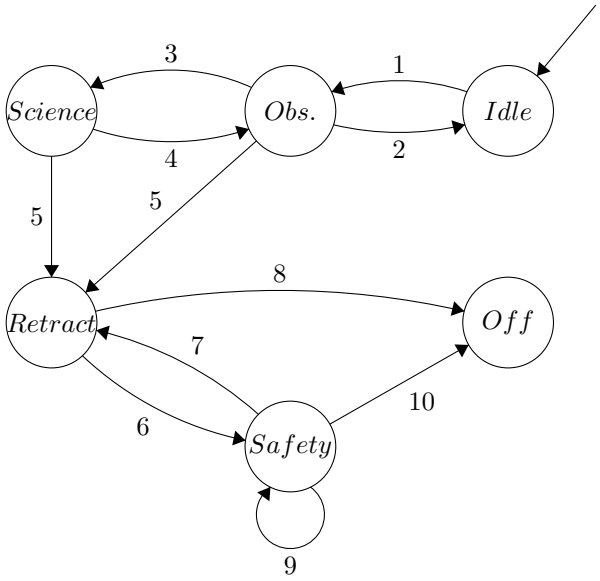


Diagram of software states of operation and transition between states [2].

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.

5.1.2.6 Target Success Sensors

5.1.2.6.1 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.

5.1.2.6.2 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.

5.1.2.7 Telemetry

5.1.2.7.1 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.

5.1.2.7.2 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.

5.1.2.8 Video Handling

5.1.2.8.1 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.

5.1.2.8.2 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.

5.1.2.9 Data Visualization and Processing

5.1.2.9.1 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.

5.1.2.9.2 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.

5.1.3 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.

5.2 Changes Since Original Technical Review Document

We did not make any changes to our tech review document.

6 Weekly Blog Posts

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

Progress
This previous week, our team made great strides by writing up our problem statement, which served useful in nailing down what we hope to accomplish this coming term, and how we can actualize these goals. We also took a group photo with our full team (comprised of us three computer scientists, six mechanical engineers, and three electrical engineers). This photo was required by Dr. Nancy Squires, our advisor. We also presented our plans in the form of a powerpoint to our Colorado collaborators over a group call.

Plans
For the next week, we are going to meet with our team to discuss future prospects for our project.

Since we've chosen a design, our next step is to begin prototyping implementation designs and begin research on the telemetry aspect of the project. We are also going to use our RockSat-X project as a stepping stone to introducing a designated lab for satellite creation to Oregon State University. In order to do that, however, we will need to work on finding space on campus for such a lab and lobbying for its creation. While this might seem outside of the scope of the project, it would be greatly beneficial for future students who have an interest in space exploration and development.

Problems
So far, we have not encountered any barriers. The only issue is coordinating schedules and planning with 11 other students. Finding times where everyone is available is difficult, but since we have an effective group chat and a Google Calendar with everyone's individual schedules filled in, we've been doing our best to mitigate this issue.

6.1.1.3 Michael Humphrey

Progress
This past week the Hephaestus project team accomplished several important milestones. We completed our first presentation to the RockSat-X organizers and took a group picture to start raising funding. We are also starting to narrow down our design for the final payload.

Plans
Because the mechanical and electrical design of the project is not yet finalized, the software team has not yet had any important responsibilities. The electrical team is forbidden from using a device like a Raspberry Pi or an Arduino, so they have decided to use an AVR microcontroller. Amber and I have not used one of these devices, although Helena has. Amber and I will need to start doing research on programming for these devices. We will be using C to program the microcontroller. We won't be able to write any code until the electrical design (i.e. inputs and outputs) are finalized, but we can start creating a software design of how we want the software to work.

Problems
No problems have been encountered yet.

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

Progress
This week, our team worked to figure out more logistical details. We assembled two new sub-teams, one is the budget planning committee (to which I represent the CS group), and a group to prepare a slideshow presentation for an event happening next week. As for progress on the tool, we just recruited a new post-grad to help who has a degree in mechanical engineering but is coming back to school for a computer science degree. He is currently enrolled in CS 161 and thus has experience in C++ and nothing else.

Plans
Our next initiative is to meet up with the electrical engineers to figure out constraints between hardware and software and read up on some of the documentation on what we can load into our payload as far as the robotics side goes.

Problems
Our barriers have been limited to task and time managing. Between the amount of students, the amount of deadlines, and the differing levels of expertise in each others domains, it can be hard to keep communication clear and comprehensive. Meeting times have also been difficult to coordinate due to the different schedules.

6.1.2.3 Michael Humphrey

Progress
Similar to week 3's blog post, this past week the Hephaestus Software Team did not have any major responsibilities. We attended the Hephaestus team meetings where the mechanical and electrical designs are still being worked out. We are going to have more communication with the Electrical Engineering team to determine the computing platform and computation restrictions. We also began working out budget numbers.

Plans
This next week we will be creating several presentations. I will be partly responsible for a 6 minute 40 second presentation to compete for a $1,000 cash prize. Other fundraising efforts are also in progress. We will also be meeting with the Colorado Space Grant committee for our next presentation for them. We will also need to start working on revising our Problem Statement and start drafting our Requirements document and any other documentation we need.

Problems
Currently, the software team is blocked by the electrical team. Until they finalize a design, we cannot start coding. We will be in communication with them, however, to determine what considerations they need to take for the design.

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

Progress
This week we refined our problem statement and created a rough draft of the Requirements document. I was unavailable to attend our weekly meeting, but I did attend the first budget committee meeting. The meeting was held over the app Discord, and we came up with a budget for the rest of our project's lifespan.

Plans
Our next initiatve is to gather information from the electronics team to gather information for our requirements document. This is important, as hardware limitations will greatly impact our software design choices.

Problems
The problems we have encountered include illness and traveling. When we do not have teammates available, it can be difficult to be as productive due to reduced man power. Hopefully everyone comes back happy and healthy soon.

6.1.3.3 Michael Humphrey

Progress
Since our mechanical and electrical design is still in progress, we have made no progress in the past week toward writing any software. Only work done was finishing the problem statement assignment and drafting our requirements document.

Plans
For the next week we will be getting datasheets and other information from the electrical team to aid in drafting our requirements documents. Any limitations of the hardware will be taken into consideration for the software requirements. Those materials should be made available by the electrical team by early next week.

Problems
Problems encountered this week were mostly personnel issues. Some of our team has been on vacation and one member is now sick and unable to make it on campus at all. I feel myself coming

down with my second illness this term, which will make it even more difficult to get the required signatures we need.

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

Progress
This week, our team worked on finalizing our Requirements Document, along with doing our first Product Design Review meeting with our collaborators in Colorado. We also moved forward on some key design choices, such as choosing to program in C on an ATMega128 microcontroller. We also worked on communication between our teams by bringing in Discord (the app) as a new medium of communication.

Plans
Next week, we hope to work on the framework for the software. With a key design choice in place, we can make progress towards actual implementation. This is an exciting turning point in our project!

Problems
Problems included communication outside of our sub-team. It can be hard to communicate within the larger group.

6.1.4.3 Michael Humphrey

Progress
This week was spent finalizing our software requirements for the project. We did extensive research

into the details of the mechanical and electrical design of our payload and drew up documents with specifics such as coordinate systems and payload layout. We now have a basis for creating our software.

Plans
For the next week, I believe we will be able to start writing the framework for the payload. We probably won't be able to start programming the actual function of the payload until it is built, but we can create the structure of how our software will be laid out.

Problems
Some problems were encountered this week with communication outside of our sub-team, but those have been resolved and shouldn't occur again in the future.

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

Progress
This week, our team worked on finalizing our Tech Review and continued collaboration with the larger RockSat-X team. Together, we're working to reach conclusions on how the design will be so we can hopefully dive into implementation within the next couple weeks. We've settled upon most choices, but it is difficult to reach conclusions with the rest of the team about where we stand and how everything will fit together. We did divide work between the three of us for the software components that are necessary to the overall design of the project.

Plans
Next week I will be out of town for most of the week, so I am leaving most decision making to the rest of my team to figure out as far as moving forwards.

Problems
Our main issues include traveling as that complicates our established workflow and coordinating all of our deadlines, both for this class and other classes.

6.1.5.3 Michael Humphrey

Progress
Last week we developed the requirements of our system a lot. We explored technologies that we want to use and confirmed many details with the robotics and electronics team about the requirements of the payload. For me, last week was spent primarily going to meetings, relaying information to teammates, and doing research into potential technologies we can use.

Plans
Next week, we will hopefully begin implementation of the software. I need to set up a meeting with the electrical team. I can't remember what they want to talk about but we definitely meet as a team with them. Most likely all of the CS team won't be able to make it, and this is a challenge we will need to overcome.

Problems
Problems I encountered included finding adequate solutions for the telemetry technology. I thought it would be easy to find several solutions we could use, but it turns out that most of the solutions I found were not compatible with our system for one reason or another. Mostly because none of them actually dealt with the transmission of the data itself, but what it did with the telemetry after it was collected. Other reasons were that they were implemented in the wrong language.

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

Progress
This week, we finalized our Tech Review. Most of the week I was out of town, but we are setting up our relations with the electrical team for next term and figuring out meeting times for our larger group next term.

Plans
Next week we hope to meet with the electrical team, but this may be difficult to accomplish as it is Thanksgiving and a lot of people will be heading out of town to meet with family.

Problems
Issues encountered included figuring out specifics regarding the implementation efforts that will need to be made. Looking into potential solutions and having to deal with some of the ambiguities about the hardware solutions presented so far made figuring out the software solutions difficult. However, as more information is presented about hardware solutions, this issue should resolve itself.

6.1.6.3 Michael Humphrey

Progress
Last week I helped start our Design Document. We've created the structure for the document and pasted the relevant sections from our previous documents. I set up meetings with the Electrical team and started communication with them to nail down specific software communication requirements. They're going to create a sort of "firmware" for the payload, meaning they'll write the code that interacts directly with the hardware, and they'll expose an abstract interface for the Software team so we only need to call something like moveArm(x, y, z); to control the movement of the arm.

Plans
Next week we need to finalize the details of how we want to control the payload arm. This will probably mean writing an API that we want the Electrical team to implement. We also need to prepare for the CDR coming up in a couple of weeks. This means we need to fill out the slides the Software team is responsible for. There will probably be other work for this presentation that we

will tackle as it comes up.

Problems
No problems this week.

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

Progress
This week was a bit of an odd week due to the Thanksgiving holiday. However, despite the holidays, our team still met up to discuss planning how we will finish the term. There still lies some action items that must be completed such as resolving issues on our Tech Review document, preparing for the Critical Design Review, and doing our end of term project assigned for this class. We will be meeting on Tuesday of this next week to further our plans and see how much we have accomplished over the break.

Plans
Next week will include meeting to discuss where we stand as a group and what work needs to be completed by what date. As the end of the term approaches, everyone is more busy and juggling many assignment deadlines from other classes so touching bases to ensure that the work required for this class gets done.

Problems
The main issue this week was scheduling conflicts due to Thanksgiving and the aforementioned deadlines drawing close. Staying on top of work is sometimes harder than one would imagine.

6.1.7.3 Michael Humphrey

Progress
This week I didn't do much. The Software team has created an outline for our design document but I haven't added my parts in. I don't foresee it being too much work, as it's mostly already written

from the tech review. More details just need to be added. Due to it being Thanksgiving week, I have delayed working on classwork in favor of helping my family prepare for Thanksgiving.

Plans
Next week we need to finish our rough draft of the design document as well as write an outline for our presentation.

Problems
No problems were encountered this week.

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

Progress
This week our Project Design Document was due. Due to the business of Thanksgiving and other classes interfering, it was rather difficult getting this document done and over with. However, we managed to pull through. We also made progress on getting ready for next term with the larger senior design group. We came up with a meeting time for next term and made plans to meet up with the electrical engineers once more to work out specifics of our design choices.

Plans
Next week we will need to submit our progress report (which will include entries such as this one) and do our Critical Design presentation for the people in Colorado. Our team will most likely meet to work on the project over winter break, but specific times for that have not been chosen yet.

Problems
The main issues this week included miscommunications regarding work balance and meeting times. Some members were unable to attend previously scheduled meetings, resulting in work not being distributed properly and a lot of questions were fielded online as opposed to in person, which slowed our workflow. However, all work was eventually submitted on time, but it was a more difficult road to complete this then what we hope to achieve in the future.

6.1.8.3 Michael Humphrey

Progress
This week we made a lot of progress finalizing the design for the payload software. This was mostly a result of writing the design document. There was much communication with the electrical team.

Plans
Next week is finals week. We will be writing our progress report and recording our presentation.

Problems
One problem we are encountered is the slow response to questions that arise about the RockSat-X program. I have several questions about the format and delivery of telemetry data that won't be answered until mid- to late-next week. That information was not able to be included in the design document.

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

Progress
This week we attempted to gather ourselves after winter break and make a game plan for this coming term. We met as a group and worked on figuring out what needs to get done within the next 5 weeks and set small goals for each coming week. We are meeting with our EE collaborators next

week to figure out telemetry codes. I've downloaded Atmel studio so I can start writing code. We are divvying out work based on our design document from last term.

Plans
For the next week, I am hoping to begin work with the telemetry side of the project and help Michael.

Problems
So far, we have not encountered any barriers. The only issue is coordinating schedules, which was an issue we encountered last term as well. What mitigated that issue was clear and consistent communication across teams, which our newly established subteams should help with. I am serving as the representative for the Physical Integration team which is the team in charge of ensuring all the mechanical pieces get manufactured and put together properly.

6.2.1.3 Michael Humphrey

6.2.2 Week 2

6.2.2.1 Amber Horvath

Progress
This week was somewhat odd due to the short week the previous week because of snow throwing off our progress and the shortness of this week due to Martin Luther King Jr. day. Little progress was made this week. The only progress made was attempting to work on our team organizational skills and begin documenting our individual process through a scrum document that Michael set up.

Plans
There was also a change in project delegation as the EE's decided that they want a local storage for the microcontroller in the form of an SD card. I will begin looking into lightweight, Atmel-compliant storage systems that interface with an SD card.

Problems
Our barriers have been limited to lack of time between all of us. Also, due to having irregular meetings, we are having issues knowing what each other are doing and how to keep track of differing obligations and deadlines. We are attempting to overcome this with the scrum document which should allow for less in-person meetings and more communication about what we are all doing on our own time. We also are waiting on the EE's for their driver code they are writing for us. They have been tasked with writing the C code that will interface with the motors.

6.2.3 Week 3

6.2.3.1 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.

6.2.3.2 Amber Horvath

Progress
This week we got the code from the EE's for interfacing with the microcontroller. I also did research on SD card interfaces with microcontrollers that we are now using to write data from the microcontroller to an SD card. This code will be up on Github once we translate it from the Atmega 16 pins to the Atmega 64 (which we are using). I worked with the EE's to accomplish this translation as they are more familiar with the Atmega 64 archictecture than I am.

Plans
Our next step is to make the code compliant with the Atmega64 archictecture. I will also begin work on writing code that utilizes this library so that we can write telemetry codes, sensor touch data, and other log data to the card for post-mortem analysis.

Problems
The problems we have encountered include illness and traveling. When we do not have teammates available, it can be difficult to be as productive due to reduced man power. These random holidays also slowed our initial progress, so it can feel like we are trying to play catch up. We are also nearing deadlines for our STR meeting with the Colorado RockSat-X group, so I am beginning to feel the pressure of that as I do not feel our team has accomplished as much as I would like us to.

6.2.3.3 Michael Humphrey

This is a cumulative update for the first three weeks of Winter term.

Progress
The progress we've made up to this point includes installing Atmel Studio, setting up a solution for our class, and setting up several skeleton files. I've created a file for the first phase of the program; the idle phase. It is mostly filled out except for the code to interface with the pin. I've also set up a file in our Google Drive detailing rules and instructions for laying out our project and how we will develop. We also had a meeting with the EEs about any telemetry data they need transmitted. They reported that they have no special need for transmitting telemetry via the microcontroller (all the data they need will be hard-wired to the telemetry interface), so I can freely design and implement the telemetry code without any outside interference.

Plans
My plan for the next week is to become up to speed with the driver and interface that the EE team has created for us, so that I can begin full-time development on the project. I plan to finish designing and start implementing the telemetry code, as well as finalize the idle phase of the code.

Problems
Problems I encountered these first few weeks include waiting for the EEs to develop the motor drivers and telemetry interface. Other problems include team members not showing up to cross-functional team meetings, so we couldn't get information on how to use the drivers and interface. We had to schedule a special meeting much later with the EEs so we can start using the code they wrote.

6.2.4 Week 4

6.2.4.1 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.

6.2.4.2 Amber Horvath

Progress
This week, I was able to get all the code up on Github for the SD card interface with the microcontrollers. Me and the EE's (namely Jonathan Hardman) are working on implementing the remaining functions and pre-processor directives for our SD card interface file. We should have that done by next Tuesday. Jonathan also updated the drivers and uploaded them on Google Drive, so I will import those over to Github. Now that these drivers are almost completely ready, I am looking forward to working on more implementation that actually utilizes the libraries and drivers that we have gotten. This step, however, hinges on having motors and a working prototype ready so we can determine whether our arm movements are correct. This will not be ready soon as our funding has been delayed leaving us with no money for purchasing these motors.

Plans
Next steps include nailing down with Michael what information we are sending over telemetry and what of that information should be sent to the SD Card. Next steps also include getting the updated code from Jonathan and getting the drivers up on Github and fully implemented. The CS 462 class is also requiring that we update some of our previous documents and get all our deliverables up onto One Note, so we will do that as well. Lastly, we have a big meeting with our sponsors in Colorado next week, so we have some slideshow slides that we need to complete.

Problems
We are in a stage where we need to start divvying out tasks and better communicating as a team. We also have dependencies that are reliant upon other teams that we have no way of controlling, such as the lack of motors.

6.2.4.3 Michael Humphrey

Progress
This week I hit a major personal milestone for this project. I finally was able to compile the (mostly useless) code we've all written so far this term and run it on the actual microcontroller! It compiled successfully for me, and although it didn't do much it was a relief to see the fruits of our labor to

work. The code did not compile on Amber's computer, but that was probably because she is using a mac and mac's are not directly supported by the avr compiler. I helped her troubleshoot a little bit, but she eventually figured it out on her own. I also added to our Description of Operations to detail more about the telemetry codes. Other than that, this week was spent waiting for drivers from the EE team for motors and pin inputs.

Plans
My plan for the next week is to create some constants in the telemetry header file that will encode telemetry codes as well as constants for telemetry operation (such as the length of time codes will be broadcasted).

Problems
No problems encountered.

6.2.5 Week 5

6.2.5.1 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.

6.2.5.2 Amber Horvath

Progress
This week I worked on furthering the progress on our SD card implementation. I updated the Makefile such that the library now compiles with the rest of the program. Jonathan also updated the code such that it uses the correct pins and is designed with our Atmega64 in mind. Since the FatFS library is meant to be generalizable across different microcontrollers, there are some functions that are left as user-designed so that they can be written for specific microcontrollers. We completed those parts of the code, so the library should be fully functional now.

Plans
Our next work will be to test the SD card library. We are also prepping for our STR presentation with our contacts in Colorado, so we have to complete an extensive PowerPoint skeleton that they provide us. We will also be meeting with Dr. Smart (a professor in the Mechanical Engineering department) to discuss path-finding for our mechanical arm.

Problems
Our main blockers are lack of communication between team members. Despite putting a scrum document into affect and requiring each individual team member to add to the document every Monday and Thursday, some members of the team have been failing to meet this requirement, so it is difficult to know what is going on between other subteams and what our individual team members are doing on their own time. We also have regularly scheduled meetings, but attendance has been rather lax on those with team members either forgetting or making excuses right before the meeting time. Despite us stressing the importance of attending these meetings and communicating relevant information, our team still struggles with this, so some measures must be taken. However, we are unsure at the time of writing this what those measures should be.

6.2.5.3 Michael Humphrey

Progress
This week I updated my three sections of the design document. Additionally, I reviewed the tech and requirements, but no changes were required to either of these documents. Additionally, I met with our TA to review feedback on these documents last time they were graded, but there was not much feedback. I communicated the feedback there was to the rest of the team.

Plans
For the next week, I hope to finish the idle phase by implementing the flags provided in the drivers. Additionally, I would like to complete writing definitions for codes in the telemetry header file.

Problems
No problems encountered.

6.2.6 Week 6

6.2.6.1 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.

6.2.6.2 Amber Horvath

Progress
This week was focused primarily on prepping for our big STR presentation. This is a presentation where we review our subsystem testing for the RockSat-X group in Colorado. The meeting was at 6 AM on Friday and required us to prepare a lot of slides for a slideshow. We also met with Dr. Smart to learn more about preparing our C-Space for pathing the arm's movement and determining what are and aren't legal movements for the arm. There are multiple ways to do this as we can either use inverse kinematics, manually moving the arm through different legal motions and recording the different degrees of rotation the motors are in, or using sensors on the motors to determine their rotation. We are hoping to do the last option.

Plans
Next week we shall work on the presentation for this class. Fortunately, we can lift a lot of the work we did last week so it shouldn't be too cumbersome. I will also be meeting with the EE's on Friday to work on testing the SD Card implementation and also continue work on the code for moving the arm.

Problems
Blockers this week were the fact that I was running a study at IBM so I was out of town most of the weekend and could not do work for this since I was busy preparing for that. This should not be a problem now that the study is concluded.

6.2.6.3 Michael Humphrey

Progress
The bulk of this week's work was spent editing the software slides for the STR presentation. In addition to this, I implemented the remaining portions of the idle phase code and sat in on a meeting with Dr. Smart about robotic arm movement and pathing. From that I have a much better idea on how the pathing algorithm will work.

Plans
My plan for next week will be slightly smaller in scope due to personal obligations. However, I will make sure my portions of the requirements and design documents are up to date and that we're on track to complete the assignment we received an extension on. I also plan on finally completing the telemetry header file, as well as signing up our team for the expo.

Problems
Nothing currently blocking.

6.2.7 Week 7

6.2.7.1 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.

6.2.7.2 Amber Horvath

Progress
This week, our team worked on finishing our presentation and design documents for the Midterm Review. We also resolved some inter-personal conflicts due to some previous issues that had been presented by teammates regarding showing up to meetings and making contributions clear to all. Both these things were useful for determining our team's success and ensuring we are on the right track moving forward as we finish off the term. I also worked with the Electrical Engineer's on a testing suite for the SD Card library we implemented. However, this has been rather difficult as it is difficult to figure out errors for a microcontroller when it does not display feedback regarding what part of the code went wrong. We were trying to mitigate this by having LEDs that would light up when certain parts of the code failed, but I was not able to borrow the LED component attached to PORTD for the weekend. I will most likely continue attempting to isolate the error or else meet and discuss with the EE's soon.

Plans
Plans for the upcoming week is to go to our mandatory class tomorrow where we will present our elevator pitches. I will also attempt to meet up with the EE's so we can be done with this SD Card business.

Problems
The code failing to work has been the biggest blocker, and it could be due to many different potential issues. One reason could be that the pins are misconfigured or that there's something wrong in my

code. The EE's were in charge of ensuring the drivers for the SD Card were correct and I was in charge of getting the testing code up and running so I'm not sure which end is failing.

6.2.7.3 Michael Humphrey

Progress
This week I have made absolutely no progress. I have been kept busy with grad school application and other personal issues. Progress will resume after March 1st.

Plans
Goal for next week is start writing telemetry header file.ash

Problems
No issues.

6.2.8 Week 8

6.2.8.1 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.

6.2.8.2 Amber Horvath

Progress
This week I've primarily been working on the SD card implementation. While testing the software earlier this week, I realized that something must be wrong on the EE's side of the implementation effort as our card wasn't even able to initialize properly. I met with Jonathan and we got that part working while also giving me a bit of code that I could use to test line-by-line the implementation of the library we're using to isolate errors and either fix or hand off depending how outside of my abilities they are. So, as I've stepped through the code, I've isolated and fixed multiple errors. However, this is a very slow task so despite me putting hours into this effort progress has been slow-going. This is a very large library with lots of lines of code that's functionality is kind of obscured

and with minimal ability to debug (just a single set of changing LED lights depending upon whether conditions pass or fail) this has been rather difficult.

Plans
I will continue working on getting this SD card implementation working but will change my time needs as necessary, especially once the arm is in a state where code is ready to be written for it. However, that is not the case in this current time.

Problems
Blockers are mainly that this is challenging and I am looking for outside support outside of Jonathan who says he won't really have time to help me until dead week. I've emailed Roger Traylor but he does not seem super familiar with this branch of work. I might also email Ben Lee.

6.2.8.3 Michael Humphrey

Progress
Not much progress this week. Can't develop without a prototype to test on. Can't write code if we can't run it. (I mean we could but that would be a colossally awful idea.)

Plans
Plan for next week is to design and begin implementation of motor test. This test (or tests) won't be used during the payload deployment on the rocket, but they will be used to demonstrate the payload actually works on earth, for various design requirements.

Problems
Only problems are that the Pathing team is hogging the only working prototype.

6.2.9 Week 9

6.2.9.1 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.

6.2.9.2 Amber Horvath

Progress
This week, I successfully resolved the issues preventing a function within the FatFS library working. The issue was due to an internal issue that caused the wrong, non-SD card specific function from being called. After debugging for a bit with Michael, we found this error and were able to get the SD Card mounted to the microcontroller and the file pointer to be opened and set to an internal location for a file to be created and written to. We also met with Roger Traylor who confirmed for us using an oscilloscope that the correct bytes were being spammed over SPI, showing that our code was doing what we expected it to do at the point we fixed during the execution. Furthermore, I met with my physical integration team to get the wires correctly set up for the payload and helped tap the metal plate our payload will be mounted on. This was exciting as I hardly ever get to do hands-on work with hardware, so I enjoyed getting to work with our mechanical engineers on this process.

Plans
Next week I will continue running through the program to debug issues relating to writing to the file and figure out what is going on with that. I will also be recording our final project presentation with my software team and preparing for our integrated system testing review for the RockSat-X program.

Problems
Blockers currently include the fact that this code isn't working as intended. While I was able to fix one problem, I almost immediately ran into another issue. This is a rather time-consuming process so, while it is a cool thing relating to the overall project, it may need to be dropped if my efforts are needed in a more critical area relating to the project.

6.2.9.3 Michael Humphrey

Progress
This week I worked with Jonathan Hardman from the electrical team to create a test program to run the arm. It operates each motor individually to demonstrate its full range of motion. The Robots

team said they needed the file as soon as possible, so I dropped everything to make it. They did not end up using it.

Plans
Next week I plan on working on the slides for our next big presentation, Integrated Subsystem Test Review. It is on Friday of finals week. I also plan to do the majority of the work for the final presentation and report for this class.

Problems
Nothing blocking.

6.2.10 Week 10

6.2.10.1 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.

6.2.10.2 Amber Horvath

Progress
Due to it being dead week, not much progress was made on my end. Our team recorded some segments of our presentation and I made a tiny bit of progress on the code but I am still a bit confused about the way it's behaving. I'm looking forward to spring break where I can work on it freely. Our team also has ISTR this week and perhaps a social Thursday evening.

Plans
Future work - continue working on code.

Problems
Blockers: none.

6.2.10.3 Michael Humphrey

Progress
This week I developed an application to read data from a file (ostensibly temperature data from a file on the on-board SD card) and display a graph of that data. This program is a prototype for our final application to fulfill the Data Visualization component of the requirements. The program is a Python script with the graph created using the matplotlib library. The program was made now to help the Electrical team fulfill a requirement for their senior design class.

In addition to the program described above to display the data file, I created a helper program (also in Python) to generate the data file with random data. The temperature sensor that will generate

the data only produces 10-bit values, so the random data generator must simulate that output. The program generates a random value between 0 and 1,023 (inclusive) and outputs it as a two byte sequence to a file specified in the arguments to the script. The program repeats this 100 times, outputting a null byte (0) as a separator between each value. This file can then be used as an input file for the data processing program.

Plans
Next week will be spent working on the senior design report and presentation.

Problems
Nothing blocking.

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

Progress
Hi all.

Sorry, but I had almost no progress this week due to being in Boise due to a family emergency. However, over spring break, I did look into the SD Card issues more and found out that the reason a function was having undefined behavior (returning success and failure seemingly simultaneously) I found that this function was actually being called in a previous library function call that I had failed to take into account during my stack trace. This was a breakthrough in that it proved this me and this function weren't losing our "minds" but also left me with the original problem I had of not knowing why the function was failing in the first place.

Plans
The next plan is to continue investigating why this is failing to work and move implementation, hopefully, more towards the arm as the ME's and EE's are getting that in a good place for us to test it.

Problems
Being out of town forced me to spend less time on this work than I would have liked, but now that I'm back in town I should be able to work more.

6.3.1.3 Michael Humphrey

Progress
No progress this week.

Plans
Plan for next week is to implement more of the telemetry interface.

Problems
No problems.

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

Progress
No notable progress was made week 2 of spring term. I was unfortunately mentally and academically recovering from my absence the week prior and didn't have a lot of time to spend on senior design work. I did meet up with my team for the first time since the previous term and we designated

tasks for the coming weeks including finishing code implementation for the arm and finishing up the library implementation for the SD card.

Plans
Upcoming plans include meeting with the ME Brett to figure out where we will place the touch sensors on the arm body and implement code to move the arm to these touch sensor points.

Problems
Blockers are limited to just the fact that the arm is too heavy to lift and so it will be hard to test. Testing arm movement would require us to manually check the rotation of the motors using a compass. This is not ideal.

6.3.2.3 Michael Humphrey

Progress
No progress this week.

Plans
Finish implementing telemetry.

Problems
No blockers.

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

Progress
This was a hard week, boys and girls. Mostly due to the fact I was very busy with my research job as we had a paper deadline on Friday and basically had to spend every "free" hour working towards that deadline. This left minimal time to work on senior design, but fortunately I was able to make some notable progress. I met up with Jonathan from the EE's to discuss the issues with the SD card as I was at a loss as to what to do next to determine why that function wasn't working. We

reworked the SPI function call but that unfortunately didn't help anything. Our next step is to try reworking our approach and seeing if we can read from a file and perhaps find some commonality between reading and writing where the software breaks to pin-point an exact issue. Our other option is dropping this stretch goal (since that's essentially what it is) and focusing our efforts on writing stable data storage to EEPROM. If we can't get this stuff working soon enough that's probably what we'll have to do. Meanwhile, we also need to start focusing our energy on programming the arm to move. That's a big one.

Plans
Next steps are to find out where EXACTLY this code is breaking. And also to implement the retract and move out parts for the deck plate holding the mechanical arm as that's part of my requirements I was assigned ("emergency retraction").

Problems
Problems are that this SD card just won't cooperate and I'm so tired of it.

6.3.3.3 Michael Humphrey

Progress
This week, in light of the SD card still being unfinished, I implemented logging to the eeprom on the microcontroller. The eeprom only has 4 kilobytes of available memory, so it is only a mediocre replacement. We no longer have time to fix the SD card library we are using, so we must pursue other means of telemetry. Telemetry via the provided telemetry pins is fully implemented. Telemetry via SD card is partially implemented, but cannot be finished if the SD card library can't be fixed. Telemetry via eeprom is fully implemented. Until SD card is unblocked, the telemetry component is complete.

Plans
Last week, we had a team meeting. We have further split up the work, and my part is to meet with the instructors to review deadlines. I also plan on getting the poster completed and submitted. I will also be taking control of writing a readme for the github repo and copy test code into science mode file.

Problems
No problems.

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

Progress
This week we began working on some more implementation endeavors for the arm's movements. I implemented the touch sensor and deck plate extension and retraction (which were my designated tasks from the Requirements document) as interrupts that execute code upon receiving information from the hardware. In the case of the deck plate, an interrupt will be sent upon a timer event line (part of the RSX rocket) going to LOW and signaling the end of our deployment period for the payload. The interrupt will trigger and power the motor attached to the bottom of the deck plate, change it's direction from outward to inwards, and step the motors to pull the plate in. The code for the touch sensors are signaled by the touch sensors being depressed, and will write a code to the telemetry line signaling that we made contact. The implementation for the touch sensor is in science.c while the retract/extend code is in its own file (retract.c and retract.h). Since this is now complete, I will be assisting Helena in her work to finish the arm pathing and movement. Unrelated to the implementation effort, I joined Sam Lundeen (from the MEs) to visit Garmin regarding doing environmental testing for the arm. We met up with Steve Horvath (my dad!) and Greg Fisher to see whether it was viable to mount our arm onto their vibration table and test how the arm faired in situations similar to what will be experienced at launch time. We are going to meet with our team to update them that this is viable and to determine what we want equipped to the arm during this time. While it would be ideal to have everything attached, we've also had funding issues and don't want to risk losing any motors or other valuable pieces that we may not be able to replace if they get damaged during the testing.

Plans
Future plans are to continue implementation efforts with Helena and I also hope to clean up our repository a bit and add some more documentation regarding how to test our code so when we have the code freeze, the TAs/McGrath aren't confused as to what's going on.

Problems
Blockers are that I still have been unable to actually test this code due to the motors and arm being not set up currently. Hopefully I will be able to do this by our (extended, thanks McGrath!) May 15th deadline.

6.3.4.3 Michael Humphrey

Progress
This week I am making sure me and my team is ready for expo. I am following up with important people and assignments to make sure everything is complete and deadlines are being met. This includes but is not limited to making finals edits on our poster, getting out client's signature for the poster, submitting the poster, and making sure everyone has signed the necessary forms for expo.

Plans
I have completed both the video and telemetry components. I have only the data processing component left to implement as my portion of the project, but that is currently blocked until we receive further instruction from NASA. I have nothing left to implement for this project. I will spend the

next week getting ready for expo and working on my presentation.

Problems
I am waiting for after Integration testing with Wallops and we receive the format of telemetry data. Then the implementation of the data processing component can begin. I will not be directly working on this component, but rather guiding James on implementing the component.

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

Progress
This week I was extraordinarily busy with other classes so I was unable to do much on the project. I plan to get back to work after my midterm Tuesday of next week.

Plans
Next week, I plan on getting back into the code and performing minor bug fixes to get the code operational before our code freeze deadline of May 15th. We also have an upcoming meeting with our sponsors in Colorado and expo coming up so things are getting pretty exciting!

Problems
No stoppers are known as of now.

6.3.5.3 Michael Humphrey

Progress
This week I made some minor changes to the telemetry code.

Plans
Next week I plan to support Amber and Helena with development of the rest of the system.

Problems
No blockers.

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

Progress
This week, Michael and I worked on fixing some bugs with our implementation of the different phases. We also enhanced the readability of the code by adding some defines within a .h file and refactoring the code to use these more readable names as opposed to the previous iteration which just used numbers like 0 and 1. We believe this makes the code easier for outsiders to understand and, if changes are made in how the implementation works, we can just change the value of the defines as opposed to having to go through and find every number to change. The remaining work is just finishing the science mode, which is Helena's responsibility. We trust her to get this over the finish line.

Plans
Future progress is to get ready for Expo (May 19th!) and FSMR which should be Wednesday (May 17th) at 11. We also need to finish recording our midterm progress update. Next week will be fun/stressful!

Problems
Blockers are that the arm still isn't moving. We spoke with Huy (a representative from the EE team) who said that we don't know whether the arm will move even in space. This is concerning but ultimately out of our hands.

6.3.6.3 Michael Humphrey

Progress
This week I made some more minor tweaks to the telemetry code. I also sat down with Amber for 2 hours and worked on making the code in the repo to compiler. We also added a bunch of missing code in the off and retract phases. We also refactored a bunch of code to make it more readable.

Plans
Next week will be spent waiting for Helena to finish writing the code for the science phase.

Problems
Blocking on science phase being finished.

6.3.7 Week 7

6.3.7.1 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.

6.3.7.2 Amber Horvath

Progress
So, we did expo! And with respect to Vee's email, I'll be answering the questions given in that email. So here we go!

If you were to redo the project from Fall term, what would you tell yourself? I'd tell myself to work harder, earlier. While the timeline of our project was different from most teams and prevented us from starting implementation until later in the project life cycle, I believe there's more that could've been done to prevent the last-minute rushed feeling that I felt towards the end of the project. However, I also have enough experience working in large group projects to know that sometimes this is just the way it is. I'd also tell myself to work more on establishing good team working dynamics earlier on. Our team had some internal strife between members due to lack of clear communication of expectations and work balancing. If I could've said some magic words to Past-Amber to prevent this, I would have.

What's the biggest skill you've learned? In my technical skill-set, I'd say this project got me even more comfortable with embedded C programming. My previous experience with C was mostly higher level with a bit of pointers and system calls thrown in there. However, programming directly onto an Atmega128 allowed me to become more comfortable with working very close to the hardware. In terms of team dynamic, I'd want to work on advocating for myself. I have a lot of stuff on my plate but sometimes I left myself get more stuff placed on me due to not saying "no". This is something I can work on both inside and outside of school.

What skills do you see yourself using in the future? Since I plan on doing more user-facing software development in the future, I don't see myself using much of the technical skills I gained during this project. However, I do plan on using some of the team strategy skills we learned such as the retrospective table, effective inter-team communication, and the documentation creating skills we learned fall term.

What did you like about the project, and what did you not? I liked the cross-disciplinary aspect of the project a lot. I had never gotten to work with electrical engineers and mechanical engineers before, so getting a peek into their work life was really cool. I also have always been fascinated by space so knowing that my software will be sent into low-Earth orbit is really exciting. What I didn't like about the project is that the software component was at the end of the project's life-cycle so a lot of winter term felt I felt under-utilized since we were still waiting on the EE's and ME's to finish up their work, along with having no motors to test on due to funding issues. Overall, the timeline of the project proved to be one of the most frustrating aspects of the work.

What did you learn from your teammates? A lot! Especially from my ME and EE teammates since I knew almost nothing about either of those disciplines. But, specifically from Michael and Helena, I learned a lot about how to ask for help when I need it. Michael was awesome at always being there to lend me a hand if I was really running out of steam on the SD card implementation. Helena was great at sitting down and doing her work with little input or oversight from me or Michael. I could trust in her to do good work.

If you were the client for this project, would you be satisfied with the work done? Absolutely! And I'm not just saying that for my grade. Nancy Squires has been to all of our meetings with our sponsors in Colorado and has been consistently impressed and pleased with the work we've done. If I was Nancy Squires, I'd be happy. We've accomplished a lot since September and it shows.

If your project were to be continued next year, what do you think needs to be working on? If this project were to be continued, I believe we would want to expand the scope to allow for more fine grained movements of the arm and allow for dynamic loading of points for the arm to move to in space. Currently, we pre-process movements but in the future it would be cool to just say "hey arm, go to this point" while in flight and have it actually move there.

6.3.7.3 Michael Humphrey

Undoubtedly I have learned more from this class in the past year then I have learned from any other class or life experience. Working with a team of twelve engineers with a common goal is a task that is almost certainly destined for failure, but in the case of this project it was a spectacular experience. Each subteamtwo mechanical engineering, one electrical engineering, and one computer science teamwas a pleasure to work with. Typically collaborations with this many people are bogged down with red tape and communication mishaps, but we were able to all work together to get work done. Each team was responsible for a specific portion of the project, but made sure that all other teams were on board with any major decisions. Even though we as the CS team didn't care how the physical payload was designed, both ME teams made sure to include us on all decisions that were made. The EE team was especially helpful in making sure we had all the drivers and tools necessary to develop the code for the payload. I can only hope that my future coworkers will be as considerate and helpful as my team members were on this project.

Additionally, I loved working with physical hardware for this project. A lot of times in CS when we work with "hardware," it's really just interfacing with a well-defined API for some connected device. On the contrary, this project required us to write and debug embedded code for a microcontroller. Working with embedded code made me long for the days when I would get unexplained seg faults, and made me realize I was downright spoiled when working with high-level languages like Python. But the satisfaction of watching a device operate from the code I had birthed with my own fingers is unmatched. It's just not the same as seeing the terminal output of an algorithm or getting a message box to pop up from a GUI. Watching the arm operate according to our specification (and only occasionally acting homicidally) is one my favorite moments of this project.

This project has given me a lot of perspective that I had not gotten in any of my classes in my academic career. I have learned a lot from my two CS teammates about what it means to work in a team. I have learned a lot from my ME and EE teammates about what it means to collaborate from different disciplines on a project. I have learned what it takes to interface with outside vendors to get parts. I have learned what it takes to raise funds for a project. I have learned what it takes to meet deadlines and satisfy a client's requirements. I have learned how to write documentation for a project. All of these skills have prepared me to fulfill my potential as an employee in the workforce.

7 Final Poster

8 Project Documentation

8.1 Project Functionality

The following sections provide instruction for using the Hephaestus software.

8.1.1 Project Structure

The Hephaestus project's software is divided into a series of modules called *phases*. The software must wholly complete each phase before it moves on to the next phase. The one exception to this rule is if an error event occurs. In this case, the phase returns an error code, and immediately enters the *safety* phase. Each phase is responsible for one operation of the experiment. For example, the *observation* phase in responsible for turning on the camera and panning it around and the *safety* phase is responsible for resolving any error conditions present and safely retracting the arm for reentry.

8.1.2 Theory of Operation

When the system powers on, it immediately enters the *idle* phase. After completion of each phase, it moves on to the next logical phase. The system halts in the *off* phase. If an error occurs in any phase, it aborts the current phase and enters the *safety* phase.

Each phase is described as follows:

1. **Idle phase**

   This phase does nothing until a signal is received indicating that the rocket has reached apogee and the experiment may start.

2. **Observation phase**

   This phase turns on the camera and performs a 360 pan. After completing the pan, it points the camera toward the arm to record the rest of the experiment.

3. **Science phase**

   The Science phase is responsible for the motion of the arm in space. This phase powers on the motors and runs them along the path noted in the configuration space. The configuration space is pre-computed but is loaded into program memory for use on orbit.

4. **Retract phase**

   The retract phase powers off all the motors except for the motor powering the lower deck plate. Once all motors powering the arm and camera have been disabled, the deckplate motor pulls in the arm body. This phase is reached if there is an error after the Science phase or if the end of the apogee period is reached.

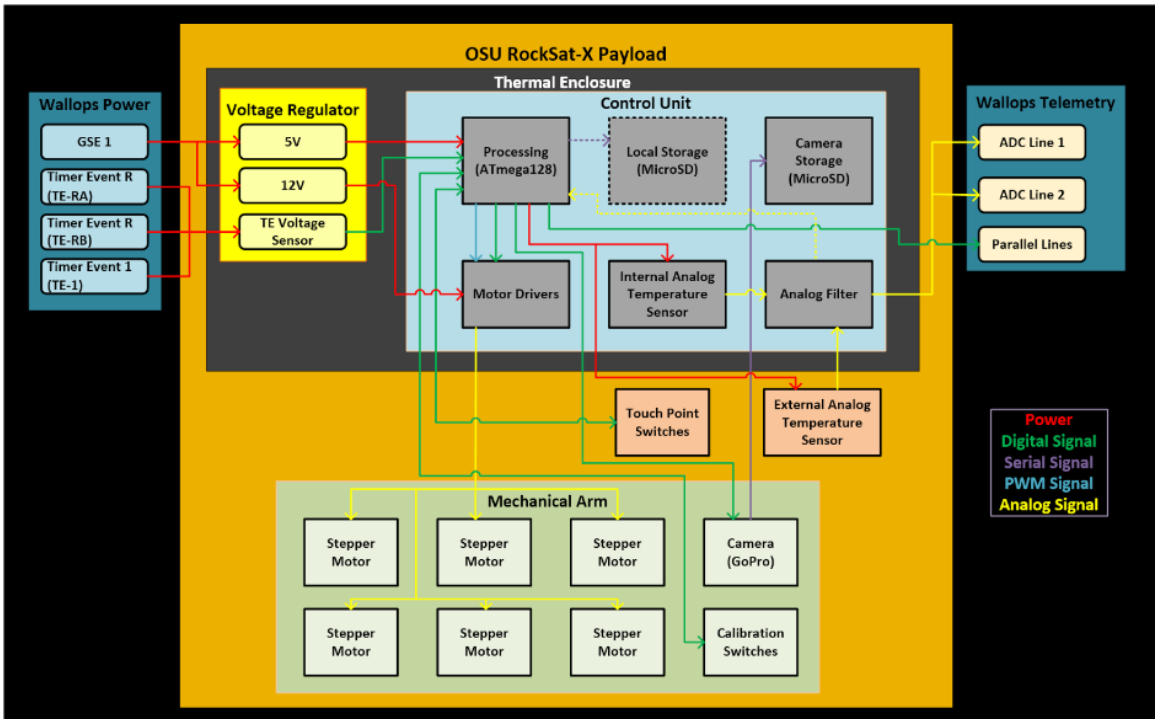
5. **Off phase**

   The off phase merely terminates the program execution.

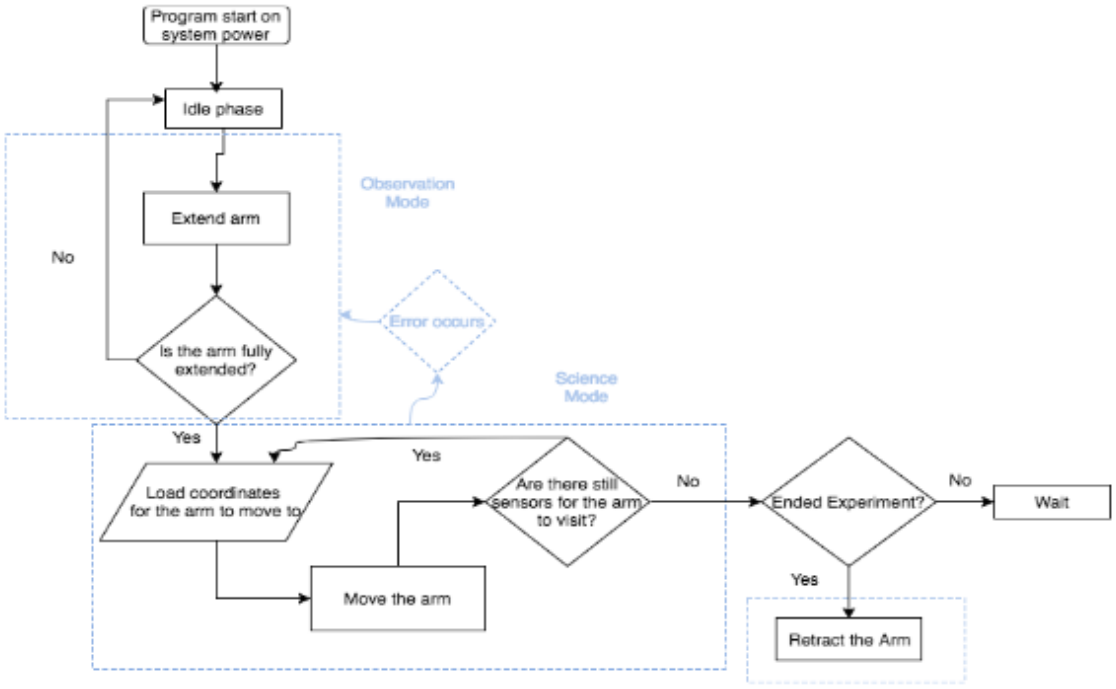
6. **Safety phase**

The safety phase is responsible for resolving and/or mitigating any errors that occur during any of the other phases. This phase is only reached if any of the other phases returns an error code. The current implementation of the safety phase simply attempts to retract arm and terminate, but this could be extended in future projects to attempt to resolve problems and resume the program.

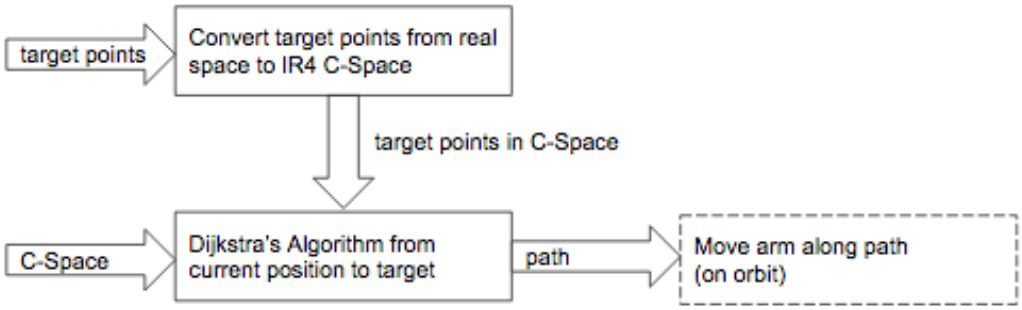
8.1.3 Block Diagram



8.1.4 Flow Diagram



8.1.4.1 Pathing and Automation Subsystem Flow Diagram



8.2 Hardware Requirements

Minimum hardware requirements to run the software for the Hephaestus payload include:

- ATmega128 microcontroller
- USB ASP programmer

8.3 Installation Instructions

To install the software on an ATmega128 chip, first ensure it is connected to the computer with a USB ASP programmer. Then, from the `code/Hephaestus/Hephaestus` directory, run `make all` followed by `make program`.

8.4 Running Instructions

The code will immediately begin running once the chip is powered on, and will continue to run until power is lost.

8.5 User Guides and Documentation

8.5.1 Program Overview

Hephaestus is a rocketry payload made by Oregon State University for the RockSat-X program. The faculty advisor for this project is Nancy Squires. The involvement of twelve of the project contributors is part of the Senior Design classes. The Senior Design groups include two Mechanical Engineering groups, an Electrical Engineering group, and the Computer Science group. This repository is for project deliverables for the CS Senior Design class as well as any design documents, code, testing, data, and documentation generated by the Computer Science team throughout the 2016/2017 Senior Design year.

Participation and flight in the RockSat-X program is subject to review and approval from the Colorado Space Grant Consortium. Pending this approval, the Hephaestus payload will fly in August of 2017.

8.5.2 Mission Overview

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.

8.5.3 Repository Contents

This section will describe the contents and structure of this repository.

8.5.3.1 Deliverables

The deliverables directory shall contain all deliverables for the CS Senior Design course. These deliverables are broken up into the term and assignment subdirectories. Assignments will be added through June 2017, at which point the course will conclude.

8.5.3.2 Design Documents

The Design Documents directory contains all design documents that we create. These shall be further organized by term and subject, pending need for organization. For the moment this shall be a place for any in progress, partial, or draft items.

8.5.3.3 Photos

The Photos directory shall include all photo documentation for the project, including any graphics for promotional use (such as logos or flyers).

8.5.3.4 Presentations

The Presentations directory shall include all presentations given by the CS team. This will include all presentations for RockSat-X program design reviews, presentations for promotional purposes, or for fundraising purposes.

8.5.3.5 Code

- **CSpace**

  Contains the code for mapping and parsing the configuration space.

- **Drivers**

  Contains the drivers given to us by the Electronics team.

- **Hephaestus**

  Contains all the code written by the Software team to operate the payload.

- **SDRead**

  Contains the code to parse telemetry files and display data in a GUI.

8.5.4 Instructions to build

8.5.4.1 Before you build

Before you can build the code for this project, make sure you have the avr-gcc compiler as well as an ATmega128 microchip and programmer. You can obtain a copy of the avr-gcc compiler from the WinAVR suite for Windows, or by installing the ‘avr-libc‘, ‘binutils-avr‘, ‘gcc-avr‘, and ‘avrdude‘ packages from your Linux OS’s package repository. In the case of OS X/macOS, ensure that you have version 5.11.1 of avrdude installed as the most recent version does not work with our implementation. The ATmega128 chip can be purchased from Microchip’s website. Also, be sure to have a programming board for your ATmega128 chip, which can be obtained from the OSU TekBot store.

8.5.4.2 Building

1. Navigate to the `code/Hephaestus/Hephaestus` directory.
2. Run `make all` to compile the code.
3. Run `make program` to program the microcontroller.

9 Learning New Technology

9.1 Helpful Resources

9.1.1 Web Sites

1. Atmel AVR Libc Reference Manual
2. ShareLatex Documentation
3. RockSat-X Program Homepage

9.1.2 Books and Print Materials

No books or print materials were used for this project.

9.1.3 Faculty and Personnel

1. Dr. Nancy Squires
2. Dr. Bill Smart

10 What We Learned

10.1 Helena Bales

10.1.1 Technical Information

I gained a significant amount of technical knowledge over the course of the year. Most of this is related to pathing and automation and configuration spaces since I did not have previous experience with that. I gained a significant amount of the technical information that I needed on configuration spaces from Dr. Smart and from books that I got from the Valley Library. I learned how to handle 4D arrays and how to map real space to a configuration space. I learned about Dijkstra's Algorithm versus A* and how to implement all this on a system with low computing resources. I also encountered the interesting problem of how to test all this on Earth under the influence of gravity since the arm's ability to move is dependent on it being in microgravity conditions.

10.1.2 Non-Technical Information

Over the course of the year, but especially in the last month with Expo, I developed some non-technical skills. Most of this has to do with navigating the logistics of the project such as fundraising, communicating with the OSU Foundation, and communicating the project to a wide and varied audience. Expo allowed me to learn how to talk about this project in a way that can appeal to a variety of people with different technical or non-technical backgrounds. Explaining the method by which we accomplished four degree of freedom motion in space was not the easiest task, but throughout the day I was able to refine my explanation to match the level of the audience. I was surprised to find that some of my most technical conversations were with the youngest people, as some high school girls talked through all of the code with me and asked about machine learning. I was very happy to talk to them, especially when Amber and I found out that one of them was going into an ASE internship, the same program that both of us went through in high school that got us into the fields that we are in today. The other main type of communication that I engaged in for this project was to discuss funding opportunities with the OSU Foundation. I was able to have a good conversation with a fundraiser for the foundation thanks to an introduction from my mother. While he was not able to help us secure grant money due to the time limitation, he was able to put me in contact with someone to set up an OSU Crowdfunding campaign. He was also willing to help us find grant money, and even found us a possible grant to apply for, even though we ended up being ineligible for it. Starting this relationship may be helpful to future RockSat-X projects at OSU.

10.1.3 Project Work Information

I developed some project work skills through this project. As I mention in my Team Work section, unique approaches to project work is characteristic of my personality type and is something that I struggle with as well. Now that I am aware of this characteristic, I can work to change my behavior so that it does not negatively impact my future work. My tendency when I'm working is to do things in bursts during which I work very hard without coming up for air and then avoiding the project for a little while before repeating the process. While this is great for creative and technical development, it can be stressful for the people depending on me, which is not acceptable. As such, I need to be careful in the future to communicate what I am doing when and to stop worrying that everyone is judging me harshly for only working at odd times. That is a silly anxiety to let get in the way of good communication. I learned a lot about how I do project work and I have some ideas on how to further improve, but I think we all ended the term on a strong note for getting things done.

10.1.4 Project Management Information

This year I mainly developed existing project management skills. I was able to quickly and painlessly get our github repository up and running thanks to past experience setting up version control systems for new organizations. I was also able to run the Pathing and Automation Cross-Functional team meetings with Mechanical and Electrical Engineers. This improved communication between the majors and gave me a support structure for completing the technical portion of the project that I am responsible for.

10.1.5 Team Work Information

I learned more about team work and how I work with others over the course of this year. We had a couple of disputes within the team. We were always eventually able to resolve these conflicts, but from this situation I learned that open communication and good listening can help to avoid a lot of these conflicts in the first place, or help to resolve them quickly and painlessly. I also learned that being reliable and communicating progress is critical to group success. I also learned from the personality tests that we took for the peer review assignment that those are two things that my personality type is notoriously bad at. Those are two skills that I was able to improve over the course of the term, but that I should continue to be aware of so that they do not negatively impact my career. My greatest regret of the term is ever letting my teammates down. But I did not just learn about handling negative team dynamics. I also learned how to foster a good team dynamic. I was able to bring cold brew coffee for my teammates on one of the days that we had class, which made us all feel a little more like a family, sipping our matching jars of coffee at 8am when we all would usually be asleep. I also learned about the power of delegation and asking for help, two things at which I know I am deficient. However, in being aware of this challenge, I was able to make the conscious choice to delegate some tasks and ask for help when I got stuck. There was still room for improvement, but it was a good start. Finally, I further developed my collaborative abilities. I really enjoyed working with my team this year, and I think that we produced a quality project and quality reports and documentation. I think a lot of this was due to the efficient and collaborative way that we worked together to get it all done.

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

If I could redo this whole project and the whole year, I would want to get the Configuration Space done way sooner. If we had talked to Dr. Smart earlier on in Fall Term and found the motors that we needed before Winter Term, we would have been able to get a lot more done. However knowing who to talk to and what hardware we needed was a challenge of its own. It would have been helpful to start the fundraising earlier as well. If I had been introduced to my contact at the OSU Foundation earlier in the year, he might have been able to help us apply for more grants. But once again that is not something that I could change.

One technical change that I would make is to increase the power of the on-board computer as that is a serious limiting factor for our ability to perform the pathing computations on orbit, or even storing a full, detailed configuration space. Our choice of on-board computer was somewhat limited by budget. In the future, further fundraising or support from the school for missions could alleviate this concern.

10.2 Amber Horvath

10.2.1 Technical Information

During the course of this project, I had the honor of adapating new technical abilities. Prior to this project, I had never worked with any microcontrollers, as I am an HCI-applied option computer science major and am not required to take ECE 375. Thus, when we were developing code for the ATMega128, a lot of the architecture and developmental practices were completely new to me. While I was very familiar with C, I had never written code that actually control an embedded system. Gaining familiarly with the different registers and ports in order to write our project was helped by the electrical engineers, but when I was doing the SD Card development, I was almost entirely

alone in trying to follow the order of calls through a 6,000+ line file of complex C code. So the act of syntesizing my existing programming knowledge with this new, foreign ATMega128-specific code lead to me honing my existing programs more but also allowing me to strengthen my ability to adapt to confusing situations and gain a better top-level down understanding of other people's code. After working on this project, I feel more confident in my ability to write C code and to understand and evaluate other code given to me.

10.2.2 Non-Technical Information

On top of just technical skills, I have also gained non-technical skills. This project has allowed me to learn how to communicate better with people outside of my major as I frequently had to work with the mechanical engineers and electrical engineers who do not have the same level of domain knowledge when it comes to programming. They, however, are more knowledgeable in their respective domains so ensuring that we were able to simplify complex topics to a point that we both understand while not losing any of the important intricacies or nuances too much was an important skill that I feel I enhanced over the course of the term. In the beginning of the project, I feel our team had many more miscoummunications than in the later parts of the project.

10.2.3 Project Work Information

Most of the information I learned from working in this project, I had learned to a much smaller degree in some of my previous group-work ventures. However, this was the first time I had done work an actual client so that made the dynamic slightly different as every change we made to the project structure or large implementation efforts needed to be reviewed by her. Nancy was a great client, though, and was consistently supportive and impressed with the work we had accomplished. I also learned a fair deal about the large life cycle these projects go through. I have never attempted to send anything into space (not surprising), but it takes a long time and a lot of effort, both on our part and our sponsors part, to organize and strategize how to get multiple teams from universities all across the country to be ready and able to send whatever their project is into space. I was consistently impressed with how professional and knowledgeable our contact in Colorado for the RockSat-X program was both on our project and all the various components that needed to be function correctly in order for our payload to be onboarded and sent on the ship. I never knew just how much needed to be done in order to get these payloads onto a ship.

10.2.4 Project Management Information

In terms of project management, I learned how important it is to have consistent meetings and smaller, sub-team meetings. During the most intense development period, winter term, our group splintered into sub-teams that were required to meet once a week to discuss work that had been done cross-functionally (such as one mechanical engineer, one electrical engineer, and one computer science team member all meet) to discuss what work has been done on their end of the project and whether they needed anything from one of these other disciplines. The teams also had "goals" in place to ensure that all the parts that required knowledge from all three groups would get completed. For example, I was part of the "Structures" group that had the goal of making the hardware of the arm with the electrical engineer's motors were all wired and able to move freely without getting caught. It was important to have all discplines represented within this group as all of us had different requirements that we were more knowledgeable about (for example, I was more familiar with the pathing through space we wanted the arm to achieve). I had never worked in smaller groups

like this that had such focused goals, but I really enjoy this method of breaking a large group into focused sub-teams to ensure that all the moving parts of a project are not only done, but done with respect to all the other parts of the larger project. This was crucial to the success of a project like ours that is cross-discplinary. However, I could see it working in some other settings I've been in such as my research group where there are lots of us all working on different projects but all need to come together to work on one final paper, for example.

10.2.5 Team Work Information

I learned a lot about team work through this project. The cross-discplinary aspect allowed me to learn better how to work within a larger team, as discussed in "Project Management Information". I also learned a lot about my fellow computer science teammates during this project, though. I learned how important clear and effective communication is, especially when things start crumbling. My team and I had a difficult time during early winter term due to teammates not clearly communicating whether they were doing the work assigned. We also had difficulties involving work being done far later than expected and series of excuses as to why these things may be late. I learned that I think I am a bit too easily persuaded by these issues. Instead of forcing my other teammates to do the work assigned to them, I normally just gritted my teeth and did the extra work given to me. However, in my future endeavors, I want a more respectful two-way street when it comes to work promised and work done. I want to work with people that say they will do the work, and actually do it. It is very stressful when this is not the case as I have experienced first hand, not only in this class but in other team projects as well, but I now am more resolved to not allow for this treatment any longer. While this all sounds very negative, for the most part, I really enjoyed my team and am proud of the work we accomplished. I just wish it wouldn't have been so stressful at some points, and I feel I could've mitigated some of these issues.

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

As just mentioned in "Team Work Information", I would be more assertive when discussing how important getting the work done on this project is to me. I would advocate more for better communication practices earlier on. I believe a lot of the issues I have already mentioned could've been avoided with these measures put in place. Beyond these complains, I wouldn't change anything. I'm really happy with this project and the work we've done. Seeing that beautiful arm move and getting to say I sent it into space: I wouldn't change that at all.

10.3 Michael Humphrey

10.3.1 Technical Information

Through this project, I was able to expand my technical knowledge in many different ways. From my knowledge of C/C++, I learned how to program in embedded C. Although it is the same language as vanilla C, there are many subtle but important differences that a programmer needs to know. I learned many of these tips and tricks from members of the Electronics team who have worked extensively with embedded C. Additionally, I used my knowledge of programming the ATmega128 chip in assembly from ECE 375 to understand how to program the same chip in C. While programming in C provides the familiar constructs such as conditionals and loops, I still needed to interact with the hardware through specific bits in various registers. I was able to build on what I knew from ECE 375 to accelerate my learning with interacting with the ATmega128 chip in C.

10.3.2 Non-Technical Information

I learned a lot about robots, robotic movements, and spacecraft over the course of the past year. This is the first project I've worked on that can interact with its environment outside the context of a computer. It brings in a whole lot of complexity dealing with physics, inverse kinematics, and other properties of real-world environments in comparison to the mathematically well-defined environment that computer programs operate in virtually. It has exposed me to the reality that software exists to solve real-world problems, not abstract problems commonly in academia.

10.3.3 Project Work Information

I learned a lot this year working on a project of this scale. It is a great experience to see many different parts of the project made by many different people come together to accomplish a single goal. That being said, I think there were a lot of shortcomings as well. Each of the CS team members took ownership of a specific portion of the code. This helped reduce version control conflicts as well as made each of us expert in that area of the code. However, documentation was definitely lacking in all areas. It is highly unlikely that one team member could quickly and efficiently fix a bug in another team member's code.

10.3.4 Project Management Information

I have learned that project management is an important part of the success of the product. Without proper management, deadlines can slip by unnoticed and implementation can stall with petty squabbles early in the design phase. It takes an excellent project manager to keep the project moving in a positive direction and everyone working together productively. I have learned to identify weaknesses in my team and step up my duties to that of a product manager when I notice things aren't happening as they need to be.

10.3.5 Team Work Information

I have learned a tremendous amount about working with other engineers on a project. I learned there needs to be a huge amount of coordination between engineering disciplines in order to minimize time wasted. The CS team depended on the designs of the EE team, which in turn depended on the designs of the ME team. As it were, the CS team had the least amount of time to accomplish our tasks because we had to wait for the EEs to finalize their design. There were many ways this could have been parallelized so there was less time wasted for other teams.

On the other hand, there was a large amount of coordination between engineering disciplines on the design of the payload. Every team had input at each stage of the design. The amount of collaboration was impressive, and it helped to mitigate issues that may have otherwise gone unnoticed until implementation.

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

If I had to do this project over again, I wouldn't change a lot. I would try to parallelize the implementation of the different parts of the payload (i.e. the mechanical, electrical, and software components). I would also work to mitigate the interpersonal difficulties that came up in the various groups throughout the year.

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++) {
```

```
44            CSpace[a][b][c][d] = 255;
45          }
46        }
47      }
48    }
49
50    //open file
51    myFile = "./test1/data1";
52    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       }
```

```
    }
    outfile << "\n";
  }
  */

}
```

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
```

```
44  int theta0, theta1, theta2, theta3; //cspace target point
45  double l1, l2; //arm segment lengths
46  double k1, k2, gamma; //variables for calculating pitch
47  double t, b, fraction1, pt1, pt2; //some more variables for calculations
48  bool endOfFile = 0; //marker for eof
49
50  //define arm lengths
51  l1 = 3.99;
52  l2 = 4.49;
53
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;

  }

  //close the file
  if(sourceFile.is_open()) {
    sourceFile.close();
  }

  //close the file
  if(outfile.is_open()) {
    outfile.close();
  }


  return 0;
}
```

11.1.4 pathing.cpp

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

using namespace std;

/*****************************************************************************
 * Program: Pathfinding within IR4 C-Space
 * Author: Helena Bales
 * Date: April 19th, 2017
 * Description: Finds a valid path using Dijstra's through the 4D binary array
    located in cspace.
 *  This is a draft of the implementation which will later be converted to run on the
    AVR
 *  board.
 * Input: ./cspace - Hephaestus C-Space data as defined in project docs
 * Output: ./cspace - Output file containing boolean data from 37x37x37x37 array
 *****************************************************************************/

int main() {
  //for file io
  char* myFile;
  ifstream sourceFile;
  ofstream outfile;
  string line;

  //for dijkstra's
  int accuracyFactor = 10;
  int MAX_ANGLE = 360;
  int x = MAX_ANGLE/accuracyFactor;
  int a, b, c, d, pos, temp[4];
  bool endOfFile = 0;
  int c0, c1, c2, c3, n0, n1, n2, n3;

  //for getting the targets
```

```
42  char cline[24];
43  char* tokens;
44  int theta0, theta1, theta2, theta3;
45  queue<int> targets0, targets1, targets2, targets3;
46
47  unsigned char CSpace[37][37][37][37], tempchar;
48
49  pos = 0;
50
51  //open file
52  myFile = "./cspace";
53  sourceFile.open(myFile);
54
55  //read the line from the file (it is one line long)
56  getline(sourceFile, line);
57
58  //loop through the array and line, copying the line to the array
59  for(a=0; a<x; a++) {
60    for(b=0; b<x; b++) {
61      for(c=0; c<x; c++) {
62        for(d=0; d<x; d++) {
63          tempchar = line.at(pos);
64          CSpace[a][b][c][d] = tempchar;
65          pos++;
66        }
67      }
68    }
69  }
70
71  //close source file
72  if(sourceFile.is_open()) {
73    sourceFile.close();
74  }
75
76  //open the file of thetaTargets
77  myFile = "./thetaTargets";
78  sourceFile.open(myFile);
79
80  //read in all the targets
81  while(!endOfFile) {
82    //get a line from the file
83    sourceFile.getline(cline, 24);
84
85    //tokenize on spaces
86    tokens = strtok(cline, " ");
87
88    //store the values as ints
89    theta0 = stoi(tokens);
90    tokens = strtok(NULL, " ");
91
92    theta1 = stoi(tokens);
93    tokens = strtok(NULL, " ");
94
95    theta2 = stoi(tokens);
96    tokens = strtok(NULL, " ");
97
98    theta3 = stoi(tokens);
99    tokens = strtok(NULL, " ");
100
101   cout << "check" << endl;
102
103   //push them onto the list of targets
104   targets0.push(theta0);
105   targets1.push(theta1);
```

```
106       targets2.push(theta2);
107       targets3.push(theta3);
108
109       if(sourceFile.eof() || sourceFile.peek() == 10) {
110         cout << "eof reached" << endl;
111         endOfFile = 1;
112       }
113     }
114
115     if(sourceFile.is_open()) {
116       sourceFile.close();
117       cout << "sourceFile closed" << endl;
118     }
119
120     //print targets for testing
121     for(int i=0; i<targets0.size(); i++) {
122       theta0 = targets0.front();
123       theta1 = targets1.front();
124       theta2 = targets2.front();
125       theta3 = targets3.front();
126       targets0.pop();
127       targets1.pop();
128       targets2.pop();
129       targets3.pop();
130       cout << theta0 << " " << theta1 << " " << theta2 << " " << theta3 << endl;
131       targets0.push(theta0);
132       targets1.push(theta1);
133       targets2.push(theta2);
134       targets3.push(theta3);
135     }
136
137     //dijkstra's algorithm through cspace from <0,0,0,0> through each of the targets
138     //in targets queue
139
140     //set starting location
141     c0 = 0;
142     c1 = 0;
143     c2 = 0;
144     c3 = 0;
145
146     //store the number of targets for later use
147     a = targets0.size();
148
149     //for each target do dijkstra's algorithm from c0-4 to n0-4 through CSpace
          [37][37][37][37]
150     for(int i=0; i<a; i++) {
151       //get the next target
152       n0 = targets0.front();
153       n1 = targets1.front();
154       n2 = targets2.front();
155       n3 = targets3.front();
156       targets0.pop();
157       targets1.pop();
158       targets2.pop();
159       targets3.pop();
160     }
161
162     //print the updated cspace to outfile
163     //open outfile
164     outfile.open("./path");
165
166     //print the cspace
167     for(a=0; a<x; a++) {
168       for(b=0; b<x; b++) {
```

```
      for(c=0; c<x; c++) {
        for(d=0; d<x; d++) {
          outfile << (int)CSpace[a][b][c][d] << " ";
        }
      }
    }
  }

  //close the file
  if(outfile.is_open()) {
    outfile.close();
  }

  return 0;
}
```

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);

```

```
26        motor_dir(MOTOR_CAMERA, COUNTER_CLOCKWISE);
27
28        motor_step(MOTOR_CAMERA, DEGREES_TO_STEPS(360), 1, 85);
29
30        _delay_ms(500);
31
32        motor_dir(MOTOR_CAMERA, CLOCKWISE);
33
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);
44    motor_dir(2, 0);
45    motor_dir(3, 0);
46    motor_dir(4, 0);
47
48    //find the next point in the path
49    path_step = cspace[M1_POS][M2_POS][M3_POS][M4_POS];
50
51    //set the lower and upper bounds for the loops so only indeces
52    //in range are checked
53    if(M1_POS == 0) { sw = 0; } else { sw = -1; }
54    if(M2_POS == 0) { sx = 0; } else { sx = -1; }
55    if(M3_POS == 0) { sy = 0; } else { sy = -1; }
56    if(M4_POS == 0) { sz = 0; } else { sz = -1; }
57
58    if(M1_POS == 36) { ew = 1; } else { ew = 2; }
59    if(M2_POS == 36) { ex = 1; } else { ex = 2; }
60    if(M3_POS == 36) { ey = 1; } else { ey = 2; }
61    if(M4_POS == 36) { ez = 1; } else { ez = 2; }
62
63
64    //check each of the neighboring points
65    for(int w=sw; w<ew; w++) {
66      for(int x=sx; x<ex; x++) {
67        for(int y=sy; y<ey; y++) {
68          for(int z=sz; z<ez; z++) {
69            if(cspace[M1_POS+w][M2_POS+x][M3_POS+y][M4_POS+z] == path_step + 1) {
70              M1_NXT = M1_POS + w;
71              M2_NXT = M2_POS + x;
72              M3_NXT = M3_POS + y;
73              M4_NXT = M4_POS + z;
74              break;
75            }
76          }
77        }
78      }
79    }
80
81    if(M1_NXT == M1_POS && M2_NXT == M2_POS && M3_NXT == M3_POS && M4_NXT == M4_POS)
      {
82      //end of path has been reached
83      //TODO send eop telemetry signal
84      return 0;
85    }
86
87    //change _NXT values to be less than 360
88    if(M1_NXT > 359) {
89      M1_NXT = M1_NXT - 359;
90    }
91
92    if(M2_NXT > 359) {
93      M2_NXT = M2_NXT - 359;
94    }
95
96    if(M3_NXT > 359) {
97      M3_NXT = M3_NXT - 359;
98    }
```

```
99
100     if(M4_NXT > 359) {
101       M4_NXT = M4_NXT - 359;
102     }
103
104     // calculate change in each motor (in degrees)
105     M1_DIF = M1_NXT - M1_POS;
106     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>
3  #include <util/delay.h>
4  #include "RSXAVRD.h"
5  #include "phases.h"
6  #include "retract.h"
7  #include "MOTOR_DEF.h"
8  #include "telemetry.h"
9
10 uint8_t plate_retracted_flg; // flag to keep track of plate's position
11
12 ISR(INT5_vect){
13
14   if(plate_retracted_flg == 0x00){
15     retract();
16   }
17
18 }
19
20 void retract(){
21
22   motor_pwr(MOTOR_DECK_PLATE, POWER_ON);
23
24   _delay_ms(500); // delay for motor after powering on
25
26   motor_pwr(MOTOR_CAMERA, POWER_OFF); // turn off all other motors
27   motor_pwr(MOTOR_DECK_ARM, POWER_OFF);
28   motor_pwr(MOTOR_PAN, POWER_OFF);
29   motor_pwr(MOTOR_SHOULD, POWER_OFF);
30   motor_pwr(MOTOR_ELB, POWER_OFF);
31
32   camera_enable(POWER_OFF);
33
34   motor_dir(MOTOR_DECK_PLATE, COUNTER_CLOCKWISE); // rotates the deck plate to
35
36   motor_step(MOTOR_DECK_PLATE, 1650, 28, SPEED + 19); // the amount of steps needed
       to pull the arm back in
37
38   eeprom_log("deck plate has been retracted");
39
40   plate_retracted_flg = 0x01;
41
42
43 }
44
45 void extend(){
46
47   motor_pwr(MOTOR_DECK_PLATE, POWER_ON); // powers on deck plate motor
48
49   _delay_ms(500); // delays to allow motor to power on
50
51   motor_dir(MOTOR_DECK_PLATE, CLOCKWISE); // push the deck plate out
52
53   motor_step(MOTOR_DECK_PLATE, 1650, 28, SPEED + 19); // the amount of steps needed
       to move the deck plate at a good speed
54
55   plate_retracted_flg = 0x00; // plate is NOT retracted
56
```

```
57
58 }
```

11.3.3.5 safety.c

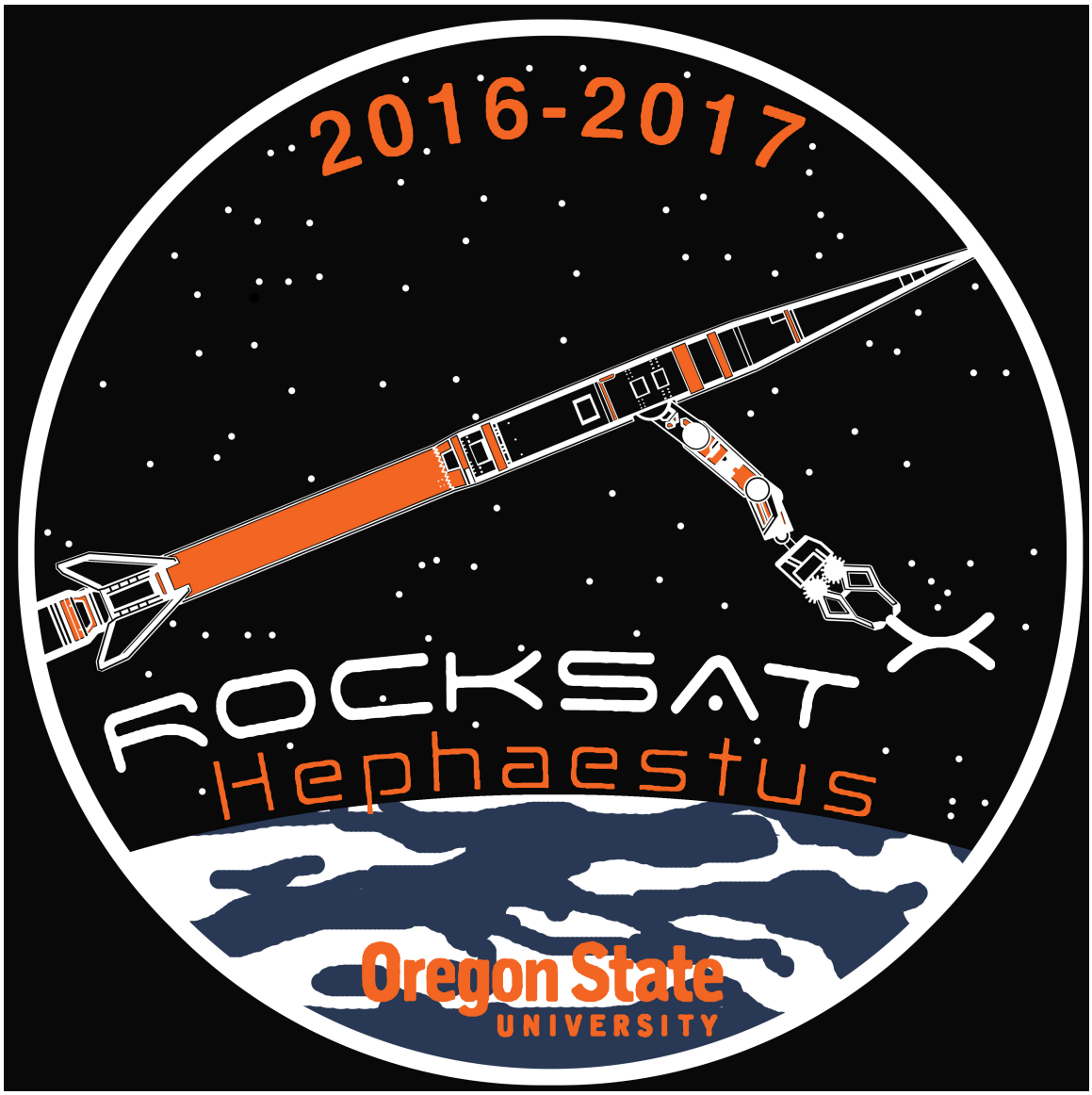
```
1  /* safety.c
2   *
3   * Created by: Amber Horvath
4   * Date Created: 1/22/17 5:36:47
5   *
6   *
7   */
8
9  #include <avr/io.h>
10 #include "phases.h"
11 #include "retract.h"
12 #include "telemetry.h"
13
14 int safety(void){
15
16   eeprom_log("in safety - attempting retract");
17
18     retract();
19
20     while(1){}; // abort mission, we are hanging here
21
22 }
```

11.3.3.6 off.c

```
1  #include "telemetry.h"
2
3  int off(void) {
4
5    eeprom_log("power off");
6
7    while(1){};
8
9    // This return statement should never be reaches, so return error.
10   return 1;
11 }
```

12 Appendix 2: Other Documents

12.1 Mission Logos



Simple Mission Logo for shirts and the like.

12.2 Team Photos

12.2.1 All Team Photo

From left to right: Ian Finn, Nancy Squires, Jonathan Hardman, Cole Morgan, Helena Bales, Sam Lundeen, Subret Aryal, Amber Horvath, Michael Polander, Brett Moffatt, Devin Wyckoff, Michael Humphrey, and Huy Nguyen.

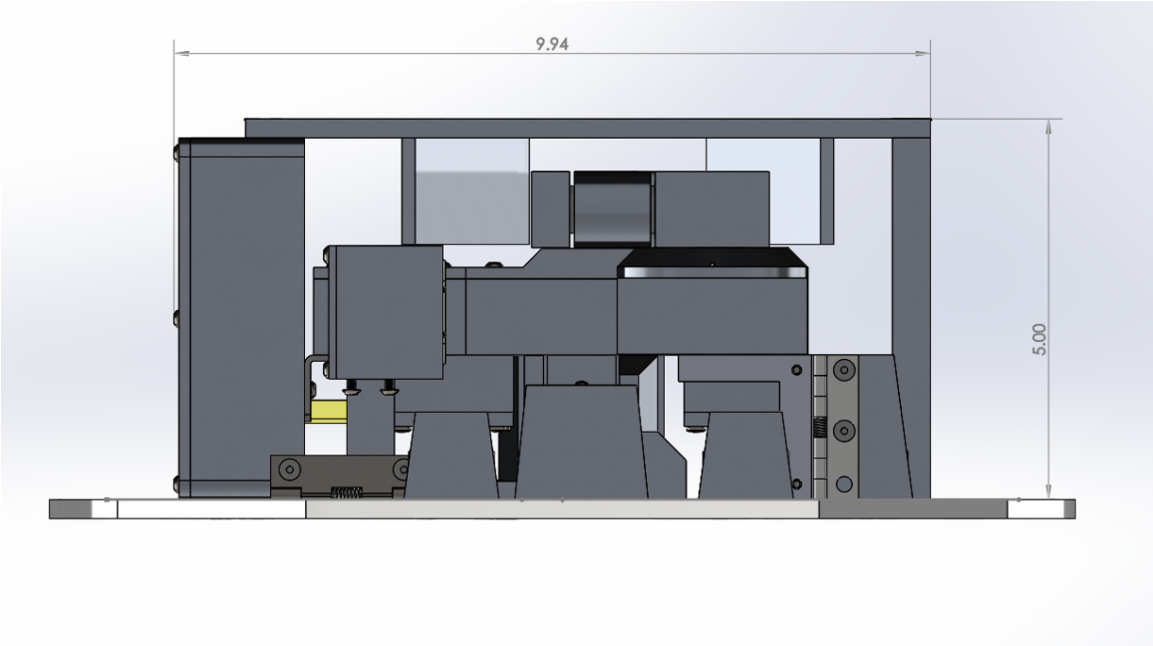
12.2.2 Software Team Photo

From left to right: Michael Humphrey, Helena Bales, Amber Horvath, and James Wilson.

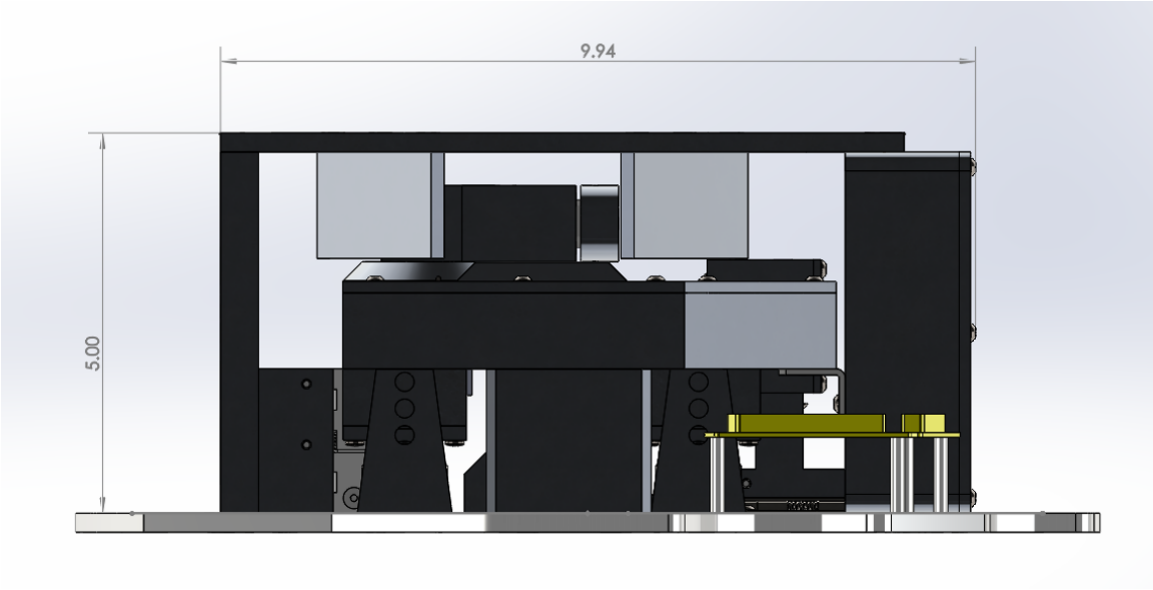
12.3 CAD Models

CAD Models created by the Mechanical Engineering teams of Hephaestus.

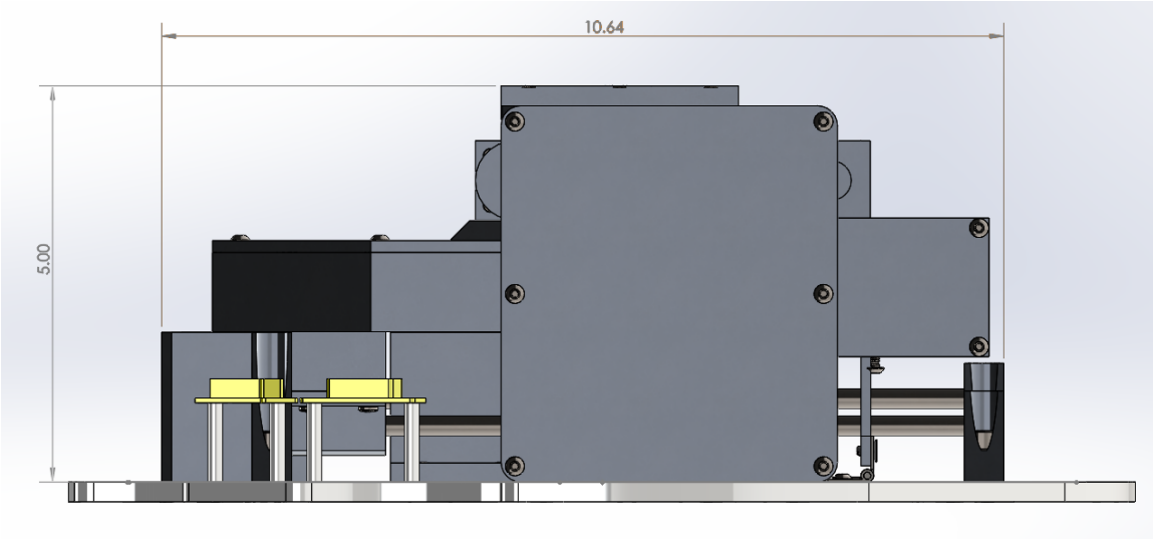
12.3.1 Front View



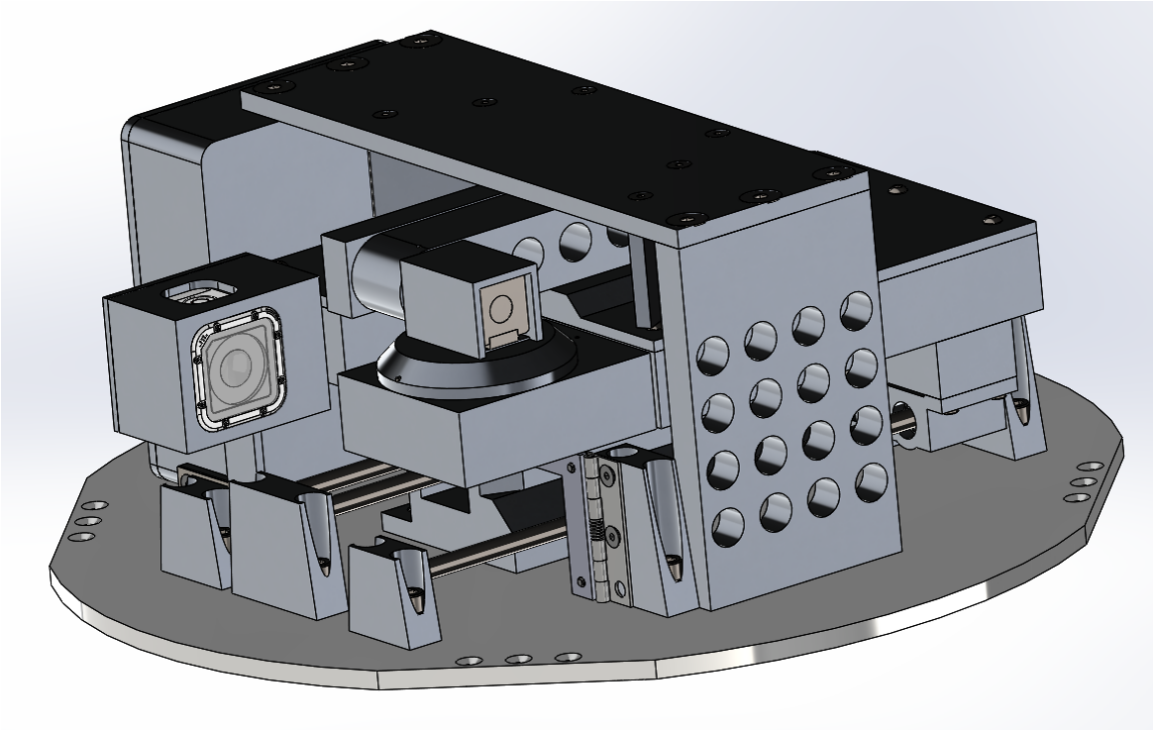
12.3.2 Back View



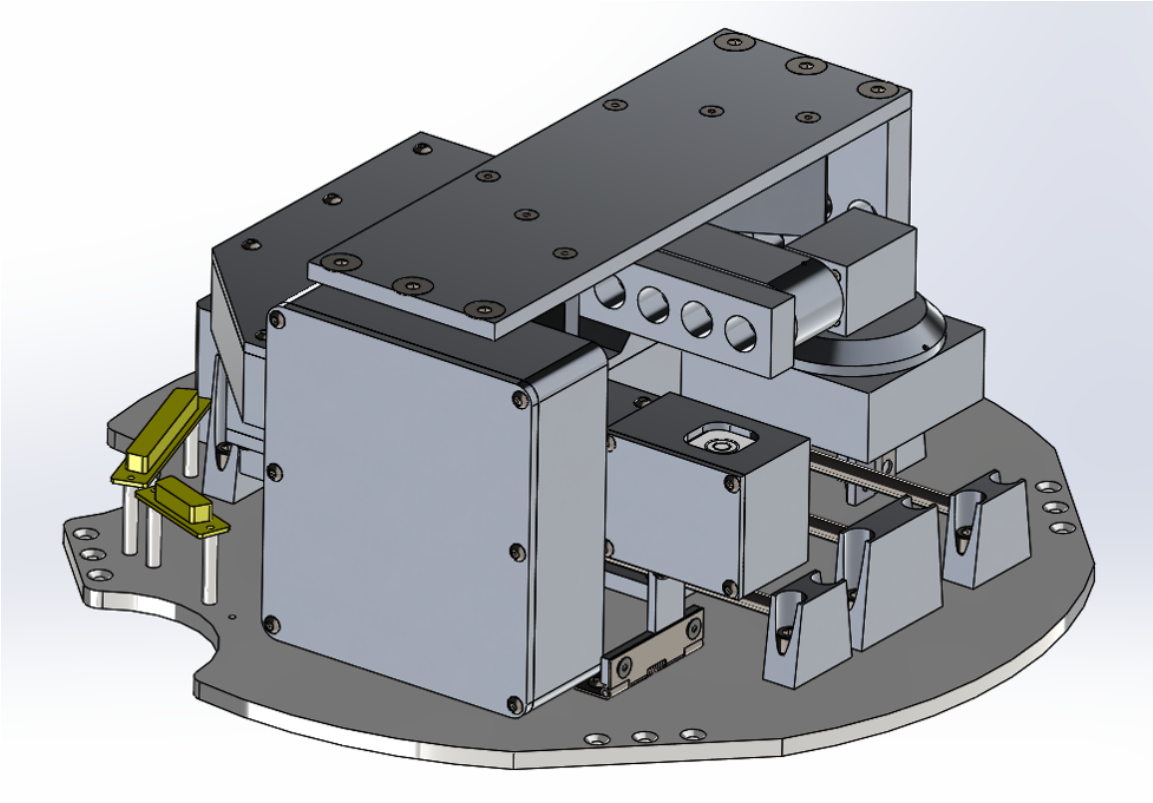
12.3.3 Right View



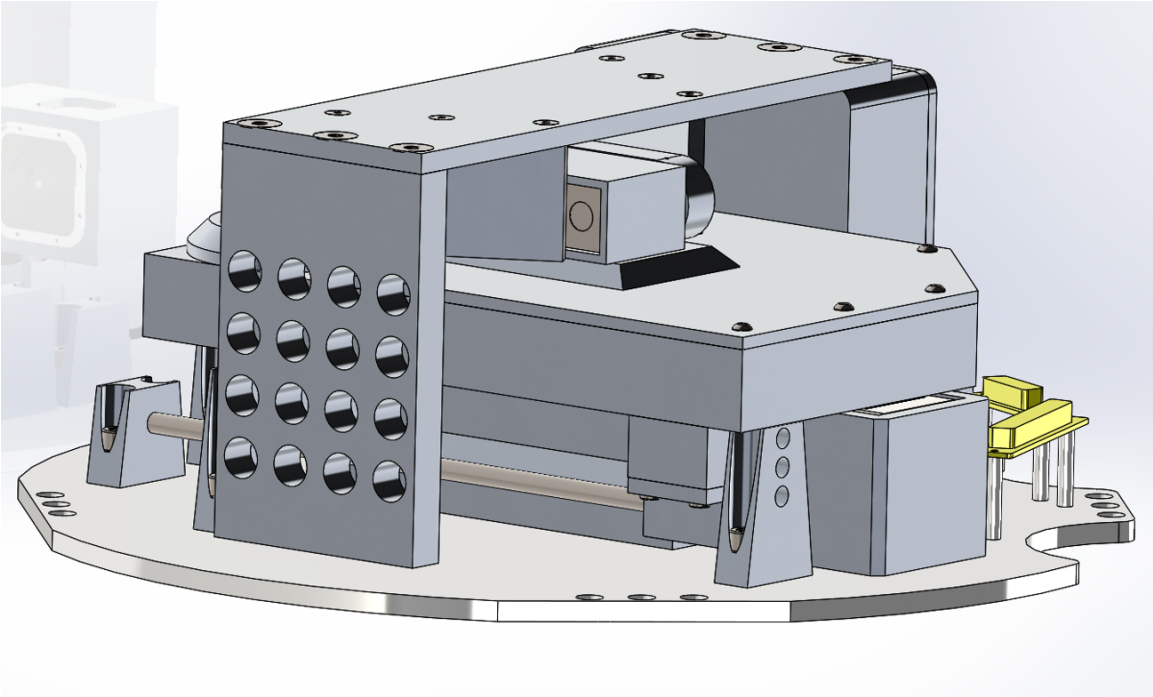
12.3.4 ISO 1 View



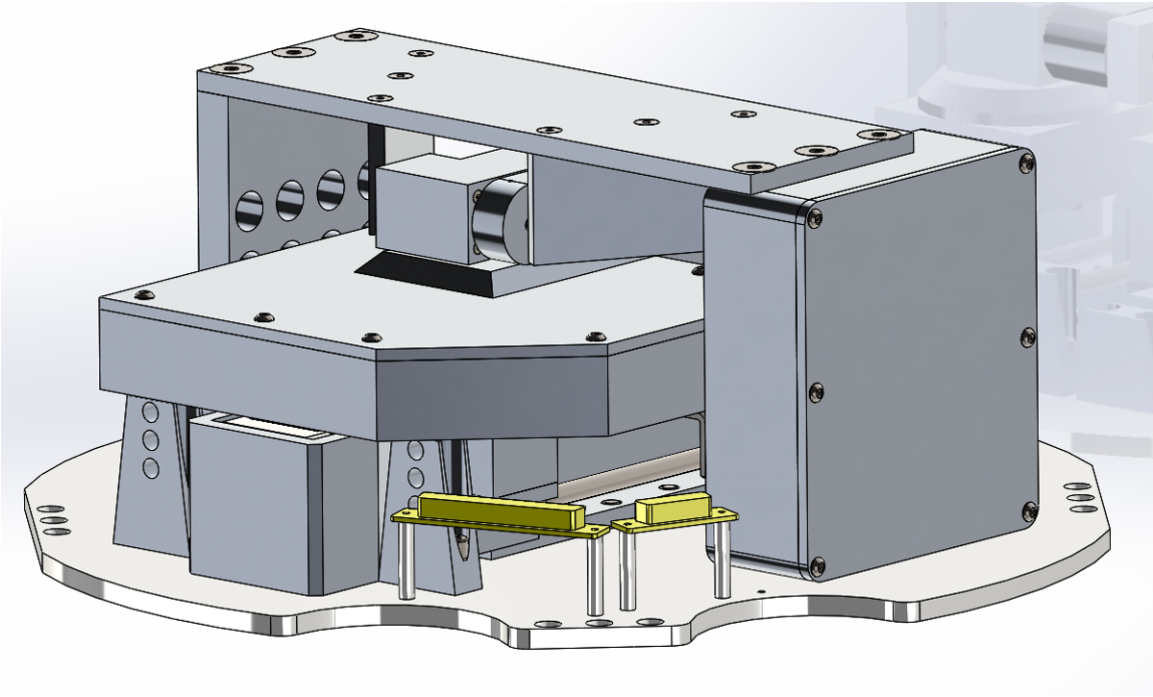
12.3.5 ISO 2 View



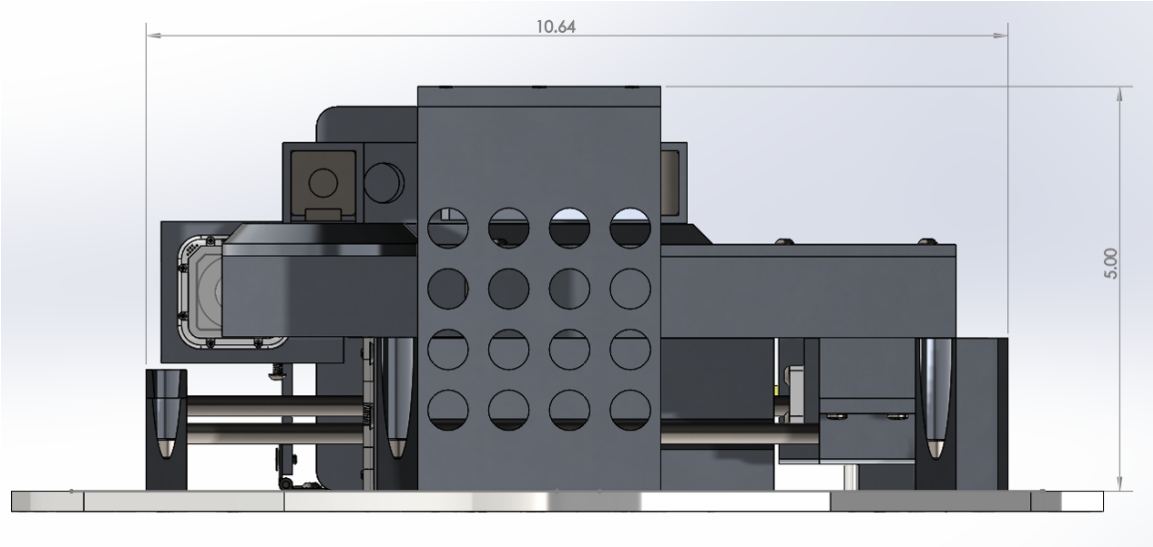
12.3.6 ISO 3 View



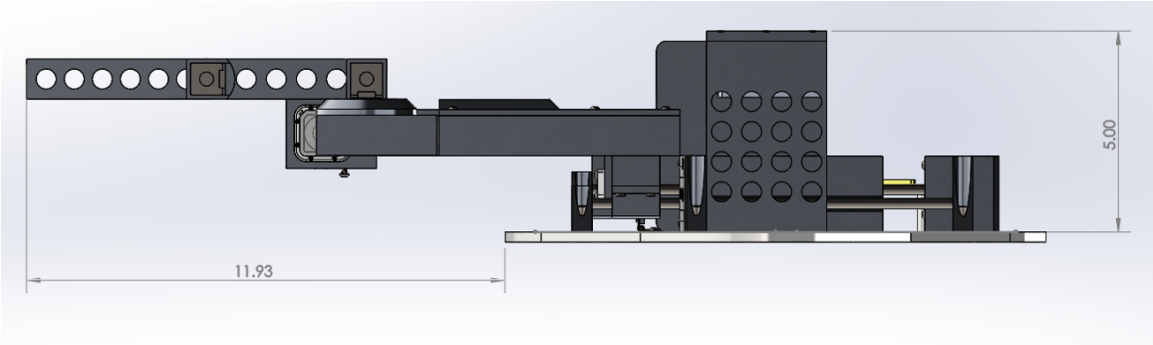
12.3.7 ISO 4 View



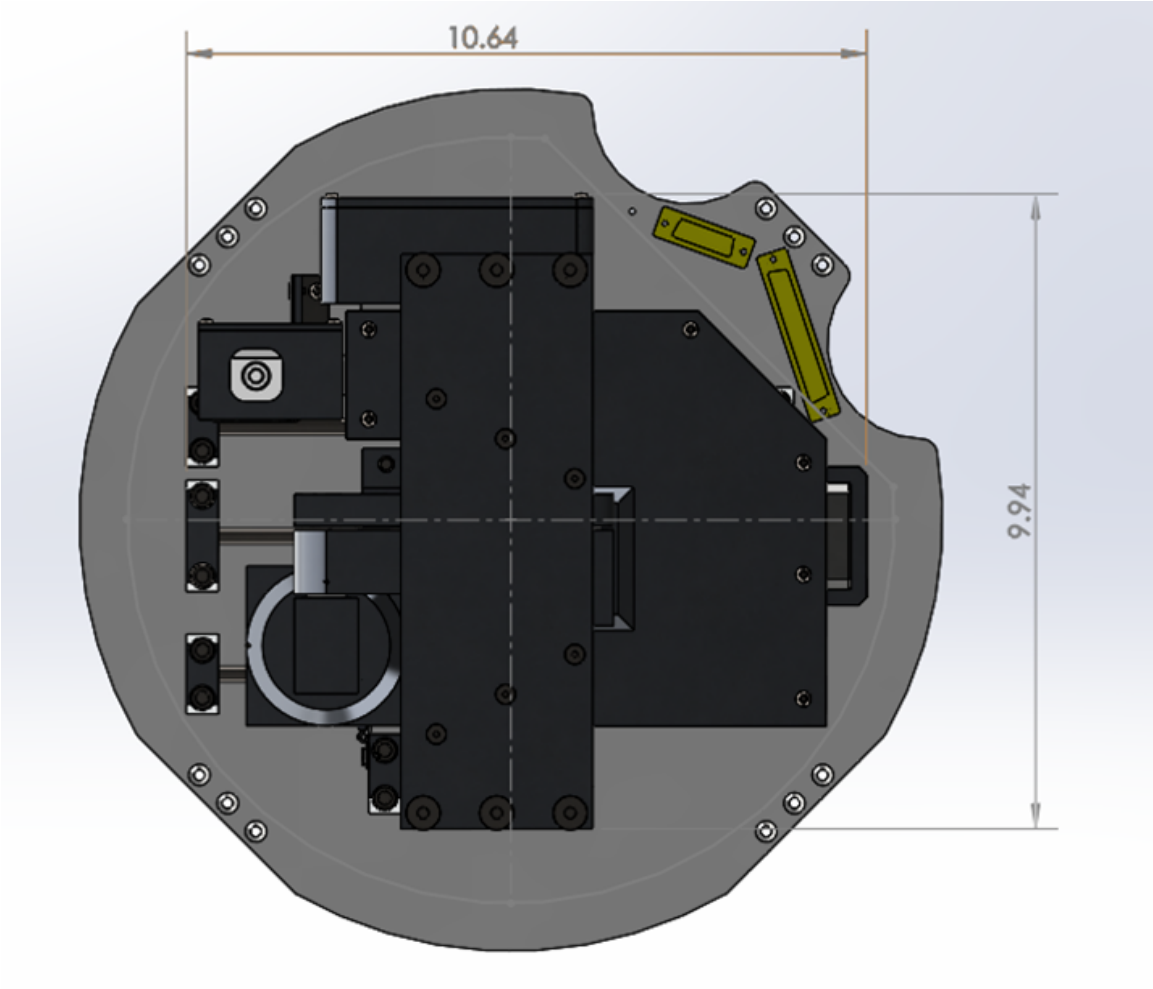
12.3.8 Left View



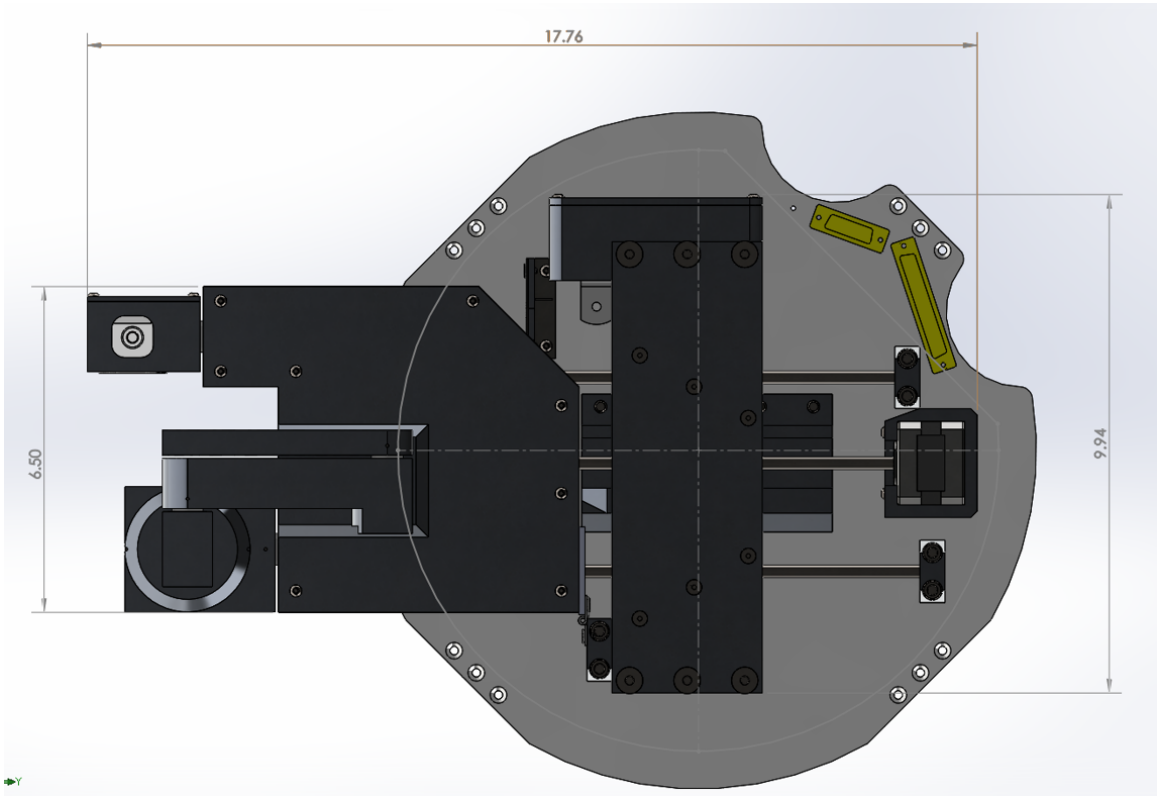
12.3.9 Left Fully Deployed View



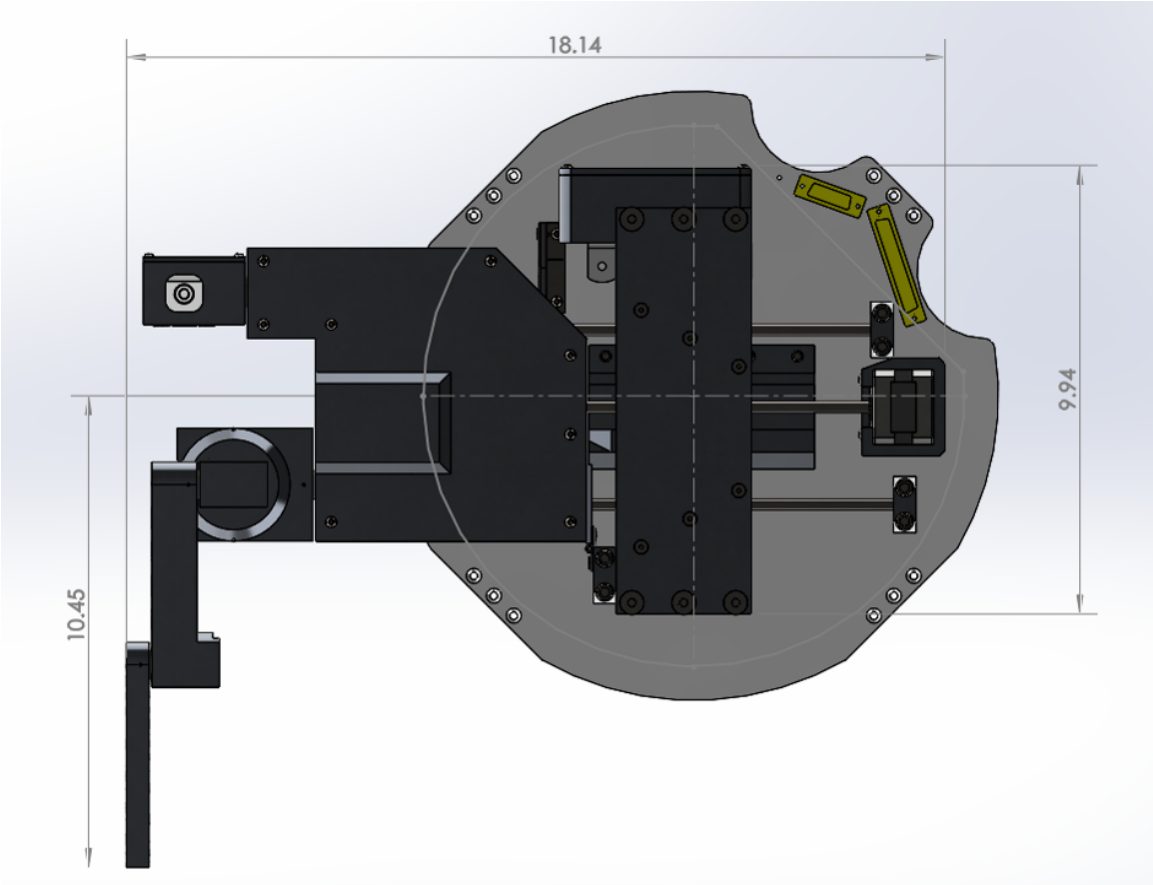
12.3.10 Top View



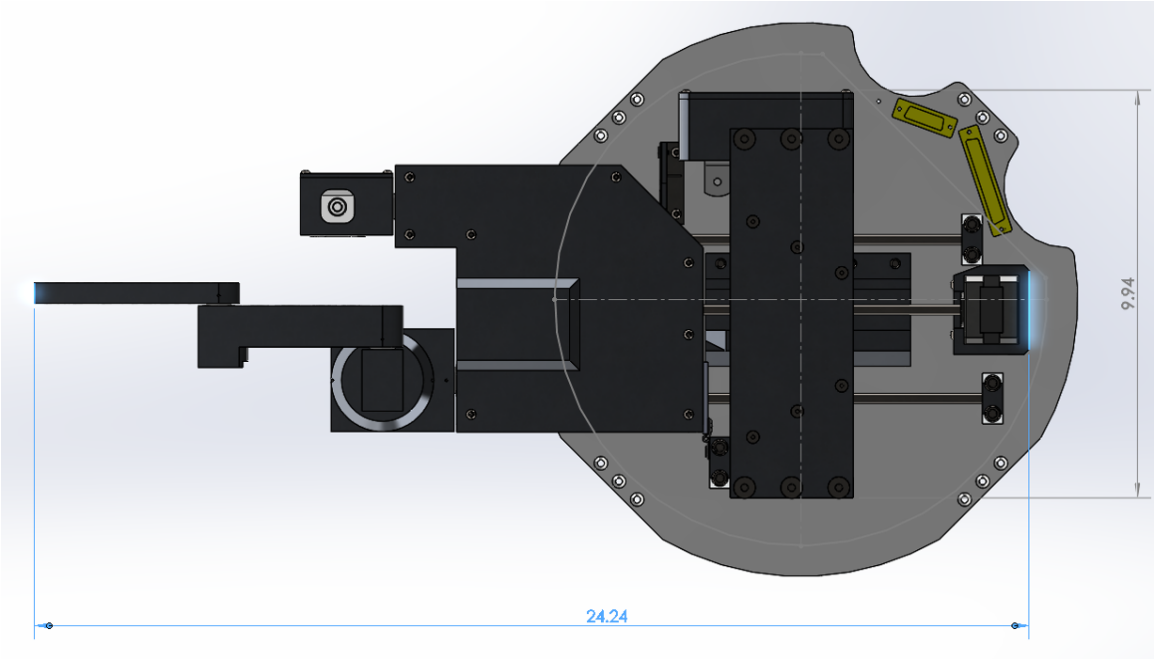
12.3.11 Top Deployed View



12.3.12 Top Deployed Angle View



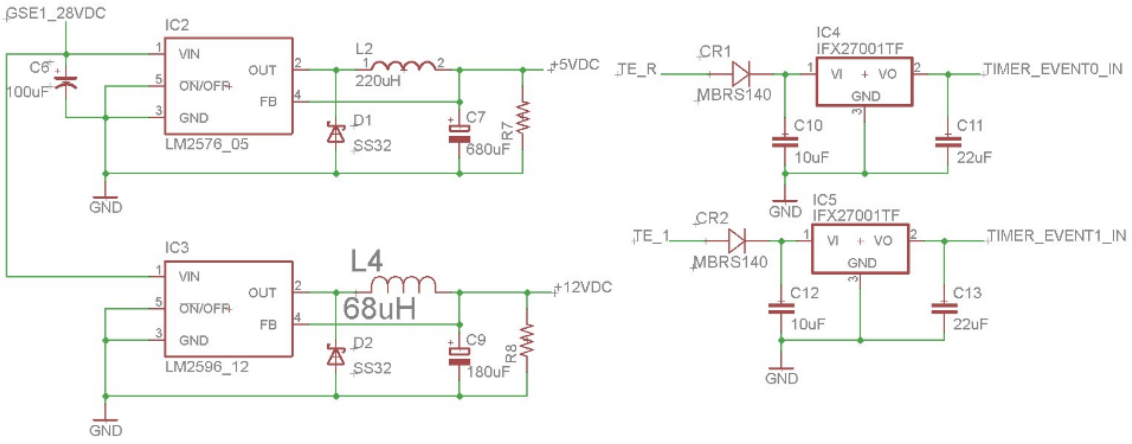
12.3.13 Top Fully Deployed View



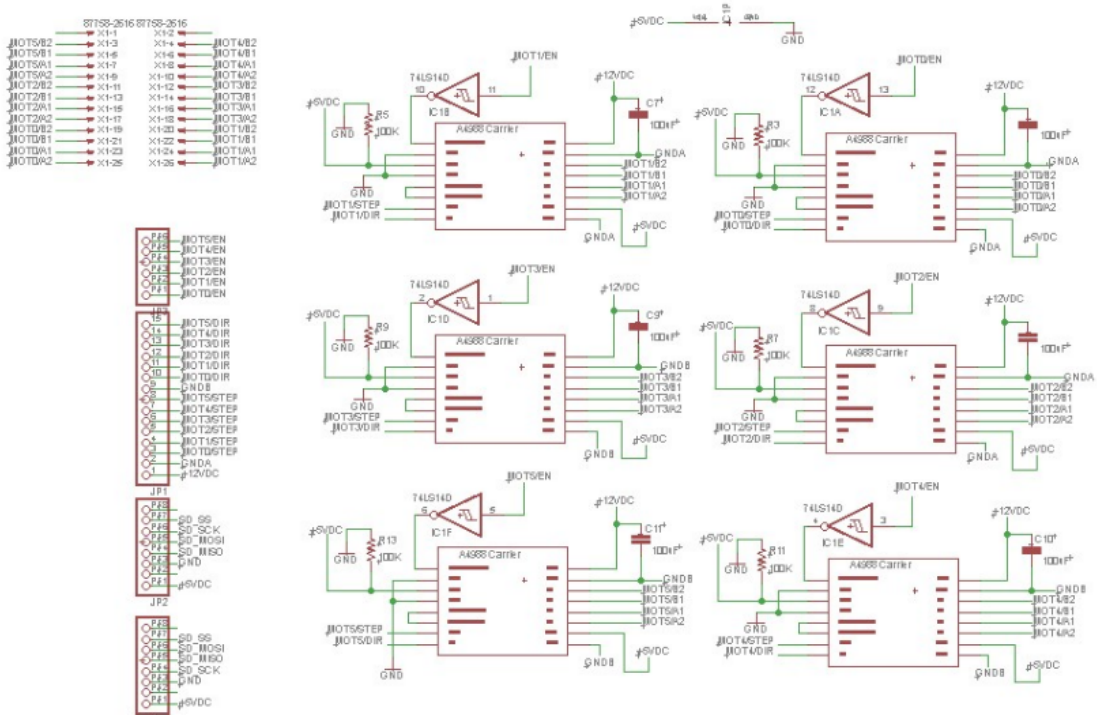
12.4 Electrical Schematics

Electrical Schematics created by the Electrical Engineering teams of Hephaestus.

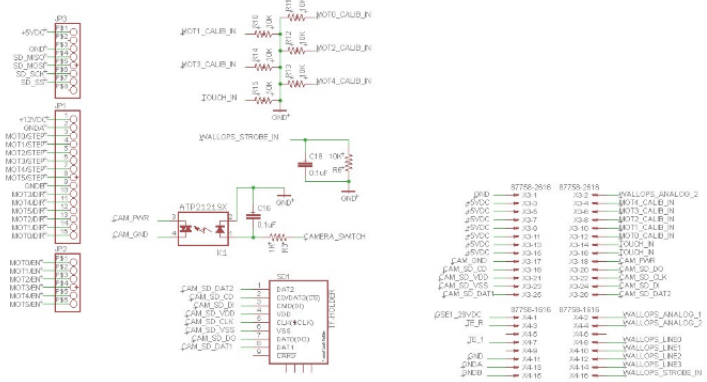
12.4.1 Voltage Regulator

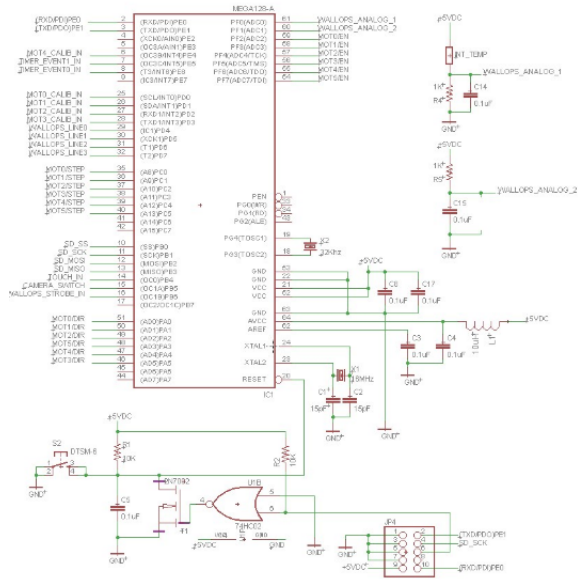


12.4.2 Motor Driver



12.4.3 AVR





12.5 Power Budget

RockSat-X - Power Budget						
12/6/16						
Subsystem	**Voltage (V)**	**Max Current (A)**	**Start Time (min)**	**Time On (min)**	**Watts**	**Ah**
Motor	12.0	1.80	1.93	2.9	21.60	0.09
Motor Driver	5.0	0.08	1.93	2.9	0.40	0.00
Microcontroller Idle	5.0	0.02	-3	4.33	0.10	0.00
Microcontroller	5.0	0.48	1.93	2.9	2.40	0.02
Temperature Sensor	5.0	0.00	-3	8.63	0.01	0.00
	Total	2.38			24.51	0.12
	Total Power Capacity					0.50
	Over/Under					0.38
				# of Flights Margin		**8.6**

12.6 Mass Budget

Mass Budget - Model	
Subsystem	**Total Mass (lbf)**
Mechanical Arm	5.26
Thermal Enclosure	1.27
Electronics • Voltage Regulator • Control unit • Processing	0.18
Other	5.07
Deck Plate	3.25
Total	15.03
Over/Under	0.03

Mass Budget - Manufactured	
Subsystem	**Total Mass (lbf)**
Machined Parts	13.56
Electronics Components	0.75
Thermal Resistance Components	0.2125
Total	14.5225
Over/Under	0.4775

12.7 Pin Assignments

12.7.1 Power Pin Assignment

Power Pin	Function	Intended Use
1	GSE 1	Voltage regulator and microcontroller startup
2	Timer Event Redundant (TE-RA)	Deployment and program activation
3	Timer Event Redundant (TE-RB)	Deployment and program activation
4	Timer Event 1 (TE-1)	Deployment retraction fail/safe
5	GND	Voltage Regulator and Microcontroller
6	GND	Half of Motor Drivers
7	GND	Half of Motor Drivers
8	GND	
9	GSE 2	
10	Timer Event 2 (TE-2)	
11	Timer Event 3 (TE-3)	
12	GND	
13	GND	
14	GND	
15	GND	

12.7.2 Telemetry Pin Assignment

Telemetry	Function	Intended Use
1	Analog 1	Internal Temperature Sensor
2	Analog 2	External Temperature Sensor
3	Analog 3	
4	Analog 4	
5	Analog 5	
6	Analog 6	
7	Analog 7	
8	Analog 8	
9	Analog 9	
10	Analog 10	
11	Parallel Bit 1 (MSB)	
12	Parallel Bit 2	
13	Parallel Bit 3	
14	Parallel Bit 4	
15	Parallel Bit 5	
16	Parallel Bit 6	
17	N/C	N/C
18	Ground	
19	Ground	
20	Parallel Bit 7	
21	Parallel Bit 8	
22	Parallel Bit 9	
23	Parallel Bit 10	
24	Parallel Bit 11	
25	Parallel Bit 12	
26	Parallel Bit 13	Status Code Bit 3
27	Parallel Bit 14	Status Code Bit 2
28	Parallel Bit 15	Status Code Bit 1
29	Parallel Bit 16 (LSB)	Status Code Bit 0
30	Parallel Read Strobe	Sample Strobe Input
31	N/C	N/C
32	RS-232 Data (TP1)	
33	RS-232 GND (TP2)	
34	N/C	N/C
35	N/C	N/C
36	Ground	
37	Ground	

12.8 Timer Events

Event	Time (sec)	2 sigma Low Altitude (km)	Nominal Altitude (km)	2-sigma Hi Altitude	Nominal Range (km)	Velocity (m/s)	Nominal Q (psf)	Mach NO.	Flight Elevation (deg)	Event Control	Timer Type	Dwell Time (sec)
Terrier Ignition	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	84.0	Ground Fire	-	-
CTU: Launch Indicator	0.1	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	TM	UMFT	1.0
UHCC: Power to Experiment	0.1	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	TM	UMFT	335.0
VT: Launch Indicator	0.1	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	TM	UMFT	335.0
UPR: Launch Indicator and Power TE-R	0.1	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	TM	UMFT	332.0
UPR: Launch Indicator and Power TE-1	0.1	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	TM	UMFT	332.0
Rail Release	0.5	0.0	0.0	0.0	0.0	45	26.2	0.1	84.0	-	-	-
Terrier Burnout	5.2	1.5	1.5	1.5	0.2	558	3450.2	1.6	82.9	-	-	-
Imp. Malemute Ignition	18.0	6.6	6.7	6.8	0.8	289	544.3	0.9	80.4	WFF 4-event CDI	Elec	1
Imp. Malemute Burnout	29.7	16.7	17.2	17.6	2.8	1657	4175.3	5.7	78.8	-	-	-
WV: TE-2	62.0	60.6	63.1	65.4	12.7	1296	4.0	4.2	76.4	TM	UMFT	8
Heat Shield Enable	66.0	65.3	68.1	70.6	13.9	1259	25.3	4.2	76.1	TM	UMFT	600
De-Spin	68.0	67.6	70.5	73.2	14.5	1240	1.3	4.2	75.9	WFF 4-event CDI	UMFT	1
CTU: Iridium Starts	70.0	69.9	72.9	75.7	15.1	1222	0.9	4.2	75.7	Exp.		
Motor Separation	72.0	72.1	75.2	78.2	15.7	1203	0.6	4.2	75.5	WFF 4-event CDI	UMFT	1
ACS Damp Rates	72.0	72.1	75.2	78.2	15.7	1203	0.6	4.2	75.5	ACS	Elec	6
ACS Valves OFF	78.0	78.6	82.0	85.3	17.5	1148	0.2	4.1	74.8	ACS	Elec	
Aft Skirt Separation	79.0	79.6	83.1	86.5	17.8	1139	0.2	4.1	74.7	TM	UMFT	1
UPR: Signal for Skirt Deploy	79.0	79.6	83.1	86.5	17.8	1139	0.2	4.1	74.7	TM	UMFT	258
ACS Roll and Pointing Align	80.0	80.6	84.2	87.7	18.1	1129	0.1	4.1	74.6	ACS	Elec	30
Nosecone Separation	82.0	82.7	86.4	89.9	18.6	1111	0.1	4.1	74.3	TM	UMFT	1
WV: TE-3	85.0	85.7	89.6	93.3	19.5	1083	0.0	4.1	73.9	TM	UMFT	125
300,000 Ft Upleg	86.8	87.4	91.4	95.2	20.1	1067	0.0	4.0	73.7	-	-	-
ACS On Target, Maintain Alignment	110.0	107.2	112.6	117.9	26.9	858	0.0	2.5	69.8	ACS	Elec	45
OSU: Experiment Deploy	116.0	111.4	117.3	122.9	28.6	805	0.0	2.2	68.5	TM	UMFT	5
VT: Antenna Deployment	135.0	122.8	129.8	136.5	34.1	641	0.0	1.6	62.8	TM	UMFT	
ACS Valves OFF	155.0	131.1	139.3	147.3	39.8	482	0.0	1.2	52.7	TM	UMFT	
CU: Start/Stop Video	185.0	136.5	146.6	156.4	48.4	309	0.0	0.8	19.6	TM	UMFT	120
Apogee, 2s Low	189.4	136.6			49.6	297	0.0	0.8	12.1	-	-	-
Apogee, Nominal	196.1		147.2		51.5	291	0.0	0.7	0.0	-	-	-
CU:Open Door & Eject Boom	200.0	136.1	147.1	157.8	52.6	293	0.0	0.7	-7.1	TM	UMFT	90
UK: Solenoid Activate	200.0	136.1	147.1	157.8	52.6	293	0.0	0.7	-7.1	TM	UMFT	5
Apogee, 2s High	204.2			157.8	53.8	300	0.0	0.8	-14.5	-	-	-
OSU: Forced Retract Failsafe	290.0	89.3	106.1	122.2	78.5	927	0.0	3.1	-71.3	TM	UMFT	5
ACS Spin-Up	305.0	74.1	91.8	108.9	82.9	1063	0.0	4.0	-73.6	ACS	Elec	25
300,000 Ft Downleg	305.4	73.7	91.4	108.5	83.0	1067	0.0	4.0	-73.7	-	-	-
UPR: Organic Collector Sys. Off	315.0	62.8	81.1	98.8	85.8	1155	0.2	4.1	-74.9	TM	UMFT	20
ACS Vent	330.0	44.3	63.4	82.0	90.3	1290	3.8	4.2	-76.5	ACS	Elec	60
ACS Valves OFF	390.0	3.0	6.2	10.6	101.9	272	507.5	0.8	-83.7	ACS	Elec	
Ballistic Impact (nominal)	419.0		0.0		104.0	174	389.2	0.4	-88.3	-	-	-
Chute Deploy	448.7		6.3		104.0	112	85.2	0.4	-85.9	Alt. Switch (20.5K)	-	-
Payload Impact	908.4	0.0	0.0		104.0	10	1.2	0.0	-90.0	-	-	-

12.9 Payload Placement

12.10 Launch Compliance

Requirement	*Status/Reason (if needed)*
Center of gravity in 1" plane of plate?	YES, (0.07, 0.08)"
Weight 30.0+/- 1.0 (15.0 +/- 0.5) lbs?	YES, 14.5225 lbs
Max Height < 10.75" (5.13")	YES, 5.0" from top of deck plate
Bottom of deck has flush mount hardware?	YES
Within Keep-Out Zone	YES
Using < 10 A/D Lines	YES, 2 ADC Lines
Using/Understand Parallel Line	YES, 4 Parallel Lines
Using/Understand Asynchronous Line	No Asynchronous Lines Used
Using X GSE Line(s)	YES, GSE 1
Using X Non-Redundant PWR Lines (TE-1, TE-2, TE-3)	YES, TE-1
Using X Redundant Power Lines (TE-R)	YES, TE-RA and TE-RB
Using < 1 Ah (.5Ah for Half Can)	YES
Using <= 28 V	YES
Using RF (If yes, list frequency and TX Power)	No
Using deployable?	YES, speed will be under 1"/s
Whole team consists of US Persons	YES
Using ITAR and/or Export Controlled hardware	YES

12.11 Budget

Budget Summary - 3/22/17			
Sections	**Budgeted**	**Spent**	**Remaining**
Motors	$1,312.50	$1,312.50	$0.00
Aluminum Stock	$527.12	$480.60	$46.52
Camera	$599.97	$599.97	$0.00
Purchased Hardware	$557.18	$702.71	-$145.53
PCB	$250.00	$0.00	$250.00
Manufacturing	$500.00	$0.00	$500.00
Electrical Components	$373.05	$435.34	-$62.29
Connectors	$180.00	$242.60	-$62.60
Microcontrollers	$55.00	$0.00	$55.00
Sensors	$301.00	$117.29	$183.71
Project Total	**$4,655.82**	**$3,891.01**	**$764.81**
1st Trip (4 Stdt 5 Days)*	**$4,356.75**	$0.00	$4,356.75
2nd Trip (4 Stdt 12 Days)*	$6,526.42	$0.00	$6,526.42
Deposit	$14,000.00	$8,000.00	$6,000.00
Total	**$29,538.99**	**$11,891.01**	**$17,647.98**
*Travel Costs Estimated using Travelocity, does not include food costs			

13 Glossary

Glossary

AIAA American Institute of Aeronautics and Astronautics 3, 6, 115

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. 22, 31, 33, 115

apogee The point at which the rocket has finished its ascent and payloads are allowed to deploy. 20, 69, 115

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. 115

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. 19, 22, 115

binary string An ordered sequence of 1's and 0's. 19, 115

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. 115

configuration space The Configuration Space (or C-Space) is a 4 dimensional space with a mapping to 3D real space that is used to represent the possible configurations of the arm's motors. 115

degree-of-freedom The directions in which independent motion can occur. In the case of the Hephaestus arm, there are four degrees of freedom. 3, 115

deployable Any portion of the payload that is expanded from its original configuration once in a space-like environment. 115

EEPROM Electrically erasable programmable read-only memory. A type of persistent storage for small amounts of data. 22, 115

GUI Graphical User Interface 20, 73, 115

matplotlib A Python library for drawing and manipulating graphs. 20, 115

microgravity An environment where the effect of gravity is significantly less than earth's. 3, 4, 115

npm npm is a package manager for NodeJS, a javascript library. It is used to install various 'npm' packages for NodeJS servers. 33, 115

OBC On-Board Computer 16–19, 115

on-board computer The microcontroller responsible for the control and experiment systems on the payload. 3, 22, 76, 115

OSU Oregon State University 1, 3, 4, 73, 115

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, 3, 4, 19–22, 30, 31, 35, 39–41, 44, 73, 79, 115

plot An interactive window generated by the Python library matplotlib to display some dataset on an x and y axis. 20, 115

port To transfer software from one system or machine to another. 31, 115

PSAS Portland State Aerospace Society 31, 115

PSU Portland State University 31, 115

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. 115

TE-1 The TE-1 line is an electrical input to the system that enables at a predetermined time during flight. 115

TE-R The TE-R line is a redundant electrical input to the system that enables at a predetermined time during flight. 115

WFF Wallops Flight Facility 3, 115