CS Capstone Team 12 - USLI
Design Document

Leif Tsang, Donald "Trey" Elkins, Ryan Wallerius

November 2018

Abstract

The purpose of this document is to explain and enumerate the various systems that the CS team is required to implement for competition in the 2019 NASA University Student Launch Initiative. Covered systems include the avionics and telemetry unit for rocket flight, the robotic soil-collecting payload, and the team website for front facing documentation and digital presence. This document will cover the functional components and design rationale for each system, and team plans for moving forward with a series of competition-worthy software products.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to define systems in order to give the team an overall understanding of the design of each component. This will provide our sub-team and entirety of the USLI team insight into the topics covered throughout the implementation of the project.

1.2 Scope

Smaller subsystems included in Avionics, Payload, and the Website will be utilized to compete in a competition hosted by NASA. The mission consists of a single-state solid propulsion rocket being launched to a target altitude of about one mile. At apogee, this rocket will separate into 2 sections and safely touch down on the ground using an integrated recovery system. After landing, the aft section of the rocket will eject a robotic payload that must move at least 10 ft. from the rocket body, collect a soil sample with a minimum volume of 10 ml, and store the sample in a sealed container on-board the payload. Once collected, the sample will be taken to a base station for analysis. The base station analysis is not a part of the NASA competition, but the team has decided to pursue it to add scientific value to our entry. Furthermore, the team must develop and display a robust web and social media presence to deliver technical documents to NASA and display educational outreach and team information.

1.3 Context

The mission will utilize programming languages and software utilities to provide commands and control for the hardware devices - namely micro-controllers, such as the Teensy 3.6 - that will comprise the computational systems integrated into the launch vehicle and robotic payload. Furthermore, we will use ReactJS to design and build a website that is capable of delivering technical documents and technical information about the team to NASA and other observers. The usage of these hardware and software components combined will provide the necessary modules and deliverables for execution of the team's entry for the 2019 NASA Student Launch Initiative Competition.

1.4 Glossary

- **ATU** - Avionics Telemetry Unit, a series of micro-controllers, sensors, and transceivers that collects, transmits, receives, and logs data during rocket flight.
- **CV** - Computer Vision.
- **NMEA** - National Marine Electronics Association, a standardized format for positional data that is widely used by GPS and other positioning systems.
- **PLEC** - Payload Ejection Control, receives signal from the ATU ground station to fire off black powder charges and eject the robotic payload from the rover aft.
- **ReactJS** - A JavaScript library to provides user interfaces through small scripts called components.
- **USLI** - University Student Launch Initiative, a competition orchestrated yearly by NASA in which university teams compete to build, launch, and operate a rocket and scientific payload. Frequently used to reference the OSU competition team itself.

2 AVIONICS

The Avionics Telemetry Unit (ATU) is responsible for the generation, transmission, reception, logging, and plotting of positional and telemetry while the rocket is in flight. The ATU assists in recovery of the rocket components after launch and collects data which will be analyzed via software (MATLAB and/or Python scripts) after the flight to extrapolate further launch data. The avionics system is composed of a ground station and a flight unit that communicate wirelessly using a custom packet system during the flight. The flight units collect and transmit the avionics data to the ground station during the flight. The ground system is capable of transmitting flight data to a computer that will output the information to a GUI with various plots and information. The ground station is also responsible for sending an ejection system to the Payload Ejection System (PLEC) once the rocket has landed and is ready to eject the robotic payload from the aft section. Both systems have similar but slightly different code. Modularization and functional decomposition of the code for both systems facilitates ease of development and adaptation.

2.1 System Design

The avionics system is composed of a ground station and multiple flight units that are placed within the rocket. Each has its own hardware and software components, and the system as a whole works together to ensure the mission profile is met successfully.

2.1.1 Flight ATU

The Flight ATU is composed of a GPS module, Teensy 3.6 ARM micro-controller, XBee Pro S3 900 MHz transceiver, TI CC 1200 433 MHz transceiver, and an antenna which are all embedded and connected on a custom PCB created by the team's electrical subteam. The GPS module receives positional data from the Global Positioning System which it then transmits to the Teensy as NMEA (National Marine Electronics Association) formatted packets via serial communication. The Teensy writes all of the given NMEA-format packets to the on board SD card, then uses a C/C++ program to discard all non $GPRMC-type packets, strips unecessary information out of the $GPRMC packets, and packages them into a new packet format which is sent over serial to the XBee Pro or TI transceiver modules. The custom packet structure has the following information (and more) added to it:

- Custom start delimiter - 0x7E
- Least and most significant bits for delimiting at the beginning of the packet
- Frame type (TX/RX Request)
- Frame ID Number
- 64-bit Target Address (currently hard coded)
- Reserved bytes
- Transmit options
- Checksum

2.1.2 Ground Station ATU

The ground station ATU is very similar to the flight ATU, but boasts a different set of code on the Teensy, a larger antenna, and the capability to connect to a computer via serial (USB). The XBee Pro and TI CC modules receive the packets from the flight ATU and sends them via serial to the Teensy (which runs a C/C++ program). If the Teensy detects a packet in

the serial buffer, it loads the packet, unpacks it, determines which ATU (of the 2 flight units) it came from, then pipes the data out over the serial interface with the computer.

2.1.3 Output GUI

A Python script on the connected computer receives the data, disentangles it, plots it over the serial monitor, then plots the data in real time over two graphs using MatPlotLib. By the end of the project, these plots will be wrapped in a user friendly application with a GUI that is usable by non-computer science or programming familiar individuals in order to make sense of the telemetry data and plot the positions of both the aft and fore sections of the rocket. The application will also have a button for triggering the PLEC once the aft section of the rocket body is at rest, as determined by the GPS data coming from the ground station ATU.

2.2 Design Rationale

A majority of the design decisions made for the avionics system come from decisions made during last year's competition by Chris Snyder - the electrical engineer in charge of the avionics system - and Mark Bereza - the software engineer in charge of the output GUI who also worked on the software side of the avionics. This year, the goal is to modify the embedded avionics code to be at least 25% shorter, correctly commented, and functionally decomposed (rather than 2-3 large functions, as it was last year). We also want the Python GUI to be upgraded from two plots and a serial display to a user-friendly windowed application with interactive buttons.

2.2.1 Micro-controller

The Teensy 3.6 was determined to be the most effective micro-controller for both ATUs due to its speed, size, and ease of programming. The Teensy 3.6 has a clock speed of 180 MHz - more than 11 times faster than the 16 MHz processor on a standard Arduino Uno. This makes the Teensy superiorly suited for high speed embedded applications with significant data throughput. The Teensy lacks many of the components that many other consumer micro-controllers are shipped with - ex. non-USB power input, large LEDs, heatsinks, pin headers, etc. - giving it an advantageously small power and spacial profile. Support for serial communication and the ability to write to an on board SD card are also critical functionalities that the Teensy offers. A micro-controller was chosen over a small embedded PC such as a BeagleBone or a Raspberry Pi due to the detrimental effects that software overhead that such a system inflicts on the data transmission and receiving process.

2.2.2 GPS

For the GPS unit, the unit from last year will most likely be recycled as it has all of the functionality necessary and using a new device necessitates the creation of a new custom PCB to integrate the system. Depending on the resources and bandwidth available, the GPS unit may be replaced with a unit that another OSU rocket team is using, as the alternative unit also supports usage of the Russian GLONASS positioning system while the current circuit only works with American GPS.

2.2.3 Transceivers

The XBee Pro S3 900 MHz transceiver is the current de facto Rx/Tx device in the system due to ease of implementation with pre-existing libraries from the manufacturer, the ability to transmit on the 900 MHz band without a commercial license, and the relative power and range that a 900 MHz transmission band offers. The current 900 MHz system is capable of and tested for reliable packet transmission within a 2 mile line of sight distance at 250 mW of power on a high gain antennae; transmission will be at either 210 mW or 250 mW depending on the other transceiver system's power output to keep our signal emission within a reasonable operational range. The competition rules stipulate a combined outgoing signal power of 250 mW as the cap for operation.

Unfortunately, the 900 MHz band is prone to interference due to its popularity as a license-free consumer radio frequency. The USLI team experienced technical difficulties last year due to the presence of multiple other higher powered RF transceivers operating on the 900 MHz band at the competition. Since the launch vehicles must be set up and wait for a given duration of time, the 250 mW transmission was easily overpowered and data integrity was significantly impeded.

To combat the problem of interference on the 900 MHz band, the primary goal for the avionics system this year is to shift over to transmission on the 433 MHz band. 433 MHz transmission offers relatively high data fidelity, a range encompassing the expected flight and landing distance of the rocket, and its infrequent use due to its restricted band status makes it highly unlikely that any interference will occur from other devices on the same frequency. Unfortunately, 433 MHz transmission requires a HAM radio license to operate and provides lower (though still high) data fidelity than 900 MHz transmission.

The main drawback to using 433 MHz transmission is the lack of modular and ready to use consumer grade radio frequency devices. The new system will utilize a Texas Instruments CC 1200 transceiver that lacks any kind of API or libraries. As such, it will be the software team's responsibility to replace the currently implemented XBee API with custom code to configure registers, transmission, and reception of 433 MHz signals. TI has a publicly available software suite known as 'SmartRF Studio' that's designed to configure integrated devices such as the CC 1200 and can be used to generate and export low level C code that can then be added to the embedded code on the ATU's Teensy on both the ground and flight units. The primary goal for the transceiver system is to use SmartRF Studio to generate code that will allow the XBee Pro S3 to be effectively and functionally replaced by the CC 1200. This goal renders refactoring, decomposition, and code modularity incredibly important in order for the system to be as flexible and adaptable as necessary to fit the mission profile.

2.2.4 Programming Languages

AVR compatible C and C++ are the languages of choice for the Teensy and embedded control code; no other high level languages are easily implemented with the given micro-controllers, and AVR assembly is too difficult to reasonably implement without tackling a steep learning curve that the team doesn't have time to address. Furthermore, C and C++ have their own well fleshed-out standard libraries and are compatible with Arduino libraries, the XBee Pro libraries, and the code that can be generated and exported by SmartRF Studio. Python is the language of choice for data analysis and the ground station ATU GUI due to its portability, ease of development (so volunteers can work on the system as well), and shallow learning curve. Several people on the team already know Python, which makes it a preferable alternative to other

scripting languages or MATLAB. Additionally, the MatPlotLib library among others makes data analysis and plotting easy. An effort will be made to convert old MATLAB data processing scripts (for the data from the SD cards on the flight ATUs) into Python scripts with operating times of a few minutes maximum instead of the hours of operation that the MATLAB scripts potentially required in the past.

2.3 State & Class Diagrams

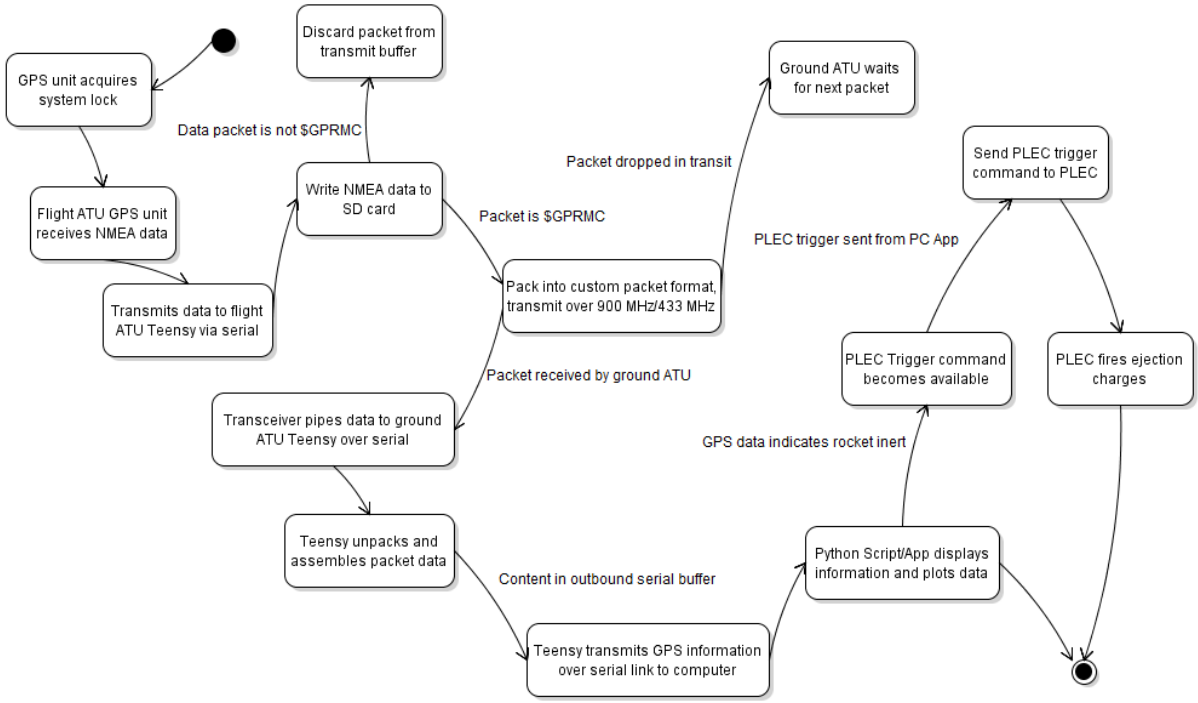

Fig. 1. Avionics State Diagram

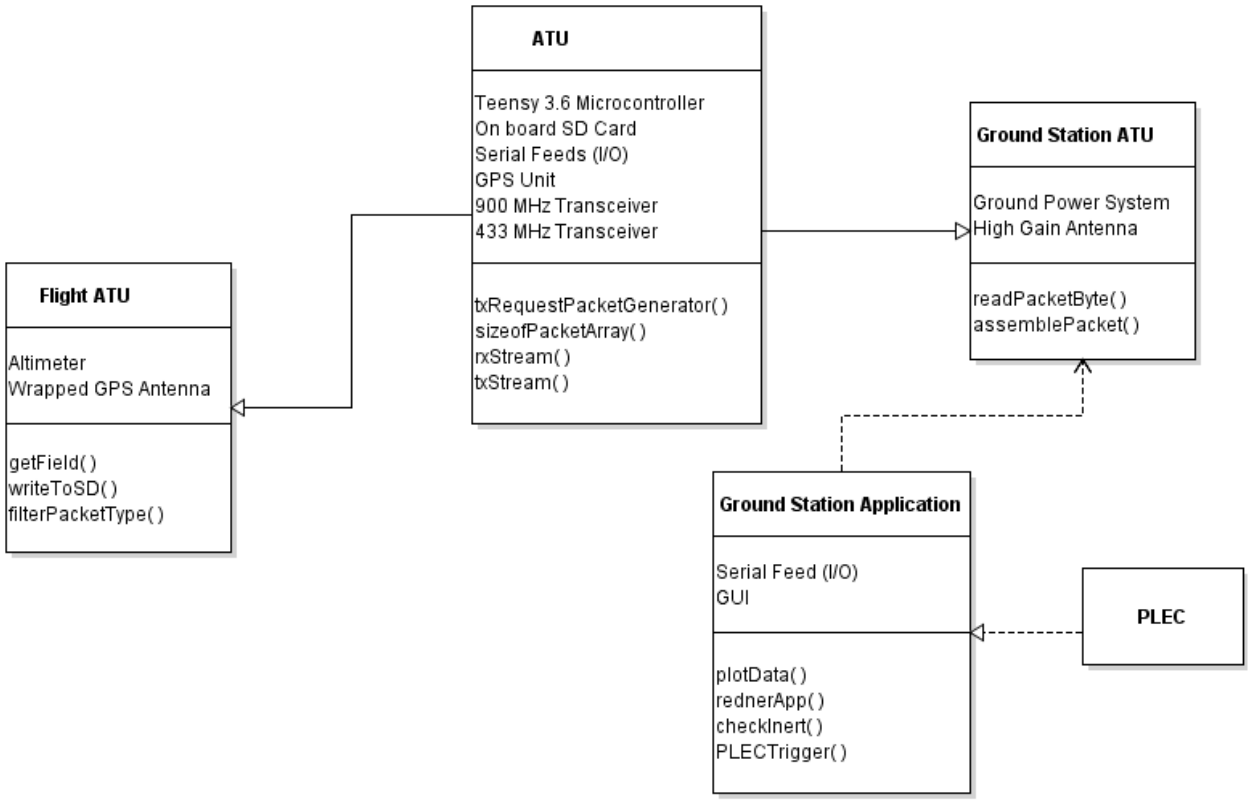

Fig. 2. Avionics Gantt Chart

2.4 Gantt Chart

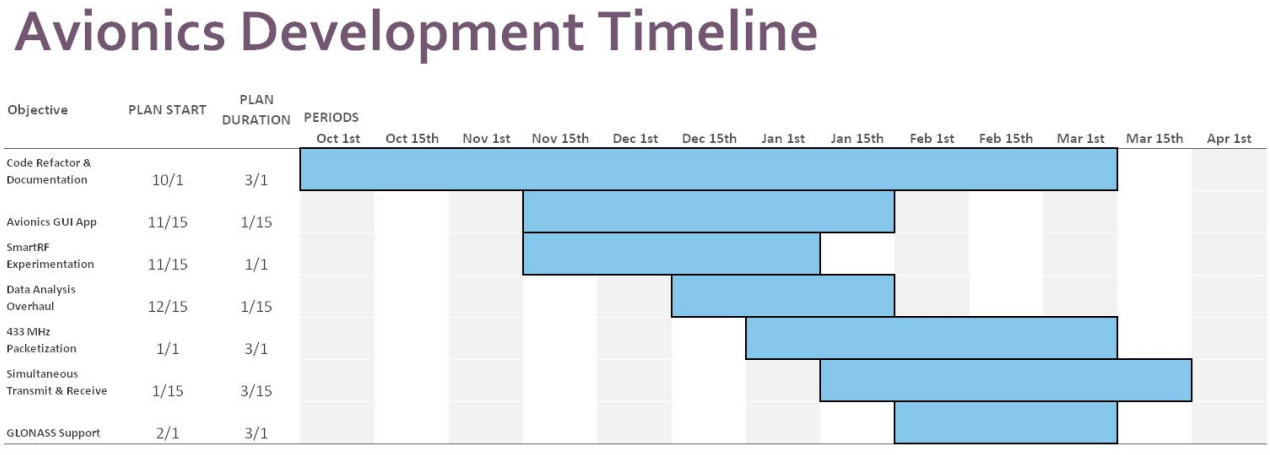

Fig. 3. Avionics Class Diagram - WIP

3 PAYLOAD

The payload will be responsible for a large majority of the mission after it has been ejected from the rocket body. All tasks must be completed while remaining completely autonomous. The tasks that must be completed are the following:

- Travel at least 10 meters from any part of the rocket
- Find a desirable location to collect a sample
- Activate auger to collect soil sample in a sealed container
- Receive GPS base station coordinates
- Travel to the GPS coordinates sent to the rover
- Doc to deliver soil sample that will be released from the rover.
- Avoid any objects that could potentially pose a threat to the rovers mobility

In order for the completion of the above tasks, this rover must be pre-programmed with a complex set of instructions to follow. The rover will continuously be taking in data from the environment using a camera and sonar sensors to check for obstructions and have the required intelligence to successfully avoid the obstacle and remain on track to the destination. The rover is equipped with a GPS in order to guide the robot away from the rocket body to achieve the minimum distance from the rocket body as well as guide the rover to the base station in order to deposit the samples. A gyro will let the rover know if the area being drilled at is feasible to obtain a soil sample and if the gyro is with in operating range for the auger. The motors must be capable of being fine tuned in order to quickly maneuver around and over terrain. Lastly, this rover must have a RF receiver to receive GPS coordinates from the rocket body and the base station for navigation.

3.1 Design Viewpoints

During the design of this payload, I took into account the practicality of the design. This program run on the robot will need to do every function autonomously, so there is no room for error. As such it has been designed with care, in order to not to become stuck in any specific process with every situation accounted for. For this reason, there exists an extra computer attached which will be in charge of computer vision. This computer will be able to verify the environment in front of the rover utilizing object detection establishing redundancy with the sonars. With this we will be able to avoid any obstacles such as the rocket body and even doc the rover to deliver the soil sample signaling the end of its mission.

3.2 State Diagram

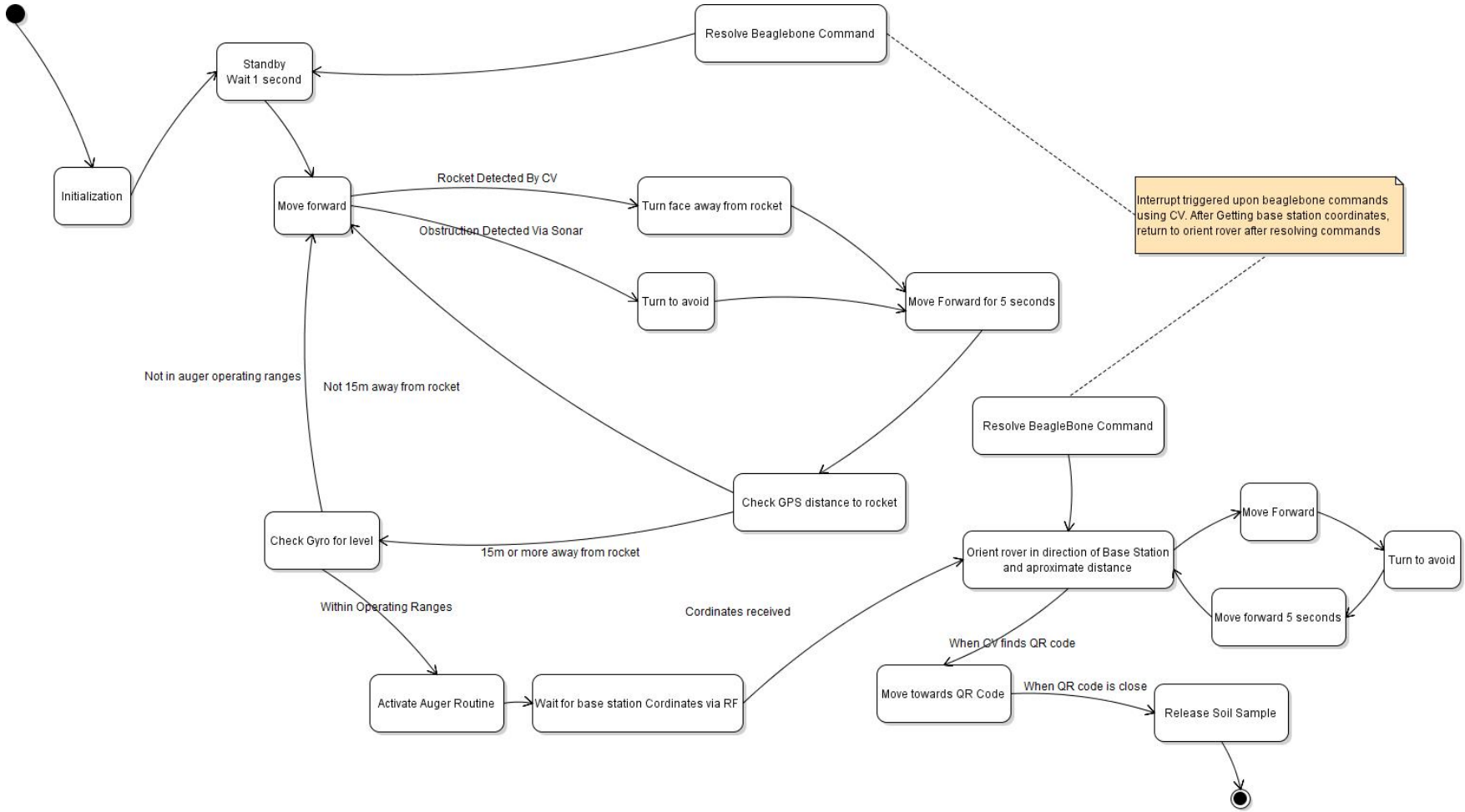

Fig. 4. Rover State Diagram

The state diagram shown above in figure 1 begins after the payload is ejected from the rocket body. The rover begins by initializing which is when the rover obtains the GPS coordinates of the rocket. The rover will then start phase 1 of its programming. This involves the rover moving forward until it is at least 10 meters away from the rocket and then finds a level spot to collect a soil sample. After the soil sample has been collected, the rover freezes and waits for a GPS coordinate that will begin phase 2. During phase 2, the rover will begin to deliver the soil sample to the base station. The rover will travel in the direction of the base station, while continuously avoiding obstacles. After the rover is in close proximity of the base station, it will begin searching for the base station using CV. Eventually, a QR code will be found in the base station that will trigger the rover to dump the soil sample to be processed by the technology located at the base station. This will signal the end of the rovers mission. During the rovers operation, we show redundancy by implementing

both computer vision as well as basic sonar capability on the Teensy for object detection. This will decrease chances for the rover to become stuck and unable to complete its mission. Each process for the rover will be extensively tested on both a test rover as well as the final rover after it has been built.

3.3 Class Diagram

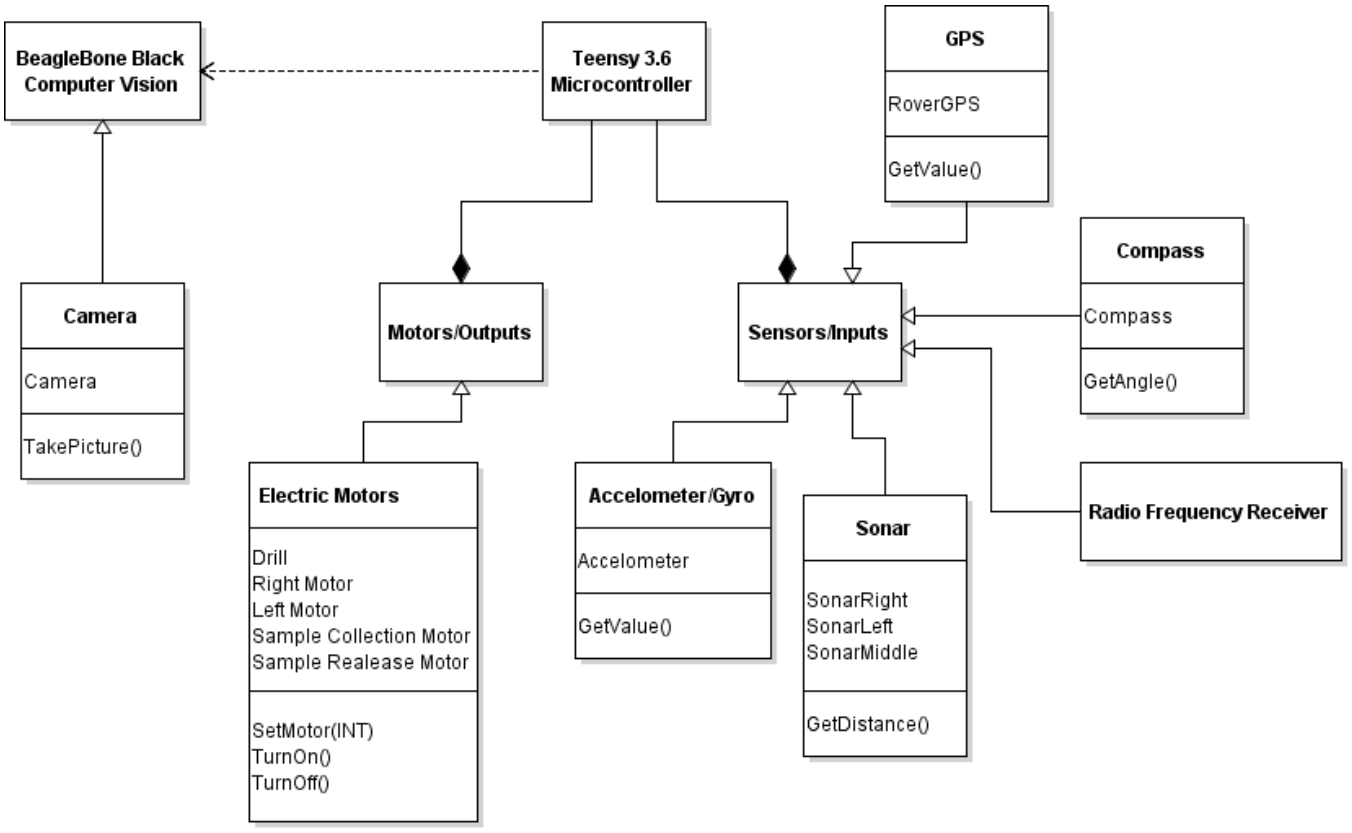

Fig. 5. Rover Class Diagram

This class diagram shows the relation of the different components and sensors to the Teensy micro controller and

Beaglebone computer. It does a good job of showing what objects will be created and what commands may be used to interact with each object. These objects will be implemented with the API's for controlling each piece of hardware on a lower level. This will help organize our code into a modular arrangement where devices can be called as they are needed for the rovers programming. The Beaglebone is connected to the Teensy as it will be sending interrupts to call specific routines that will be pre programmed onto the Teensy. Connected to the Beaglebone is a camera that will be able to take a picture of the area in front of the rover throughout program.

3.4 Gantt Chart

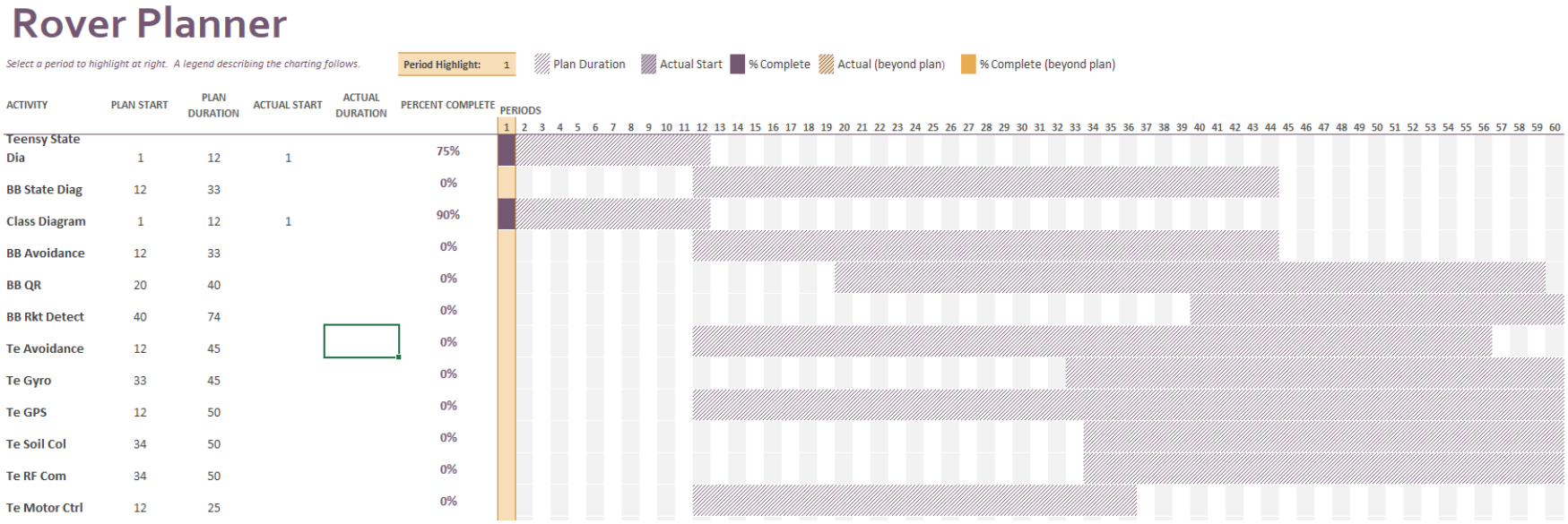

Fig. 6. Payload Gantt Chart

Each period will account for 1 day and the first period began on November 28, 2018

4 WEBSITE

The website is responsible for creating a central location to store critical documents for NASA to retrieve, create a social media presence for the OSU USLI team, and create a location that sponsors can easily access and contribute. In order for this to be created the the following will be implemented:

- React JS
- Redux
- Node.JS
- Webpack
- Browser Sync
- Babel
- Material UI

The reasoning to why will be discussed later. Although the website is not apart of the competition, creating a website is crucial to the team's success. After implementing design choices, the website will be aesthetically pleasing, easy to navigate, easy to access information about the team and creates a central spot for sponsors. React is also used because of the versatility with mobile development.

4.1 Technologies Implemented

4.1.1 ReactJS

ReactJS is an open-source Java-Script used to build user interfaces specifically for single page applications. The purpose for development was to handle the viewer layer for web and mobile applications. React allows developers to create large web applications which can change data, without reloading the page. React is also fast and simple. This allows our development team have the tools it needs in order to meet the goals set by the USLI team.

4.1.2 Redux

Redux is a state container for JavaScript applications. Redux helps developers write applications that will behave consistently, easy to test, and run in different environments such as client, server and native. We must have a website that has minimal downtime.

4.1.3 Node.JS

Node.js is a an open-source environment that executes JavaScript code outside of a browser. Developers use Node.js for client-side scripting. Node.js allows developers to run scripts server-side to produce web page content before the page is sent to the user's web browser. The reason our team is using Node.js on our website is because of the ability of not having to wait and can go on with multiple tasks. This also meets the teams goal of efficiency.

4.1.4 Webpack

Webpack is an open-source Javascript module bundler. The reason developers implement webpack is to bundle JS files for usage in a browser.

4.1.5 Browser Sync

Browser Sync will be a crucial tool the team uses for testing. One of the most important parts of development is testing. Browser Sync makes changes and testing faster by synchronizing file changes and interactions across many devices. This feature is called Live reloading. This makes the page auto-reload when changes in code is made. This will help our team because we have multiple developers accessing and developing the new website.

4.1.6 Babel

Babel is a JavaScript compiler that has the following features:

- Transform syntax
- Polyfill features missing in target environment
- Source code transformations.

This is another tool that will help our team during the development of this website.

4.1.7 Material UI

Material UI is one of the most used user interface libraries for React. Material UI has React components that implement Google's Material Design. The following reasons are why we are using Material UI:

- Deliver on fully encapsulated React components
- Customizable
- Cross browser compatible and accessible
- Ease of learning

All of these tools will give the team the ability to develop a website that will meet the goals set and contribute to the teams success.

4.2 User Use-Case Chart

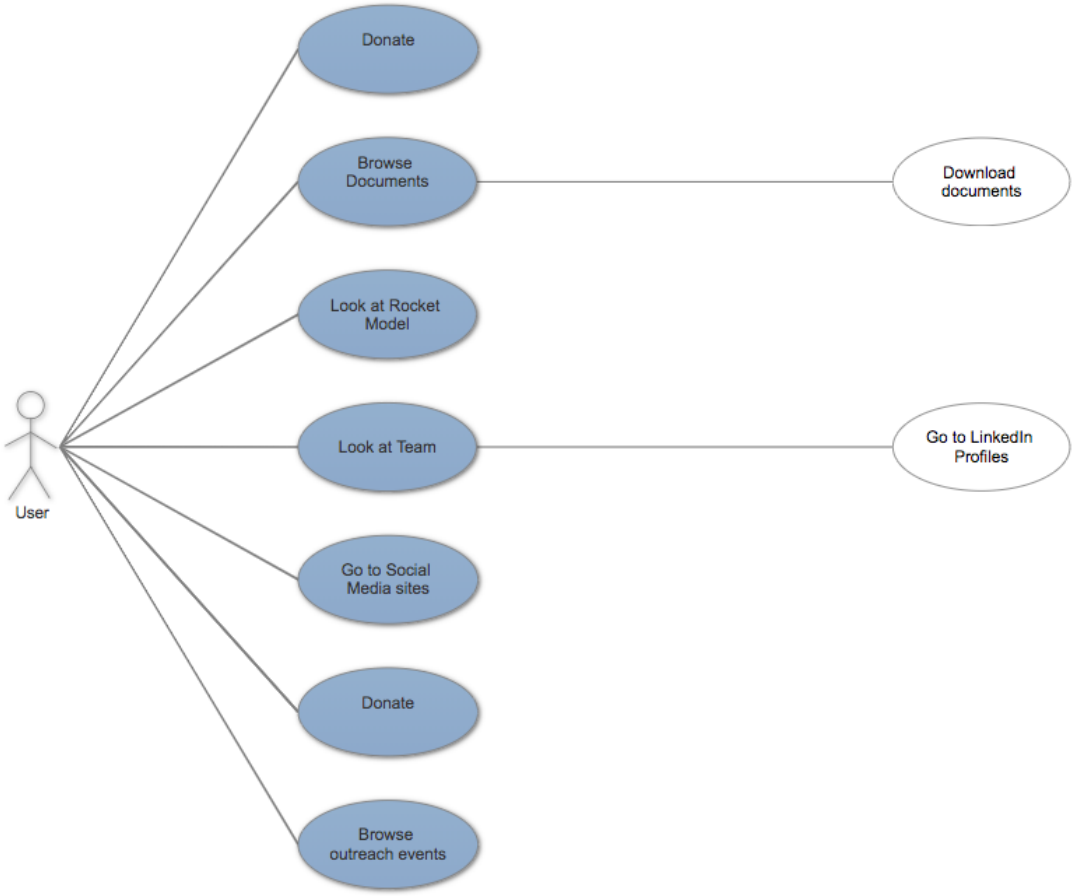

Fig. 7. Use Case - User

This figure shows what the user is able to do when they log onto our website. We will have an About us section, Documents section, Donate section, Outreach section and a sponsors section. NASA needs to be able to download our documents that we submit for the competition so there will be an option to download the pdf.

4.3 Admin Use-Case Chart

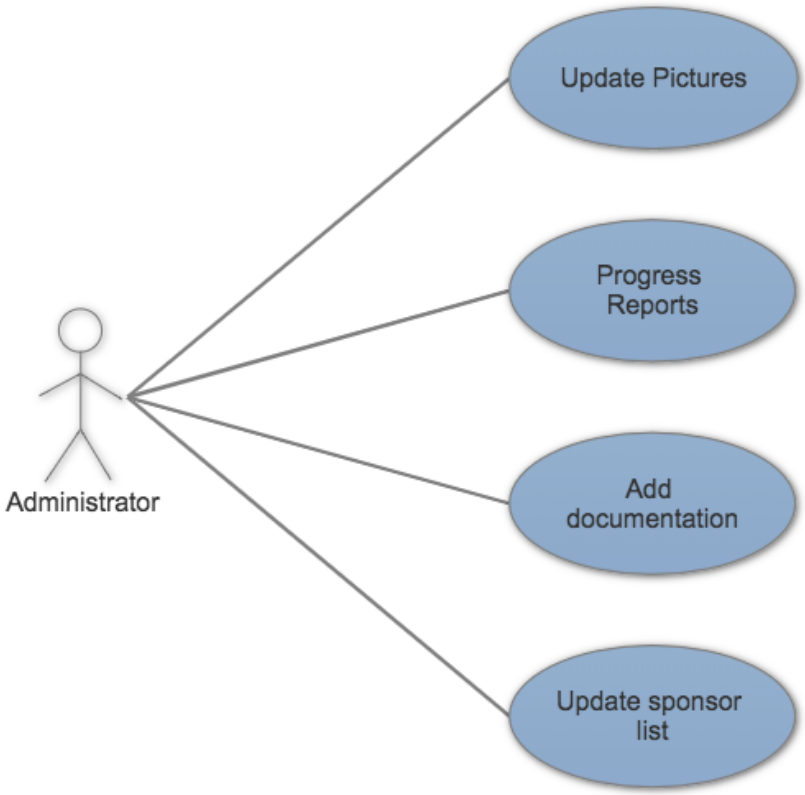

Fig. 8. Use Case - Admin

This figure shows the options that the admin of the website has. We will have pictures of the team up so they will be able to update the website with current pictures. Admins can also update our progress, add documentation needed for NASA and update the sponsor list.

4.4 Gantt Chart

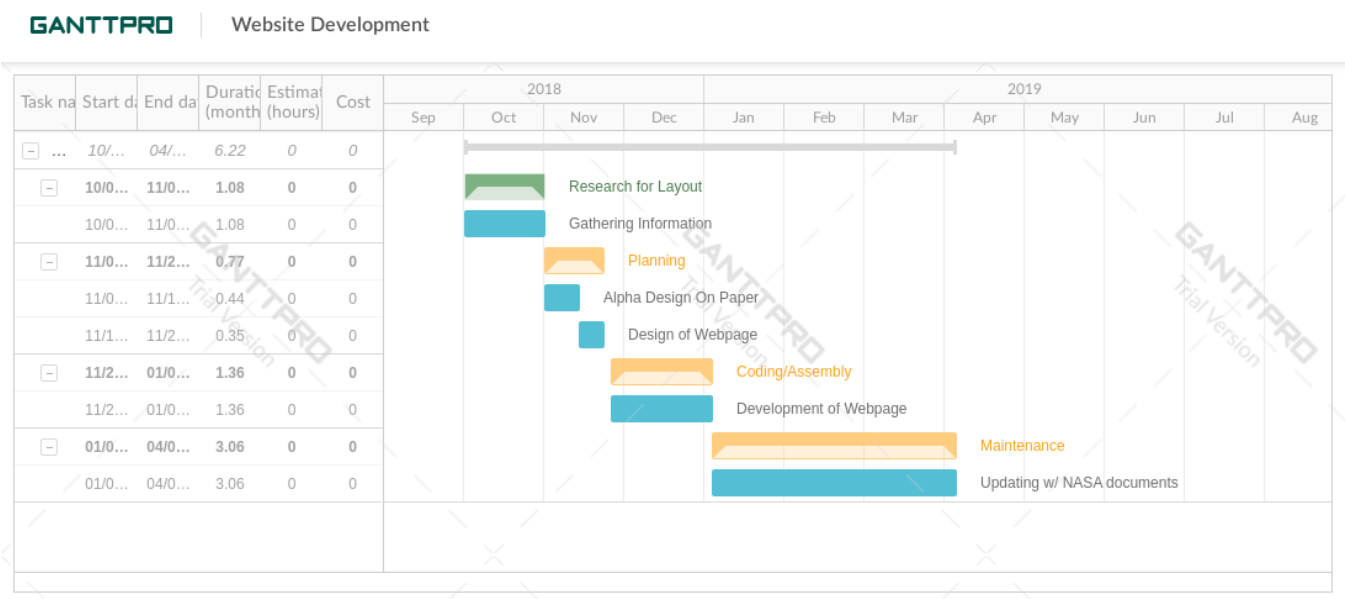

Fig. 9. Website Gantt Chart

This figure is an overview of the development of our website for the Oregon State USLI team. We must have a functional product by January 4th. This is when a critical document for NASA is due. After January 4th we will be continuing our development as well as maintenance of the website.

5 CONCLUSION

This document has has explained and summarized the avionics, payload, and website systems that the CS team is implementing for NASA's 2019 University Student Launch Initiative Competition. Key terms were explained, design components for each system were enumerated, and design rationale was laid out and expanded upon. The team must apportion the work appropriately, communicate effectively, and work together to deliver the required functional products for OSU's rocket launch at the NASA USLI competition in April 2019. Each system presents its own challenges and obstacles, but each system is critical to OSU's competition entry and potential victory. Care must be taken to ensure that each system is completed to requirements and expectations, and that the best products that team 12 can create are delivered to the USLI team.