Technical Review And Implementation Plan For RockSat-X Payload - Hephaestus

Helena Bales, Amber Horvath, and Michael Humphrey

CS461 - Fall 2016

November 13, 2016

Abstract

The Oregon State University (OSU) RockSat-X team shall be name Hephaestus. The possible methods for implementing our project requirements 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. This document will focus on the implementation of the software, but shall include necessary project context including hardware.

Approved By _____ Date _____

Approved By _____ Date _____

Approved By _____ Date _____

Approved By _____ Date _____

Contents

1 Introduction

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.

1.2 Role Breakdown

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

1.2.1 Helena Bales

1. Target Generation

2. Arm Movement

3. Arm Position Tracking

1.2.2 Amber Horvath

1. Emergency Payload Expulsion

2. Program Modes of Operation

3. Target Success Sensors

1.2.3 Michael Humphrey

1. Telemetry

2. Video Camera

3. Data Visualization and Processing

2 Technologies

2.1 Target Generation

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.

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.

2.2 Arm Movement

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.

2.2.2 Proposed Solutions

1. **The movement of the arm shall be tracked through the position of the tip of the arm and the movement of the motor**. 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 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.

2.3 Arm Position Tracking

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.

2.3.2 Proposed Solutions

2.4 Emergency Payload Expulsion

2.4.1 Requirement Overview

2.4.2 Proposed Solutions

2.5 Program Modes of Operation

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.

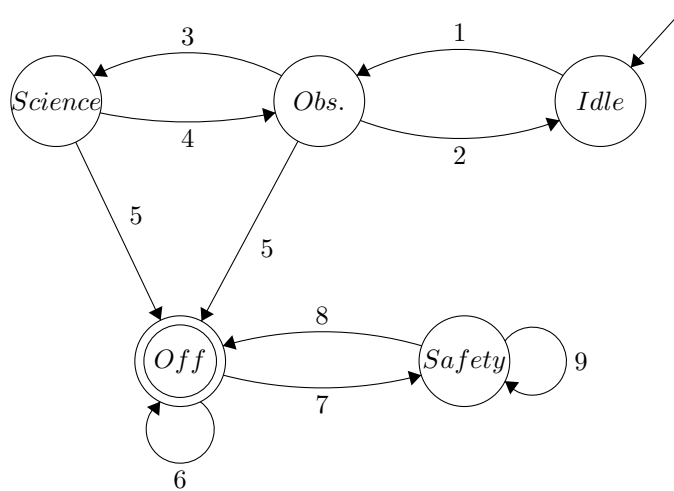### 2.5.2 Proposed Solutions

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.

2.6   Target Success Sensors

2.6.1   Requirement Overview

2.6.2   Proposed Solutions

2.7   Telemetry

2.7.1   Requirement Overview

The software shall report via telemetry all the target points it generates, as defined in section 2.1. The software shall also report which code branch it takes to facilitate debugging and post-mortem analysis, if necessary.

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 architecture of the payload.

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 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**

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.

The best option for now appears to be adopting Open MCT for the purposes of this mission.

2.8 Video Handling

2.8.1 Requirement Overview

The software shall be responsible for controlling the camera output.

2.8.2 Proposed Solutions

The three options being considered for controlling the camera are:

1. **Enabling and disabling third-party camera.**

2. **Enabling/disabling on-board camera, and storing video output.**

3. **Option 3.**

2.9 Data Visualization and Processing

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.

2.9.2 Proposed Solutions

The three options being considered for visualizing the data are:

1. **Matplotlib.** Matplotlib is a plotting library with a Python API. Matplotlib is written in Python, and will therefore run on all major platforms. 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 they 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 access 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.

3 Conclusion

4 Glossary

5 Appendix

5.1 Mission Patch

Figure 1: Mission Logo [1]

5.2 Project Overview

The Hephaestus project is a Capstone Senior Design project for Oregon State University's 2016/2017 Senior Design class (CS461-CS463). The CS senior design project is one part of the overall Hephaestus project. In addition to the CS team, there is one team of Electrical Engineers and two teams of Mechanical Engineers working on this project through other senior design classes. The Hephaestus payload is a rocketry payload developed as part of the 2016/2017 RockSat-X program. The RockSat-X program is a year long program where groups of students develop rocketry payloads with

the help of the Colorado Space Grant Consortium and Wallops Flight Facility. The term "rocketry payload" refers to an experiment inside a section of the rocket. Each section of the rocket is called a can, and is a standard space that we can fill with an experiment. The Hephaestus payload shall take up half a can and shall be mounted on a standard base plate provided by Wallops. We, as the Hephaestus team, will create the hardware and software for the payload, then integrate it into the rocket before launch.

5.2.1   Project Phases

The project shall include several phases. The first is the design phase. The design phase shall last all of Fall 2016 term at OSU. In the design phase, we shall design the robotics, electronics, materials, and software. The design phase shall include presentations to the RockSat-X program, where there will review our designs. Following the design phase will be the implementation phase. In the implementation phase we shall last through June 2017. This phase shall include testing of the payload. We will perform testing both at OSU and at Wallops. At OSU we will be testing the payload functionality. At Wallops, we will be testing the structural integrity of the payload, as well as its resistance to vibrations, heat, and cold. Following the implementation phase will be the integration phase. This phase will occur at Wallops in July. This is the point at which our base plate will be integrated into the rocket as a whole, along with the other participating teams. The final phase will be launch. Launch will occur in Summer of 2017. The rocket shall be launched from Wallops Flight Facility. During the flight we shall send telemetry to the ground station at Wallops. The payload shall perform the experiment once it reaches appogee. The payload will hopefully be recovered post-flight.
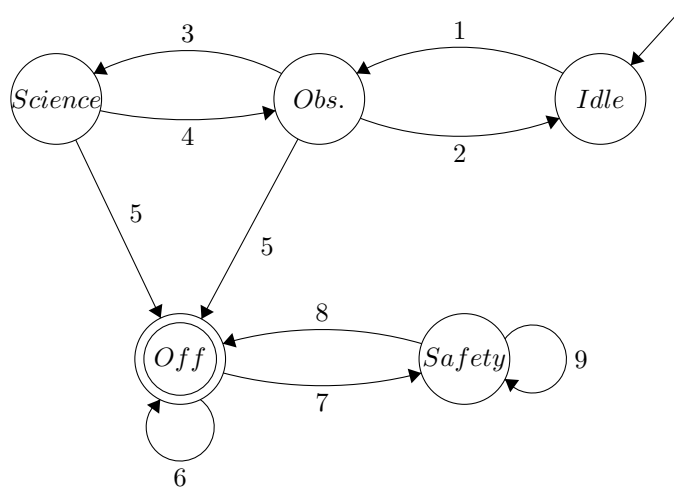
5.3   Software State Diagram

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: 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.

5.4   Model of Payload Hardware

5.5   Payload Wiring Diagram

5.6   References

[1] TODO Who made the Logo for us? [2] H. Bales and M. Humphrey, "Diagram of Software Modes of Operation," 2016. [Online]. Available: Hephaestus Requirements Document.