The ARLISS Project
Progress Report
Senior Capstone

Steven Silvers, Paul Minner, Zhaolong Wu, Zachary DeVita
Capstone Group 27

2016-17 Academic Year

Abstract

This document details our experiences designing our project over the duration of the academic year. This includes progress made, problems encountered, and plans for the future. We also detail changes we plan to implement in the future.

Contents

I. Introduction

II. Original Requirements Document

SOFTWARE REQUIREMENTS SPECIFICATION

for

The ARLISS Project

Capstone Group 27
Version 1.0

Steven Silvers
Paul Minner
Zhaolong Wu
Zachary DeVita

Oregon State University
November 2, 2016

Abstract

This document details our plans to create software for a autonomously driven satellite, which will be launched from a rocket at 12,000 feet into the air. Upon safely landing, it must drive itself to a specific GPS location. Our plans include details about specific software modules which need to be implemented, as well as the environment our software must operate in.

TABLE OF CONTENTS

REVISION HISTORY

Name	Date	Reason For Changes	Version

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to provide a thorough and verbose description of our teams software which we will be developing for use in the ARLISS International Competition; the acronym ARLISS referring to ‘A Rocket Launch for International Student Satellites’. The document will explain the purpose and features of the system, as well as, objective of the system, and the constraints under which it must operate. This document is intended for any stakeholders in the project, i.e. clients, instructors, and engineering collaborators, as well as, for the software developers who will be creating the system.

1.2 Document Conventions

This document follows IEEE Format. Bold-faced text has been used to emphasize section and sub-section headings. Italicized text is used to label and recognize diagrams. Requirements will be prioritized into two categories; High-level requirements are those which are mandatory and will be specified throughout the document, while ”stretch-goals” will be denoted with a heading of ‘Secondary Requirements’.

1.3 Intended Audience & Reading Suggestions

This document is to be read by the software development team, the extended team of engineers who will be collaborating on this project, course instructors, and the client for the project. The document will be publicly accessible as well, and may be read by anyone viewing our exhibit. The document can, and will, be read by our peers in software development, as well as, engineers of all disciplines.

Overall Description – The general public, as well as, the media may find the overall description to be most pertinent, and to give an adequate and effective overview of the ARLISS Project.

System Features – Developers, as well as, the extended team of engineers who are collaborating on this project will require a comprehensive understanding of the system features in order to integrate their design, and, additionally, for testing purposes.

Nonfunctional/Functional Requirements – The computer scientists developing the software, and the electrical and mechanical engineers developing the hardware will need to utilize, and adhere to, the requirements.

1.4 Project Scope

Our team of computer scientists will be developing software to operate an autonomously navigated satellite to a specified set of coordinates during the international competition. Our team’s satellite will be ejected from a rocket at approximately 12,000’ AGL, and fall safely to the surface of Earth with the aid of a parachute. During the estimated 15 minute “hang time” that the rocket spends in the air, there will be barometric reading taken every 1000ft by the satellite. Once the satellite reaches the ground it must detach from the parachute and circumvent any potential restraint created by the canopy or suspension lines of the parachute. Once moving, the autonomous satellite must utilize its system of cameras and radar to detect obstacles so that it may navigate around them. GPS will be necessary for the satellite to find the coordinates.

The most imperative objective of the software is to safely guide the satellite to its intended destination. This is a requisite, but the secondary objective of taking barometric readings is an additional goal which our group is confident in completing.

2. OVERALL DESCRIPTION

2.1 Product Perspective

This is a new and entirely self contained product. The software is composed of multiple independent systems which will be used at different steps in the satellite’s mission. For example, once the satellite has been deployed from the rocket, the parachute deployment system will be used. These systems will be deployed in a specific order, and the timing of the deployment will be based on various sensor readings determining if the time is right to deploy the next system. The only user interface would be the graphing software used to display barometric readings obtained by the satellite during it’s descent. The software would interface with many hardware components, including cameras, thermometers, servos, and various sensors. These components will serve as the software’s eyes and ears. Memory and power will be constrained, so the software must be as memory and power efficient as possible.

2.2 Product Functions

The software has four main functions, which it performs at each stage of the mission. Each function is described in greater detail in section 3. Parachute deployment, change to drive mode, sensor and motor array initialization, and obstacle avoidance.

2.3 User Classes & Characteristics

The software as a whole is only useful in completing the specific challenges our specific satellite has to complete, although the individual parts of the software may be useful for other applications. Autonomous driving in particular is an upcoming field, so users who wish to create autonomous vehicles may find use from our obstacle avoidance system. Unfortunately, our obstacle avoidance system is designed specifically for navigating dry lake beds, so it's general use is limited.

2.4 Operating Environment

Since the mission is being performed only once in a specific location, the software will be optimized to work best in those conditions. The mission takes place in Black Rock Nevada, in a dry lake bed. This means the navigation software needs to deal with rocks, cracks, and tire tracks, but not much elevation change. Therefore, the navigation software will only work properly in locations similar to Black Rock Nevada. The rest of the software functions could work correctly in other conditions, nonetheless.

2.5 Design & Implementation Constraints

Space is a precious commodity on this satellite, therefore, the least amount of hardware possible will be fitted to it. This limits resources such as power, computation speed, and memory. This means the software will need to be implemented as simply and efficiently as possible while still being adequate to complete it's task. In addition, the autonomous navigation software must act quickly. If not, the satellite will get stuck, or hit an obstacle before the software can correct course to avoid it.

2.6 User Documentation

There will be a manual included with the satellite in PDF format, which details what each function of the software does, as well as it's limitations. For example, in the Obstacle Avoidance section, the manual will list the types of obstacles the software recognizes and avoids. In addition, the code itself will be well commented, so that future developers can understand what it is doing.

2.7 Assumptions & Dependencies

Since the satellite is being designed at the same time as the software which runs it, hardware specifics aren't known. We have a general idea of what the satellite should be when it's finished, but specifics such as the placement of sensors, or even which sensors will be included, isn't currently known. As the project progresses, more details will be revealed to us, and we should be able to adjust the software accordingly.

3. EXTERNAL INTERFACE REQUIREMENTS

3.1 User Interfaces

Because of the CanSat operates autonomously, the team has made the decision to not include a user-interface for the CanSat for the time being, but it is possible a user interface will be created in order to more easily test the satellite. The team evaluated all possible solutions based on the project goals, such as being cost-effective, efficient and weight/space constraints to decide a user-interface wasn't necessary.
A user-interface would, however, be useful during the testing stage. It could display the comprehensive live data that allows engineers to do quick analyze and have the full control of the CanSat.
A user-interface would also be a plus if we can display live data from various sensors during the final competition. Although since the CanSat operates autonomously, it is unnecessary to have a user-interface and will add on another communication layer. Due to the weight/space constraints, the team will evaluate the possibility of having a user-interface upon the completion date of the project.

3.2 Hardware Interfaces

Due to the fact that the mechanical and electrical teams are still under their early modeling and developing stages, we have not yet gotten any hardware specs. It is fairly straightforward to the team that there will be microprocessor(s), memory, buses, micro-controller(s), sensor(s), I/O device(s) and motor(s) involved, we just don't know the specifics. The hardware interfaces run in parallel with several electrical connections carrying parts of the data simultaneously.

3.3 Software Interfaces

This CanSat will be implemented in an object-oriented language. The team may have to design a custom operating system for the CanSat. Different classes will be created for each system of the software.

3.4 Communications Interfaces

A circuit board will perform most of the communication between the software and hardware. Communications interfaces will carry high level of intelligence and pre-defined protocols to achieve the hardware and software parallelism, to provide a secured platform for the CanSat to ensure its functionalities.

4. SYSTEM FEATURES

4.1 Parachute Deployment

A. Description & Priority

This system will detect when the satellite has been deployed from the rocket and activate the parachute module to carry the CanSat safely to the ground.

B. Stimulus/Response Sequences

This subsystem will respond to being released from the rocket, after-which it will deploy the parachute module.

C. Functional Requirements

This system just needs to activate the on-board parachute.

4.2 Transfer Control to Drive Mode

A. Description & Priority

This system will need to get the satellite ready to drive once it has landed on the ground. This involves activating various servos within the CanSat to expand the satellite from the payload container. In addition, the can shaped fairing will be shed from the satellite.

B. Stimulus/Response Sequences

Once the CanSat's sensors have determined that it has landed on the ground, the software will then activate this module.

C. Functional Requirements

This system needs to get the satellite out of the soda can container and expand the satellite's treads to the ground so that it is ready to start moving.

4.3 Sensor & Motor Array Initialization

A. Description & Priority

The driving sensors and motors need first time initialization before it can begin driving. This would include locking on to the target GPS coordinates, and preparing the sensors needed for the obstacle avoidance system.

B. Stimulus/Response Sequences

This process will begin the moment that the previous drive mode system completes its task.

C. Functional Requirements

The GPS location needs to be determined, and the sensors which will be used for the obstacle avoidance system need to be initialized.

4.4 Obstacle Avoidance System

A. Description & Priority

Once the satellite is locked on to the target GPS location, it will need to autonomously drive to the target location. This could be anywhere from less than a kilometer to fourteen kilometers depending on how the parachute is carried. There will be obstacles between where the satellite lands and the target location such as rocks and tire tracks that will need to be avoided by our satellite. This system will take input from the satellite's on-board sensors and use that data to control motor function so that the obstacles will be avoided.

B. Stimulus/Response Sequences

As soon as the sensor and motor array initialization completes, this system will begin. The system will be driving the satellite towards the target GPS location, unless it it is responding to a sensor detecting an obstacle. These obstacles include rocks, tire tracks, and other impassable terrain. The software will respond to the obstacle by finding an alternate route avoiding that obstacle.

4.5 Functional Requirements

Drive the satellite to the target GPS coordinates while avoiding any obstacles that impede the satellite's progress.

4.6 Performance Requirements

The speed at which this software reacts is of crucial importance for the obstacle avoidance system. If the software doesn't react to it's surroundings fast enough, the satellite will hit obstacles before it has a chance to avoid them. Therefore, the software must be able to react to obstacles faster than the satellite can move.

4.7 Safety Requirements

This software is entirely safe. The satellite is unable to do any kind of damage to any person or object in it's vicinity.

4.8 Security Requirements

Security is not a concern for this software. The satellite on which this software is running will only be deployed once, and the software itself is very specific for a single task.

4.9 Software Quality Attributes

This software must be thoroughly tested to ensure it will be able to complete it's task on the first try. Therefore, the software will be tested both by simulating the environment it will be running in, and using the actual satellite in environments like where it will be deployed. The simulations will be used to ensure the software behaves as inteded when faced with obstacles such as rocks and tire tracks, while the satellite tests will ensure the software can detect real obstacles using the provided sensors on the satellite.

4.10 Business Rules

Each system in the software acts independently from the other systems. The parachute deployment system must activate first, followed by the drive mode, sensor and motor array initialization, and obstacle avoidance system. Each system has a specific task to perform, then it ends and a new system begins.

5. SECONDARY REQUIREMENTS

5.1 Data Visualization

A stretch goal for this project would be to develop a system to visualize data sent back from the satellite to a remote control station. This would include data from the satellite's flight, as well as its trip across the desert.

5.2 Appendix A: Glossary

AGL: Stands for ”above ground level”

ARLISS: A collaborative effort between multiple institutions to build, launch, test and recover prototype satellites miniaturized to fit inside a soft drink can.

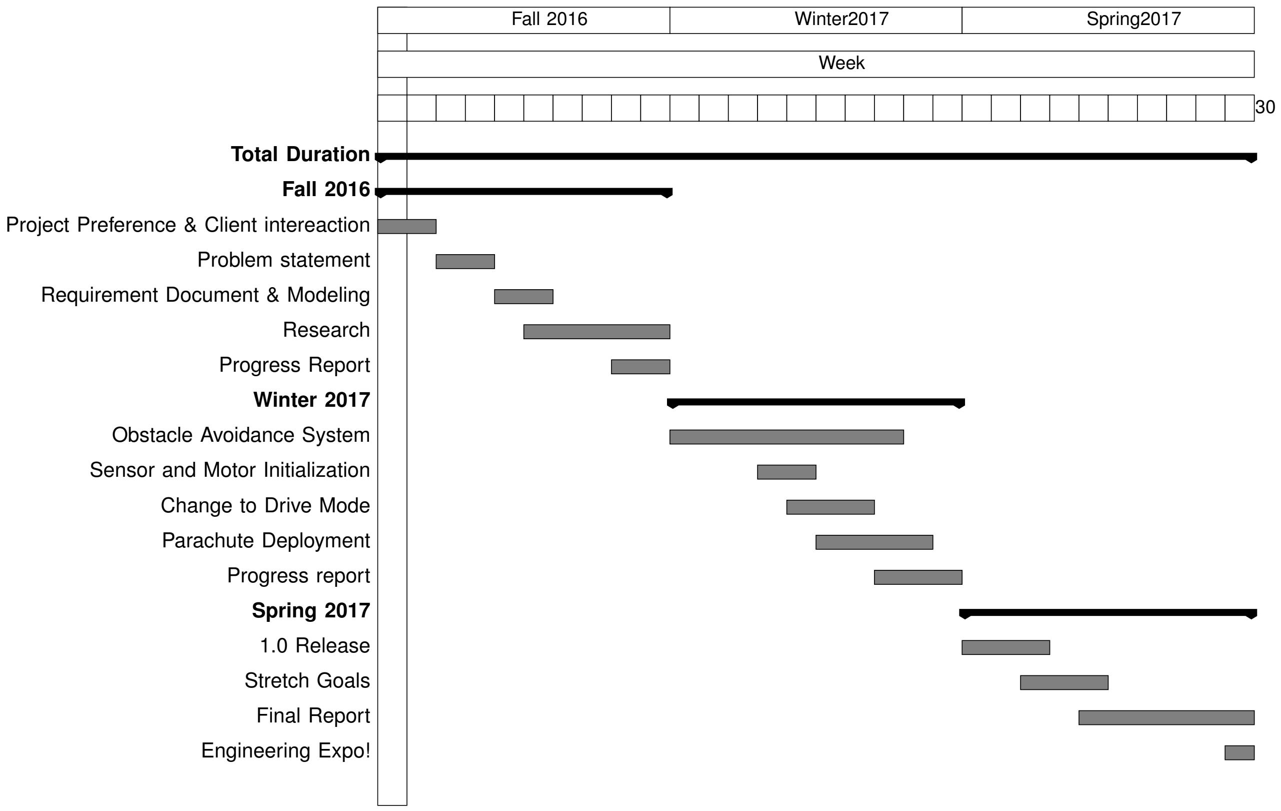
Autonomous: Used to refer to the self-driving satellite. The satellite must navigate to a GPS destination without any human intervention.

CanSat: Name given to the prototype satellites used in the ARLISS competition.

Destination: The location specified by the event organizers which the satellite must navigate to after being launched from the rocket.

Obstacle: Any object which may impede the satellite's progress to it's destination. This includes rocks, tire tracks, plants, and uneven terrain.

Satellite: Referred to also as ”CanSat” this is the physical device that our software system will be loaded on.


Fall 2016
Winter2017
Spring2017
Week
30
Total Duration
Fall 2016
Project Preference & Client intereaction
Problem statement
Requirement Document & Modeling
Research
Progress Report
Winter 2017
Obstacle Avoidance System
Sensor and Motor Initialization
Change to Drive Mode
Parachute Deployment
Progress report
Spring 2017
1.0 Release
Stretch Goals
Final Report
Engineering Expo!

Steven Silvers	**Zhaolong Wu**
Signature	*Signature*
Date	*Date*
Paul Minner	**Zachary DeVita**
Signature	*Signature*
Date	*Date*
Nancy Squires	
Signature	
Date	

III. Changes to Project

IV. Original Design Document

The ARLISS Project
Design Document
CS 461

Steven Silvers, Paul Minner, Zhaolong Wu, Zachary DeVita
Capstone Group 27, Fall 2016

Abstract

This document defines the plan for how the various pieces of the ARLISS Project will be developed and implemented. Each piece that was discussed in the previous technology review will be gone through in detail in its own section.

Contents

I. Introduction

In this document we have divided our project into twelve components, and a framework for each one of these components has been designed by the corresponding member from the list below. Each section will detail how a component will operate, and how it will interact with other related components. This document is intended to provide a structure, and the architectural layout for the entire project.

Project Framework
 Zachary DeVita
CMOS Image Sensing
 Zachary DeVita
Ultrasonic Radar Sensor
 Zachary DeVita
Obstacle Avoidance
 Steven Silvers
Control Board
 Steven Silvers
Motorized Tracks
 Steven Silvers
Parachute Deployment
 Paul Minner
Getting Unstuck from Obstacles
 Paul Minner
Finding and Touching the Finish Pole
 Paul Minner
Navigation System and Algorithm(s)
 Zhaolong Wu
Getting Rover Unstuck When It on Its Side
 Zhaolong Wu
Payload Fairing
 Zhaolong Wu

II. Framework

For our overall project design and framework, our team has decided to utilize the C++ programming language. We will be writing our individual portions of code using a C compiler in order to reduce the overall overhead and energy cost of the program, but we will eventually combine the portions into a C++ framework where we will be able to utilize classes and access modifiers. The team feels strongly that utilizing classes and access modifiers for a project of this magnitude would be a huge benefit, and, since C is a subset of C++, we can get the best of both programming languages. Using C++ compilers and C++ specific tools will also reduce time spent debugging and testing [1].

As far as the framework for the project, we will need a series of classes to pass control of the satellite between. This is due to the fact that our satellite's journey will be comprised of four primary stages where the controls for the satellite are vastly different from each other. All of these classes governing stages of control should have a private modifier so they can't be accessed by other classes.

The first stage of the satellite's expedition will be when it is encapsulated in its can-shaped housing. This stage will take place during the period of time that the rocket is on the ground prior to launch, and it will last until some period of time after the satellite has landed safely to the ground via the parachute. Though the satellite will remain in a static state during this period, it must be turned on and ready to receive instructions.

There will be a class designated for the control of the satellite for this period of time. The satellite will simply be listening for interrupts from other classes. There will be an interrupt that tells this class when to deploy its parachute. This interrupt will be provided by another class, and it will be based on some form of a timer. This event should occur shortly after the satellite has been ejected from the rocket. Another interrupt will tell the satellite to pass control to the class responsible for the next control stage of the expedition. Both of these classes with the interrupts should have a protected modifier so they are only visible to the class which has control of the satellite. This event should occur shortly after the satellite has been ejected from the rocket. This event should occur shortly after the satellite has landed safely on Earth's surface.

The second stage of the satellite's expedition will encompass the majority of the satellite's expedition, and there will be a class written specifically for the segment. This class will be responsible for the period of time that the rocket navigates from its landing zone until it has reached approximately 8-9 meters from the pole. The pole marks the final destination of the satellite, and the satellite must physically come in contact with the pole to finish the competition. The reason for the 8-9 meters is due to the fact that GPS is only accurate at 7.8 meters with a 95% confidence interval [2].

During this period there will be many other classes interacting with the class retaining control of the satellite. There should be a class which tells this control class if there is obstacles in the way and what direction to move to avoid them. In the case that the object is a tire track, which would have the potential to divert the path of the satellite for miles, then the satellite will need to create a 90 degree angle, perpendicular to the track and attempt to traverse it. This class should be a protected class as well, and should have a parent/child relationship with each class governing the sensors being used to identify obstacles.

There should also be a class which governs the GPS sensor. This class should help direct the satellite on a macro level, not taking into consideration any obstacles at the local level. This protected class should have its navigation instructions be overridden in the case of an obstacle in its path. This class will also send an interrupt to the control class when the satellite has reached 8-9 meters of the pole which will tell the control class to pass off control to the final governing the satellite's navigation.

The third stage of control will be reached only in the case where the satellite becomes stuck or on its side. The satellite design is being made specifically to preventing this type of situation from occurring, but there is still a chance of it happening. There will be a specific routine devised for this circumstance, and control of the satellite will be passed to this stage in this type of event. This class will have access to the CMOS sensor via inheriting access from the protected class. When the satellite has recovered from the event, and some routine to avert the obstacle has been completed, then the control will be returned to the previous control stage.

The final stage of control will cover the period of time when the satellite reaches the 8-9 meter range from the pole, and it will last until contact with the pole has been made. The class responsible for the satellite's control over this period will no longer take input from the GPS. The CMOS camera sensor system used previously for the obstacle avoidance system will be used here for navigation. The CMOS imaging sensor will need to switch modes in this stage to look for a pole shaped object and direct the satellite towards it. A separate protected class should be governing the sensor in this stage of navigation. Instead of looking for obstacles the sensor should be searching for an object which is similar to some stored image, while allowing for an appropriate level of error.

III. CMOS Image Sensing

For the ARLISS project we are using a CMOS imaging sensor, i.e. camera, in order to detect and avoid obstacles. The path of the satellite, on a macro level, is determined by the GPS sensor, but modifications to the path will need to be set locally using this system.

The idea behind the imaging sensor is that, when there is an abrupt change in color in an image, especially when referring to nature, it often shows that there is an abrupt change in the depth being viewed. For our purposes we want to use this observation to detect objects in the satellite's path, as well as, changes in terrain like a steep incline or a steep decline.

Manipulating raw camera frames can require a relatively large amount of space, as well as consume a large amount of energy. Energy is the most valuable resource for our project because we have a strict limit to the size and weight of the satellite so we have a very limited space for a battery to fit. There are several methods which we must use to minimize this cost, and there is one critical library in particular which we will use to help conserve this limited battery power.

As far as libraries go, we will be utilizing the library provided by OpenCV. This open-source library provides the capability of image and video I/O, image processing for individual video frames, as well as, built-in object recognition functionality [4]. All foreseeable video manipulating should be able to be done with this single library.

The plan for reducing energy consumption is simple; first we will take a frame of raw footage as input from the CMOS device, then we will convert it to grayscale, blur the image, and then run the image through a Canny photo filter to detect the edges. Blurring the image allows for small changes in color or depth to be lost during the edge detection. We are only interested in obstacles of a certain size so the greater the blurring of the image, the less noise from minor cracks in the ground or pebbles will show up in the final image. Canny edge detection is an effective, multi-stage algorithm which reduces noise in an image, and converts the pixels to black or white to show the contour of an object in the image. Using the OpenCv library we will be able to capture video and directly load each video frame from the sensor to an Mat object. OpenCv also has a built-in function imread() which can be used to convert each image to grayscale as it is read in from the camera.
This functionality of importing the live video feed, converting it to frames, and applying functions to the data for the edge detection will need to exist in its own class. This class will be responsible for the flow of the input video data from the sensor, and outputting the data as a Mat image to a separate class. The output from this class will help determine what sequence of events needs to occur in order to safely navigate the terrain.

IV. Continuous Tracks

For the method of how we would travel along the ground, the team members of the ARLISS Project decided to go with the continuous track, or tank tread, method of traveling. The implementation of these tracks will be carried out by the Mechanical Engineering sub-team of the ARLISS Project.

Continuous tracks are widely popular for their ability to traverse rough terrain much better than traditional wheels, however they sacrifice speed in doing so. Our continuous track system will require at least four separate motors, front left, front right, rear left and rear right. Four motors are required by this design because using only two motors leads to a couple different problems, mainly that either the satellite will not be able to turn or its tracks will be significantly underpowered.

Tracks of various styles and sizes will be tested to see which one best suits our needs. We will do our best to simulate the texture of the Black Rock Desert in Nevada where the competition will be held to get the best and most useful test results possible. Tracks will be compared based on ease of traveling across flat ground, how the tracks handle changes in elevation and if there is any noticeable effect on power consumption between tracks. We will also look at various stress tests for the different tracks, such as how easy are they to break and how easy it is to get the track to fall off the satellite. These tests will help us ensure that the satellite will perform without fail at the competition.These tests will range from being as simple as trying to pry the tracks off of the satellite with our hands to advanced as setting up a course of very rough terrain that will try to either break or remove the track. determining the proper length of tread on the track is very important for the satellite. if the tread is too long it won't travel along the hard, flat desert ground as efficiently as it could be. Make the tread too short, and the satellite might not be able to climb out of tire ruts or other divots in its path.

While the satellite is traveling on the rocket and during its journey back down to the ground, it will need to be packed away in a container the size of a standard soda can. In order to achieve this, the continuous track system will need to be able to compact itself in order to conserve space used inside of the of the soda can container. the way that this will be accomplished is by having small servo motors than can contract the Continuous track motor systems and tread into the confined space. Once the satellite has landed the servo motors will then expand the motors and the treads to their standard position for the expedition across the desert. This will be controlled by the software class that handles getting the satellite out of the soda can container, as described in the Framework section of this document.

V. Control Board

The control board acts as the physical central hub of our system, connecting together and managing the various I/O devices in our satellite system. The control board will be custom built by the Electrical and Computer Engineering sub-team of the ARLISS Project. Having a custom board built in house benefits us in a couple ways, it can make for the most efficient use of limited space and doesn't waste power or weight by not including I/O devices that we will not be using for our project, which would not be true if we used a pre-made third party board.

The current requirements for this control board include being able to handle I/O for our CMOS imaging sensor, sending it messages for when it needs to capture an image and the be able to move that data back for processing. The control board also needs to be able to support motor control functionality, individually telling our motors how much power to use and whether to turn forward or backward. The control board will also need to be able to interface with our parachute module, telling it when to deploy and hearing back from the module if it deployed properly. Lastly the control board will need to be able to interface with our on board GPS sensor. This is essential because the GPS sensor will handle deploying the parachute module as well as be the chief tool of navigation while on the ground trying to reach the final goal.

Our custom built control board will make use of the Atmel ATxmega128A4U chip. This chip is a low power eight bit AVR microcontroller featuring one hundred twenty eight kilobytes flash memory, with thirty four I/O pins and support for thirty four external interrupts[9]. The selling point of this chip is it has low power consumption, has enough I/O pins to suite our needs while also not going overboard. Finally, this chip is part of a family of chips, the mega128 family, that many people in our ARLISS project team are already familiar with as a similar chip from this family was used in the Electrical Engineering course Computer Organization and Assembly Language.

The detailed design and implementation is being handled by the Electrical Engineering team of the ARLISS Project. If designing and building our own board falls through and is no longer an option, the backup plan is to use the Raspberry PI Zero as our control board. The PI Zero will meet our basic needs for this project, but will still have some of the extra I/O devices and ports mentioned earlier that will not be needed for the ARLISS Project.

VI. Obstacle Avoidance

As detailed in the CMOS Image Sensing section, the data that we will be working with to detect and avoid obstacles will be a 2D array of binary values. A value of '1' will represent a change in the gray scale image, which will ultimately outline the terrain and give an easy to traverse dataset with which to avoid obstacles with. A value of '0' will represent no major change in shade of gray between neighboring pixels of the image, meaning that this will either be the interior of an object bounded by '1' values, or empty space in front of the camera.

The biggest concern and challenge facing our satellite project is the conservation of power and limiting power usage. Due to the space constraints of the competition our batteries will be smaller than desired for this kind of journey so we must conserve where we can. One way that we are doing this within the obstacle avoidance system is by not continuously taking in video feed from the camera. The setting where our system will be tested is for the most part very flat, meaning we can realistically take a new image to analyze after a short time interval without having to worry too much about a new obstacle appearing before we can avoid it. The exact time interval is not yet known, as it will take physically testing the satellite system to determine the correct value to use.

The way that the 2D array will be used to avoid obstacles is by examining the values by column from left to right to detect if an obstacle has appeared in our path. We expect there to almost always be a row of solid '1' values along the bottom of the 2D array that represents the horizon, where the ground meets the sky in the cameras eye. The algorithm for avoiding an object is quite simple, if a column or close to a column of '1' values appear towards the right side of the array, we stop forward motion, and rotate counterclockwise while periodically stopping to take a new picture to see if we have angled around the obstacle. once we have a picture showing no obstacles, we travel forward in this direction for approximately five seconds before going back to driving towards the target GPS coordinates. If the obstacle appears on the left side of the 2D array, we do the same thing except rotate clockwise.

Once we have the satellite built and the CMOS imaging implemented with the gray scale to 2D array conversion working, we will be able to test our satellite in various conditions and with different obstacles to see what kinds of obstacles we can drive over and ignore, and what obstacles must be driven around. During this testing period we will also tweak the timing at which how often a new picture will be taken and analyzed. The target for this testing is to find the time cycle at which we can take the fewest pictures while still avoiding all of the obstacles that we need to, and thus saving battery usage by the camera.

VII. Parachute Deployment

For the parachute deployment protocol, we have decided to use GPS to determine when to deploy the parachute. This method is perfect for us because we already need a GPS unit for other uses, and the GPS unit will be more accurate than just deploying the parachute based on a timer, and take less time in the air than deploying the parachute immediately.

The parachute deployment system will be the first part of our software to run, so we will start the system once the satellite separates from the rocket. This will be accomplished by tripping a mechanical switch which sends a signal to the satellite. Once started, the system will take periodic altitude measurements every second until the altitude drops below 1,000 feet. Once below 1,000 feet for two measurements in a row, the software will send a signal to deploy the parachute. We chose 1,000 feet as the desired height because the average reserve parachute for a skydiver deploys in less than 400 feet [10]. Since our satellite only weights as much as a can of soda, 400 feet is a conservative estimate of the distance the parachute will take to deploy. This should ensure we dont waste too much time in the air, but also gives us a large margin of error so we dont accidentally hit the ground before our parachute deploys. The worst case accuracy for civilian GPS coordinates is 25 meters (75 feet) [11]. This accuracy is much worse than the navigation specification because of the high speed the satellite is traveling. In addition, altitude error is actually about 1.5 times the amount of horizontal error, which can be a significant amount. In addition, if the GPS doesnt have an unobstructed view of the sky, the data cannot be trusted at all [11]. It is highly unlikely that our view will be obstructed, but it is possible a plane could fly overhead, or the rocket could briefly block our view of the sky. This is the reason we take two measurements before deploying the parachute. All of these factors effect the accuracy of our parachute deployment height, which is why we have been conservative. An extra 600 feet of distance to deploy the parachute is plenty of space, but still minimizes our time spent in the air. Once the software sends the signal to deploy the parachute, the parachute deployment system will end, and a new system will begin to check when the satellite is on the ground.

The system will be implemented within its own class which is responsible for deploying the parachute. An interrupt will be sent to the first stage class once the satellite is ejected from the rocket. Then, the parachute deployment class will launch its own function, which will loop continuously, checking the altitude from the GPS until the value of the altitude is less than 1,000 feet. Once this happens, the loop will execute one more time to check if the altitude is still below 1,000 feet. If it is, end the loop, send a signal to the first stage class to deploy the parachute, and hand control back to the first stage class. If the altitude isnt still below 1,000 feet, start the process over again. The first stage class will be responsible for actually deploying the parachute.

VIII. Getting Unstuck From Obstacles

The system for getting unstuck from obstacles has two components. First, determining when the satellite is stuck, and second, getting the satellite unstuck. To determine when the satellite is stuck, the satellites distance travelled will be monitored, and if the distance drops below a certain threshold for a specific amount of time, the system to get the satellite unstuck will be enabled [12]. To get the satellite unstuck, the system will attempt to move the satellite in different directions until the satellite frees itself.

The first component which checks if the satellite is stuck will be running constantly in the background while the satellite is navigating itself to its destination. It works by checking the coordinates from the GPS sensor every five seconds. After taking a new reading, the algorithm will determine the distance the satellite has moved in those five seconds. If the distance is below the minimum threshold of 1 meter, the satellite will switch from its normal navigation mode to getting unstuck mode [12]. This mode will work by attempting to move the satellite a different direction every five seconds, then checking the GPS coordinates like before to see if the distance travelled is greater than the minimum threshold of 1 meter. If it is, then the satellite must have moved and the system can switch back to its normal navigation mode. The first direction the rover will try is directly backwards, and each time the rover fails to get unstuck, the algorithm will attempt to drive 45 degrees clockwise of the previous direction driven. If the algorithm travels 360 degrees, then it will drive 45 degrees counterclockwise. This cycle will continue until the satellite gets unstuck, or the satellite runs out of battery.

This system will be implemented in two separate pieces. The first piece, which detects if the satellite is stuck, will be running concurrently with the navigation system. The algorithm will be a continuous loop, which checks the coordinates from the GPS, calculates the distance travelled from the current coordinate and the previous coordinates, checks if the distance is less than the minimum threshold, and sleeps for five seconds. If the distance is less than the minimum threshold, an interrupt will be sent, transferring control from the navigation system to the getting unstuck system. This piece will be located in the class which contains the second stage of control, which is navigation. This second piece will be located in the class which contains the third stage of control, which deals with the satellite becoming stuck, or on its side. The algorithm will be a continuous loop which increments the direction by 45 degrees, attempts to move for five seconds, and checks if the satellite has moved by calculating its distance travelled and comparing it to the minimum threshold. Once the satellite is determined to be unstuck, control will be transferred back to the navigation system.

IX. Finding and Touching the Finish Pole

For the protocol to find and touch the finish pole, we have decided to use the existing CMOS imaging sensor to find the pole. Once the pole has been located, the satellite just needs to drive in the direction of the pole until it makes contact with it. The most complicated portion of this is the object detection algorithm. This algorithm will use the OpenCV library, due to its object recognition capabilities [4]. The object detection algorithm, however, will work differently than the obstacle avoidance algorithm.

Control of the satellite will be switched to this system once the satellite is within the error range for the GPS coordinates of the finish. Once in this mode, this system must do two things, locate the pole, and move towards the pole. To locate the pole, the satellite will slowly rotate, scanning its surroundings with the CMOS imaging sensor. An image will be taken every second, and each image will be scanned for a specific shape. That shape is the shape of the finish pole, which would resemble a long, skinny rectangle on a 2D image. This shape scanning ability is achievable using the OpenCV library [13]. Once the shape has been located, the algorithm will instruct the satellite to move forward. Images will still be taken every second, and the direction the satellite is driving will be adjusted based on how far the shape is from the center of the image. If the shape is on the right of the image, the satellite will direct itself more to the right, and if the shape is on the left, the satellite will direct itself more to the left. Once the satellite is very close to the target the shape may not be detectable, so the satellite will continue to move forward in the same direction. Once the satellite hits the pole, it may get deflected and drive in another direction, but we will have completed the competition, so that isnt important. The system can turn itself off after a certain amount of time has passed.

This system will be implemented as its own class. It is the final stage class, and is started once the satellite is within the error margin of the finish pole. There will be two functions in this class. The first will search for the pole, and the second will move towards the pole. The first function will be implemented as a continuous loop, which rotates the satellite ten degrees, gets an image from the CMOS sensor, and checks that image for the shape of the pole using OpenCV. Once the pole is located, the next function will be called. This function will also be implemented as a loop. Each iteration, it will move the satellite forward, get an image from the CMOS sensor, use OpenCV to find the location of the pole in the image, and determine how far to the left or right to navigate the satellite based on the location of the pole in the image. Eventually, the satellite should hit the finish pole, and we will have completed our mission.

X. Navigation system design

Navigation system is the one of the most crucial parts of any autonomous vehicles. The team is planning to implementing the rover's navigating system with two major goals: to get the rover drive towards the target pole without getting stuck; Design or use some existing algorithms to let the rover remotely updates the best driving path, in order to achieve the optimized battery life. The team divided the navigation system implementation into hardware and software two pieces, in order to help us research more comprehensively.

On the hardware side, by far we have designed two layers to ensure that the CanSat to meet the functional requirements. The team first collaborated with the electrical engineering team and made the agreement on that we will use the FGPMMOPA6C MTK3339 GPS chipset. The MTK3339 ultimate GPS module has an excellent high-sensitivity receiver and a built in antenna. It is capable to get up to -165 dBm sensitivity, track 22 satellites on 66 channels with a 10 Hz update rate, with its ultra low power usage, this GPS module is unbeatable considering its the under 30 dollars price and ultra compact size(16mm * 16mm *5 mm, 4 grams). A GPS module essentially generates a set of way-points(path) based on the given target pole coordinates, in order to make the CanSat to have more precision control and a better motion awareness, we decided to add a 3 axis accelerometer onto the system. At this time being, we have not yet finalized the selection on which specific accelerometer we are going to use on the CanSat.

Software design is the most important piece of the Navigation system, as mentioned above, the CanSat will navigate itself to the target pole based on the given pole coordinates. In order to enable the autonomous driving feature, we have made a basic design which is based on an existing function called distance and angle calculation method.[14] With this method, the system essentially compares its current location with the target pole location, Calculate the shortest path between these two points, then travels through the shortest path. If the system senses any obstacles in the path, it will calculate the new path based on the new current location. This simple process is continued until the CanSat gets close to the target pole(roughly within 8 meters), the target pole mode will be turned on in this case. Lastly, if the CanSat has successfully touched the target pole, it gets stopped, and the whole system will be shut down, see **Fig. 1**. This distance and angle calculation method can be expressed in a set of mathematical equations, please see below:

$$A = Long_{pole} - long_{current}$$
$$B = Lat_{pole} - lat_{current}$$
$$\theta_{pole} = tan^{-}1(B/A)$$
$$Angle(\theta) = Pole_{angle} - Current_{angle}$$

Where, A is distance to be travelled in latitude and B is distance to be travelled in longitude

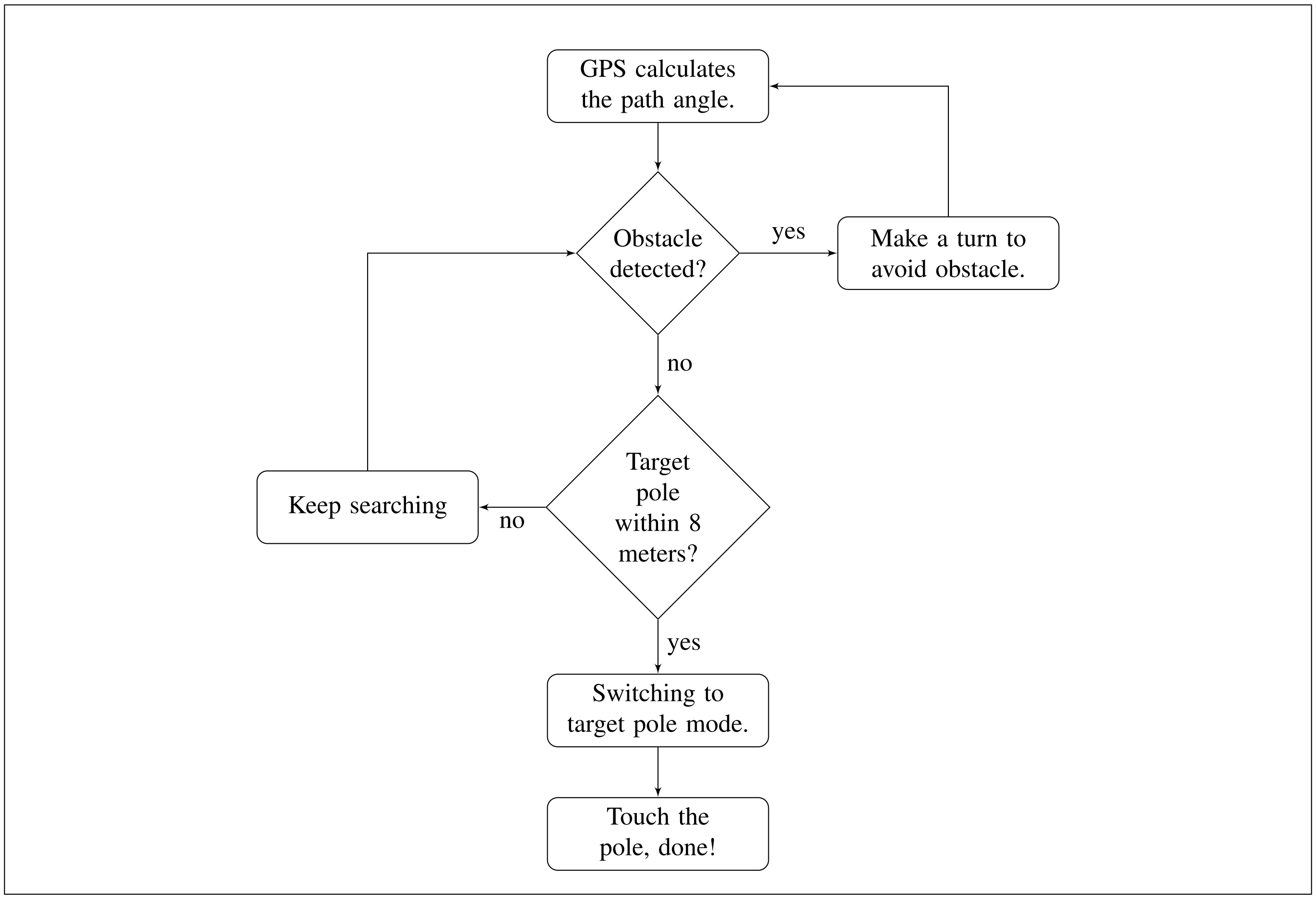

GPS calculates the path angle.
Obstacle detected?
yes
Make a turn to avoid obstacle.
no
Target pole within 8 meters?
no
Keep searching
yes
Switching to target pole mode.
Touch the pole, done!

Fig. 1: the flowchart above shows the details of how the navigation system works.

Furthermore, for details about the obstacle avoidance algorithms, as we discussed in the section No.VII, we are essentially using the CMOS image sensor to constantly generating pictures of the front of the path, convert those pictures into a 2D array of binary values, where 1 means object and 0 means open space. [15] This 2D array contains a set of binary values can be expressed as this imaginary figure below, the center of this array has a bunch of 1 values which represent an object, more specifically, is an obstacle.

```
00000000000000000000000000000000
00000000000001111100000000000000
00000000001111111111000000000000
00000000111111111111100000000000
00000000001111111110000000000000
00000000000111111100000000000000
00000000000001111000000000000000
00000000000000000000000000000000
```

XI. Getting rover unstuck when it on its side

The main goal of this task is to get the rover recovered from falling sideways. We have designed a very efficient protocol that ensures the rover is able get unstuck under all kinds of scenarios. This protocol can be simply broken down to two layers. The first layer is to determine whether the rover has fell sideways, and the second layer is using a servo driven mechanical arm on the top of the rover to get it unstuck when it lands sideways. These two layers has to be executed in a sequence to ensure the protocol operates in a cost effective fashion.

To determine whether the rover is operating under normal conditions or running into problems, we simply comparing its current moving speed and heading data from comparing the current waypoint with the last logged waypoint. If suddenly, the accelerometer captured an unexpected motion and the rover is constantly sitting on a same spot, or moving under a minimum threshold of 0.2m/s, if it is, the system will make the decision to confirm that the rover fell sideways. Then triggers the microprocessor to send an instruction to let the mechanical arm to get the rover up straight.

There used to be three layers in this protocol, but with the mechanical arm placed on the rover, it is capable to get the rover unstuck in any kind of situations, so the self-rescue program layer is a script we ar

The second layer is an add-on, though that the computer science team believed that the self-rescue program has enough power to recover the rover itself from land sideways. Mechanical engineering team has still made the decision on to place a servo driven mechanical arm on the rover in case the worst type of scenarios. This mechanical arm will be used in the fairing process as well as the second layer of the recovery protocol. A program will call the rescue function that lets the servo to move mechanical arm push the rover up when it's on its side.

XII. Payload fairing

The main purpose of the research done into this subsystem was to help determine the ways the payload can stay safe during its descent and landing. Fairing is essentially the first task the CanSat has to pass when it landed on the ground. The major goal of the fairing protocol is get the rover out then move away from the fairing, and parachute quickly, safely, and without stuck, jam, or having any residues.

Per competition requirement, it is known that the fairing will be built out of a soda can and will probably contain cushioning on the inside to protect the payload. So this limits the options that can be used to separate the fairing. We have developed a solution that is inspired by the SpaceX Falcon Heavy and Falcon 9 payload fairing design, or two half-shells fairing design.[16] We broke down this fairing protocol into two stages. First determine whether the CanSat is landed or not, if so then move onto the fairing separation stage so that the rover is able to move away as quick as possible.

To determine whether the CanSat is landed or not, we enable the GPS module to check the altitude constantly, then have the accelerometer to track its ground hitting motion. Speaking the motion, the impact from a CanSat hits ground is significant, any accelerometer in the market is capable to capture this motion. Soon as the accelerometer captured the CanSat's landing motion, a signal will be sent to the CanSat's microprocessor, which triggers the fairing protocol. The design of the fairing separation has been redesigned completely. The CanSat will be wrapped into a thick plastic sheet, we use tape to ensure that the CanSat will not move around during the launching stage. As mentioned above in section **XII**, there will be a servo powered mechanical arm placed on the top of rover to prevent the rover lands on its side, the mechanical arm is also used to help the rover breaks out of the plastic wrapped fairing.

Additionally, to ensure that the rover moves away from the parachute quickly is really important, previous years competition shown that several teams rover drove into the parachute, end up broken or stuck. To prevent this happens on the CanSat, we simply write a protocol that separates the parachute and the rover during the landing stage. We simply write a script that let the GPS and accelerometer to determine it's landing direction, and set the rover's initial heading the opposite direction of its descent heading. Many teams from previous years competition have stuck on this stage, specifically their CanSat were jammed by the parachute, made theirs rovers aren't able to go anywhere.

XIII. Amendments

A. *Control Board: 2/11/17*

This amendment reflects a change in the design in the control board. We are no longer using a custom built control board built by the ECE subteam, but are instead using a Raspberry Pi Zero. This change was made by the ECE team as a way to have a compact board with high processing power. It was realized that the 8-bit processor would not be mentioned previously would not be powerful enough to process our image conversions in a timely manner.

B. *Ultrasonic Radar Sensor: 2/08/17*

This amendment reflects a change to the design of the obstacle recognition system. The object recognition system is no longer using an ultrasonic radar sensor in its implementation. We will instead solely be using a CMOS camera sensor for this process. The entire design section for the ultrasonic radar sensor has been commented off in, and thus will not appear in this document.

C. Getting Unstuck From Obstacles: 2/11/17

This amendment reflects a change to the design of the system to get unstuck from obstacles. The system now no longer calculates average speed, but instead keeps track of distance traveled because it takes less time to compute and will achieve the same purpose.

D. Getting Unstuck When It On Its Side: 2/12/17

This amendment reflects a change to the design of the system to get unstuck from it lands its side. We are no longer using the programmed self-rescue method, but instead a servo driven mechanical arm will be placed on top of the rover, when needed it will push the rover up.

E. Payload Fairing: 2/12/17

This amendment reflects a change to the design of the system on the payload fairing method. The original two half method is not cost effective in this case anymore, instead we will use the second choice plastic wrapped method, we believed that this method has the most cost-effective fashion and will get its job done.

References

[1] jain.pk, “Using inline assembly in c/c,” Oct 2006. [Online]. Available: http://www.codeproject.com/articles/15971/using-inline-assembly-in-c-c

[2] “Gps accuracy,” Oct 2016. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[3] D. P. Massa, “Choosing an ultrasonic sensor for proximity or distance measurement part 2: Optimizing sensor selection,” Mar 1999. [Online]. Available: http://www.sensorsmag.com/sensors/acoustic-ultrasound/choosing-ultrasonic-sensor-proximity-or-distance-measurement-838

[4] G. Agam, “Introduction to programming with opencv,” Jan 2006. [Online]. Available: http://cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

[5] “Cmos camera as a sensor.” [Online]. Available: https://www.ikalogic.com/image-processing-as-a-sensor/

[6] D. Santo, “Autonomous cars’ pick: Camera, radar, lidar?” Jul 2016. [Online]. Available: http://www.eetimes.com/author.asp?section_id=36&doc_id=1330666

[7] “Ultrasonic sensors - measuring robot distance to a surface.” [Online]. Available: http://wpilib.screenstepslive.com/s/4485/m/13809/l/599715-ultrasonic-sensors-measuring-robot-distance-to-a-surface

[8] M. Grinberg, “Building an arduino robot, part ii: Programming the arduino,” Jan 2013. [Online]. Available: https://blog.miguelgrinberg.com/post/building-an-arduino-robot-part-ii-programming-the-arduino

[9] Atxmega128a4u. [Online]. Available: www.atmel.com/devices/ATXMEGA128A4U.aspx

[10] E. Scott, “Uspa raises minimum deployment altitude,” Feb 2013. [Online]. Available: https://skydiveuspa.wordpress.com/2013/08/02/uspa-raises-minimum-deployment-altitude/

[11] J. Mehaffey, “Gps altitude readout: How accurate?” [Online]. Available: http://gpsinformation.net/main/altitude.htm

[12] N. F. Joshua Herbach, “Detecting that an autonomous vehicle is in a stuck condition,” Mar 2015.

[13] A. Rosebrock, “Opencv shape detection,” Feb 2016.

[14] “Autonomous Vehicle Navigation and Mapping System.” [Online]. Available: http://www.rroij.com/open-access/autonomous-vehicle-navigation-and-mappingsystem.pdf

[15] “Autonomous Autonavigation Robot.” [Online]. Available: http://www.instructables.com/id/Autonomous-Autonavigation-Robot-Arduino/

[16] “FAIRING.” [Online]. Available: http://www.spacex.com/news/2013/04/12/fairing

Steven Silvers

____

Signature

____

Date

Zhaolong Wu

____

Signature

____

Date

Paul Minner

____

Signature

____

Date

Zachary DeVita

____

Signature

____

Date

Nancy Squires

____

Signature

____

Date

V. Tech Review

The ARLISS Project Technology Review CS 461

Steven Silvers, Paul Minner, Zhaolong Wu, Zachary DeVita
Capstone Group 27, Fall 2016

Abstract

This document reviews and evaluates potential pieces of technology that could be used to complete our project. Arguments are made for and against each piece of technology before finally settling on which technologies will be used when implementing the project.

CONTENTS

I. Introduction

In this document there are twelve components of our team's software which have been researched by the corresponding member from the list below. Each component compares and contrasts three alternate methods of accomplishing the same goal, and, at the end of each technology review, the method which best fits our teams needs has been chosen.

A Comparison of Languages
By Zachary DeVita
Methods of Object Recognition
By Zachary DeVita
Switching Modes to Locate the Pole
Zachary DeVita
Control Board
Steven Silvers
Avoiding Obstacles
Steven Silvers
Mode of Transportation
Steven Silvers
Parachute Deployment
Paul Minner
Getting Unstuck from Obstacles
Paul Minner
Find and Touch the Finish Pole
Paul Minner
Navigating algorithms selection
Zhaolong Wu
How to get unstuck if the rover fell or landed sideways
Zhaolong Wu
Payload fairing
Zhaolong Wu

II. A Comparison of Languages

For this tech review I will be comparing and analyzing the differences, and benefits of Assembly Language, C and C++ in regard to microcontrollers. These are the three most prominent languages used for programming single integrated circuit controllers, and each one has its own unique pros and cons.

A. Options

1) Assembly Language: Assembly language has some huge benefits that are virtually incomparable to C, or C++ for that matter. There are several functionalities which are possible in assembly which cannot be accomplished by higher level languages as well. In terms of speed and memory cost, assembly language blows both C and C++ out of the water; there is no comparison or debate on its superiority. The programmer retains ultimate control over each bit of memory when using assembly, but it is debatable whether this is a benefit or an encumbrance. With this control comes more potential for bugs, issues with readability, portability, testing, and a far more laborious implementation of the code. Assembly utilizes preprocessing directives and is used at the earliest stages of the bootloader which is not possible in higher level languages.
2) C: Though not nearly as compressed as assembly language, C utilizes a syntax which is much more compressed than most of higher level languages, which allows it to provide similar functionality at a lower cost of memory. In comparison to assembly language, C has a much larger library, and, thus, a larger instruction set and op-code associated with its commands. This can drastically slow performance in smaller processors which only have an 8 or 16-bit BUS because additional cycles are necessary to carry out each of the operations. This would only be a legitimate issue with small microcontrollers though. The benefits of C over assembly are considerably greater in this day n age where the amount of memory and speed are not nearly as concerning as in previous decades. C is far, far, far easier to program in. This allows C programs to be written faster, and with fewer errors. The programs are easier to read, debug, manage and maintain as well. An interesting fact, you can also write assembly language in a C program, but you cannot write a C program in assembly [1]. Im not sure why you would ever do such a thing but it is possible.

3) C++: It is a common assumption that C++ is unsuitable for use in small embedded systems. This is partially due to the fact that 8-bit and 16-bit processors lack a C++ compiler, but now there are 32-bit microcontrollers available which are supported by C++ compilers. The majority of C++ features have no impact on code size or on CPU speed when being compared to C [2]. C++ has all the same benefits over assembly language that C has, which are very important considerations when choosing the language. The most focal comparison here would be between C and C++. One of the original intentions of C++ was to prevent more mistakes at compile time, and, in comparison to C, it has achieved its goal. It is cleaner than C, easier to debug, easier to read and the code is more robust. C++ has added functionality, i.e. function name overloading, and classes which can be declared using different access modifiers.

B. Goals for Use in Design

The programming language is hugely important for the success of the project. Different languages offer different sets of functionalities, some which might be necessary to meet our goals and some which might just add unnecessary overhead. This decision might be the most important decision made at this stage in the project.

C. Criteria Being Evaluated

The most important aspects to consider when selecting a programming language for our specific project are going to be energy consumption, and ease of use. We are very tight on energy due to our satellite having size and weight constraints, and due to the fact that we have no idea how far our satellite might need to travel to reach its destination. Obviously ease of use would be important because this project is very time sensitive. Additional criteria being evaluated are the speed at which the code is executed, the memory cost of executing the same block of code, and the amount of extra functionality being provided by one language over another. This functionality may be gained from libraries or by tools that are made available by using that language.

Another important consideration is that you only pay for what you use. This is in regards to the cost of memory which seems to always be a concern when considering C++. What this means is that, because C++ is almost exactly a superset of C, there is no principally caused in a program that only uses C features [3]. A block of code written in C can be run through a C++ compiler, and the compiler will treat it the same as if it was ran through a C compiler; it will produce the same machine code.

The big negative with C++ is with the additional overhead which is common in higher level languages. In our specific circumstance, this isnt really a memory or bus size issue as much as a power issue. Because our satellite is so small, and will only weigh a maximum of 150 ounces, the overhead issue starts to become a legitimate issue for whether or not we can power this satellite long enough to reach its destination. The team that won last year had their satellite travel approximately 7 miles to its destination after being blown way off course. Having a program which uses twice as many micro-operations or cycles to complete the same task will absolutely require more energy to operate.

Visual Comparison of Languages

	Energy Consumption	**Ease of Use**	**Functionality**	**Memory Cost**	**Speed**
Assembly	1st	3rd	3rd	1st	1st
C	2nd	2nd	2nd	2nd	2nd
C++	3rd	1st	1st	3rd	3rd

D. Selection

Based on the research, Im a little torn in my decision. I thought coming into this assignment that C was the obvious choice because of lower cost of memory and higher speeds than other higher level languages, and its substantial benefit over assembly with ease of use, debugging and maintainability. Assembly is out of the question primarily due to there being a considerable chance we would never finish the project. Assembly is just unrealistic for our purposes, and both memory and speed shouldnt be too much of an issue in our project since we will be building this satellite in 2017.

There are some important benefits for using C++, particularly the use of classes and access modifiers, and the fact that it is easier to debug and better at catching bugs at compile-time. Considering the fact that almost all C code can be cut and pasted into a C++ compiler and return the same machine code, our team will use C and utilize the benefits of slightly higher speeds and a lower memory cost, but we will reserve the option to port over to C++ in the case that we decide to implement classes with access modifiers.

III. Methods of Object Recognition

There are many sensors used for object recognition, each with its own differences and benefits. For our project we are looking for an object recognition system which will be accurate, has a relatively high resolution at a short range, is light on energy consumption, and which will operate properly in potentially 100+°F temperatures. These are the primary concerns for choosing this hardware, and this still leaves many options to choose from. The two types of sensors used for this type of functionality are proximity sensors and image sensors. Proximity sensors include the photoelectric, light sensors, as well as the ultrasonic, a.k.a. radar, sensors. These two sensors, as well as, imaging sensors will be analyzed and compared in this technical review.

A. Options

1) Photoelectric Sensors: As far as photoelectric sensors go, I did some research on LIDAR and Sharp IR sensors as they are the most applicable for our teams project. I found a lot of interesting articles about these sensors; they shine a small light at a surface and measure the time it takes for the light to be reflected back. These types of sensors can determine the distance of objects in near-instantaneous time because light moves so fast. They have proven to be very accurate in the right settings as well. IR sensors are very cheap as well and use little energy to be powered. LIDAR sensors, on the other hand, are much less energy efficient and seem to cost a considerable amount of money for a reliable one [4]. The worst part about photoelectric sensors is that sunlight causes a tremendous amount of noise. These sensors will be completely unreliable and ineffective for our purposes, as our satellite will be navigating through the Nevada desert. Photoelectric sensors are absolutely not an option for our teams object recognition system.

2) Ultrasonic Sensors: Ultrasonic sensors, on the other hand, are not compromised by direct sunlight. These sensors are cheap to purchase, light weight, use little energy to be powered, and are accurate in most cases. They rely on the speed of sound opposed to light to determine distances of objects so the response is not nearly as immediate as with photoelectric sensors, but the difference is negligible in the scope of our project. These sensors have a range of more than 6 meters which is more than the range necessary for our requirements. These sensors provide accurate detection of even small objects, and are reliable in critical ambient conditions where there is lots of dirt and dust [5]. They are not easily weatherproofed for a humid climate, but that should not be a problem where the competition is being held.

3) Image Sensing: Image sensing is a third option for object detection. Image sensing is primarily done using a CMOS (complementary metal oxide semiconductor) or a CCD (charge-coupled device) sensor which are commonly used in digital cameras. Both of these systems convert light into electrons, but the methods each of them uses have very distinct differences. Of these two sensors I have chosen CMOS for the comparison based on the fact that they are less sensitive to light, extremely inexpensive, and more importantly, consume far less power than their counterpart [6].

CMOS sensors create a moderate amount of noise due to the process of converting light into electrons. This has a potential to affect the accuracy of a reading. Because these sensors have no way to determine distance, they often require the additional support of photoelectric or ultrasonic sensors. Other relevant perks to using image sensors is that they are able to prioritize obstacles or dangers, and they can detect specific shapes. This is important for the final leg of the satellites expedition, where the satellite must recognize the shape of the pole which it must make contact with to finish the competition. Using an image sensor will definitely require far more complicated programming to detect objects but it seems like a valid consideration for the project [7].

B. Goals for Use in Design

The goal for the sensor, if not clear, is that we need some kind of sensor to send data to the micro-controller so that the data may be analyzed and obstacles can be detected. If the obstacles can be detected using some system of sensors, than the obstacles can also be avoided. The goal is to avoid all obstacles which are determined to be a threat to our satellite reaching its destination.

C. Criteria Being Evaluated

The most important criteria that these sensors are being evaluated on would be the accuracy of the sensor and the energy consumption by the sensor. Again, energy consumption is important because we have tight constraints on space and weight for the satellite. This means that we are limited to a relatively small battery to provide power to the satellite for the entire duration of its expedition. Accuracy is important because we don't want the satellite to miss an obstacle or waste energy thinking there are obstacles when there are none. We need a sensor which creates minimal noise while operating. Also important criteria can be the range of a sensor, the cost of a sensor, and any environmental conditions which may cause adverse affects to the precision of the sensor.

A Comparison of Object Recognition Sensors

	Accuracy	**Range**	**Energy Consumption**	**Cost**	**Environmental Conditions**
Photoelectric	High	0.8m	Low	Low	Won't work in direct sunlight
Ultrasonic	High	6m	Low	Low	Problems at high pressure or with humidity
CMOS	High	N/A	Low	Low	Problems with humidity

D. Selection

After an analysis and comparison of the various sensors our team has chosen to use a CMOS imaging sensor. The CMOS imaging sensor will be used to identify obstacles in the satellites path, and we will estimate distance based on location of obstacles relative to the bottom of the frame. Since the camera is fixed, objects on flat ground will appear in approximately the same location in any given frame. There will be an accelerometer onboard so pitch can be taken into consideration when determining distances on an incline. I believe this will be an effective and accurate method of detecting static obstacles in Black Rock Desert, Nevada. The camera sensor is also cheap, effective for the climate, and can be implemented to minimize energy consumption.

IV. Switching Modes to Locate the Pole

At the satellites final destination there will be a metal pole which the satellite must bump in order to successfully finish the objective. The problem created by this is that the GPS, global positioning satellite, is only accurate to within 7.8 meters with a 95% confidence interval, according to the U.S. government [8]. This means that between 8-9 meters of the pole that there must be a switch to a different interface to guide the satellite. The pole is metal so we have the option of using capacitive or inductive sensors. The other option, which seems much more difficult but may be necessary due to the range, would be utilizing the CMOS image sensing technology that we are already using for the object recognition.

A. Options

1) Capacitive Sensor: A capacitive sensor is a proximity sensor that can detect nearby object by their effect on the electrical field that is sent out by the sensor. Capacitive sensors are cheap, light and relatively durable. Like any electrical component, they have problems in humidity, but they would be much easier to weatherproof than, for example, an ultrasonic sensor. Either way, humidity should not be an issue in the tail end of summer, and in the Nevada desert. This type of sensor will detect anything with a positive or negative charge which could be a potential problem, but probably not at the location of the competition [9]. The bad news about this approach is that capacitive sensors only have a range of approximately 1cm or less. This is not an option for our project.

2) Inductive Sensor: Inductive sensors are proximity sensors as well, but they can only detect metal, whereas the capacitive sensor can detect anything which affects the electric field. Again, these are cheap to buy, light, durable, easy to weatherproof for our purposes, etc. Unless there will be other metal objects in the satellites path for the last 9 meters, I would feel safe to say that this would be accurate as well. But again the range makes these sensors, similarly, not a viable solution for the problem. These sensors have a range of up to 2 inches [10].

3) Image Sensing: After analyzing GPS, capacitive and inductive sensors, all there is left is using the CMOS image sensing technology that will already be in place. The advantage of using CMOS is that the system will already be in place for the obstacle recognition component of the project. This means that there is no additional weight or space used. This also means that there will not be any additional energy consumption, cost or weatherproofing associated with the selection. There will be much more programming needed to implement this design though.

B. Goals for Use in Design

The goal for this component of the software is to determine what method is being when switching modes in the final stage of the satellite's expedition. The sensor being used to locate the pole which the satellite must make contact with must be established. It is based off this that we are able to design the software for making the transition.

C. Criteria being evaluated

The most important criterias for this component are obviously the accuracy of the sensor, and also the range of the sensor. The range is important because GPS is only viable until approximately eight meters from the target so the system must begin working at around eight meters. Other factors being considered are cost, energy consumption and environmental effects on the sensor.

A Comparison of Methods to Locate the Pole

	Accuracy	**Range**	**Energy Consumption**	**Cost**	**Environmental Conditions**
Capacitive	Moderate	1.0cm	Low	Low	False positives near electrically charged objects.
Inductive	High	2 in	Low	Low	Problemswith humidity
CMOS	High	N/A	*None	*None	Problems with humidity

D. Selection

After analyzing GPS, capacitive and inductive sensors, all there is left is using the CMOS image sensing technology that will already be in place. The way this will have to work is that when the satellite is within 8-9 meters of the pole, the program will need to alternate to a different interface. In this interface, the satellite will be guided by the object recognition system, and it will quit reading data from the GPS. The object recognition software will be designed to pick out abrupt changes in light on the edges of objects which signifies a change in depth. When these objects are interpreted, they will be compared to an object that is a digital representation of the pole. When the satellite finds an object which has the same shape, obviously using some level of error in the computation, it will self-drive itself to the object. This seems like a difficult chunk of code to produce, but it is possible, and it has been done before. Considering there will likely not be any objects, i.e. rocks, which have a similar shape to the pole, I think this is an excellent way to solve this problem.

V. Control Board

A. Options

The options being considered for the control board in our project are the Arduino Uno, Rasperry Pi Model A+ and a custom board developed by a team of ECE capstone students.

B. Goals for Use

The control board is what brings all the hardware together to create a single, functioning system. Our software system will be loaded directly onto this board to control input and output to the various system peripherals, such as the motors and GPS module.

C. Evaluation Criteria

These board options will be evaluated based on their ability to run the embedded system that we develop, ability to interface with various required hardware sensors of the system, if they meet the space constraints of the CanSat design, and how well they manage limited resources such as power.

D. Discussion

a) Arduino Uno: The Arduino Uno is the base level Arduino board and is based upon the ATmega328P board with fourteen digital I/O pins, six of which support pulse with modulation[11]. Arduino supports its own language based on Java, C and C++ as well as its own IDE making it easy to work with. The Uno is 68.6mm long by 53.4mm wide, which means it will fit in a soda can with dimensions 66.3mm by 115.2mm however, we do not know how much of that space will be taken up by other hardware. Battery wise the Uno only requires a maintained five volts from a battery to operate. The Uno is a traditional microcontroller, once it is given power it immediately begins running code which may be ideal for our project.
b) Raspberry Pi Zero: The Raspberry Pi Zero is a well documented pre-built board that could be easily adapted to our project. It is small enough to easily fit within our CanSat while also leaving space for other electrical components.[12] Another advantage of the Pi is the built in camera port, which will be needed for the obstacle avoidance system.
c) Custom Board: A custom built board would be the best option if done correctly. The dimensions could be developed with the CanSat restrictions in mind and could only include the necessary hardware. On the downside it would be a untested board, with no way of knowing how much power it would consume or if it would be able to run our system until after it is designed, built and tested.

E. Selection

With these pros and cons in mind, I would select the Raspberry Pi Zero as the controller for our project. It is a popular, well documented board that we know for sure works. My second choice would be the custom board, only because with the given time line of the project if it didn't work we would be in a very bad place. The Arduino, while an excellent board for embedded systems, may not provided the needed processing power for our system.

VI. Avoiding Obstacles

A. Options

Options for obstacle avoidance include detecting obstacles and then driving out of the way to get around it, develop an algorithm to determine if it can be driven over or not, or drive straight through the obstacle.

B. Goals for Use

The goal of this system is to make sure our CanSat can reach the finish destination without getting caught or stuck on anything in the CanSat's way.

C. Evaluation Criteria

The technology will be evaluated on how easy it will be to develop, and how effective it will be at accomplishing the goal of reaching the destination.

D. Discussion

a) Drive Around: This method of avoiding obstacles while seeming like the smart choice has some problems. Driving around an obstacle would be smart if the obstacle were say a large rock, but if it were something like a long ditch created by a car tire it could go on for miles, and our CanSat would never be able to reach the destination. It could be fairly easy to develop, as soon as it detected an obstacle it could run a routine that makes the CanSat backup, turn and move forward a bit before trying to proceed to the goal.
b) No Avoidance: This method while sounding awful could actually work. Our CanSat will be tested in the Black Rock Desert of Nevada, a very flat, open space. It would be very easy to develop this as it would mean no programming, which would also save memory on the board. However, if any obstacles at all popped it could be game over for our CanSat.
c) Algorithm Avoidance: This would be the most difficult of the options to implement and would involve heavy testing as well as a great knowledge of automated driving systems. On the up side, knowing when the CanSat can and cannot get over or past an obstacle would be extremely helpful, cutting down drive time and therefore saving battery.

E. Selection

With the evaluation criteria in mind, developing an algorithm to determine whether or not the CanSat should try to avoid an obstacle is the best option for accomplishing our goal. While the other two options would be much easier, they also carry much higher risk of the CanSat failing to reach the finish point, making the algorithm based avoidance the responsible choice.

VII. Mode of Transportation

A. Options

The options considered for mode of transportation are a traditional two wheel setup with a wheel at the top and the bottom of the CanSat and the CanSat being oriented so that the can body would by horizontal. The next option would be using caterpillar tracks with the can oriented vertically. The final option would be a four wheel setup with the can oriented horizontally.

B. Goals for Use

The goal for this part of the system is the actual mode of transportation for getting the CanSat from where it landed to its final destination.

C. Evaluation Criteria

This technology will be evaluated on its ability to efficiently transport the CanSat, ability to handle rough terrain, and ability to conserve space within the delivery CanSat.

D. Discussion

a) Two Wheels: The two wheel setup would most likely use the least amount of power out of the three options, as there would be two total motors. Since they would be oriented horizontally, you could get the biggest wheels possible with the size constraints of the can. Bigger wheels would be better for getting over certain obstacles as opposed to small wheels. With the horizontal orientation, there would not be very much ground clearance between the CanSat body and the ground, which could raise a problem with very small obstacles. Because it is oriented on only two wheels, getting flipped on becomes a not very big concerned because the wheels would always be touching the ground.

b) Four wheels: The four wheel option would provide a stable base for driving the CanSat, but also opens up the CanSat for possibly getting flipped by an obstacle and stuck. This option could be done with two or four motors, but with size limitations already being pushed with four wheels, it would most likely be limited to two motors.
c) Caterpillar Tracks: This option is interesting, as the tracks would definitely be the best at traversing rough terrain and getting over obstacles. On the downside they would take up a lot more room in the CanSat than traditional wheels and would also require a bit more power.

E. Selection

Based on the above criteria and the pros and cons of each, I would use the caterpillar tracks. What they give up in size consumption and power consumption, they more than make up for in ability to get over rough terrain. It does not matter how much space or power is saved for other devices if our CanSat gets stuck on an obstacle while driving to the goal.

VIII. Parachute Deployment

A. Options

The first option being considered for deploying the parachute is deploying at a specific altitude based on an altitude sensor. This option would require software to be written to read the values sent from an altitude sensor. The second option is deploying the parachute after a certain amount of time. This would require a timer to implement. The last, and simplest solution is to deploy the parachute immediately after the satellite separates from the rocket. This option could be entirely mechanical, and therefore not require any software to implement.

B. Goals for Use

In order for the parachute deployment to be considered a success, it must activate before hitting the ground. Ideally, the satellite would reach the ground as quickly as safely possible in order to maximize the battery life of the satellite.

C. Evaluation Criteria

These options will be evaluated based on simplicity, time taken to hit the ground, accuracy, power draw. Simplicity is important to see if a slightly better solution is actually worth the effort to implement. The time taken to hit the ground has an effect on how much power is used before the satellite starts moving towards its target and is therefore important. Accuracy is important since we dont get a second chance if the satellite is destroyed. Finally, power draw is important since power is a limited resource.

Comparison of Evaluation Criteria for Parachute Deployment

	Simplicity	**Time**	**Accuracy**	**Power**
Altitude Sensor	Least	Fastest	High	High
Timer	Less	Fast	Lower	High
Immediate Deployment	Most	Slow	High	Low

D. Discussion

The altitude sensor solution will most likely be the most accurate, and take the least amount of time to hit the ground, but may draw more power than other solutions, and is the least simplistic. This is because software has to be written to check the value of a sensor incrementally, which is the most complicated solution, and constantly checking the value of a sensor will drain power faster than if there were no software. The timer based solution will be less accurate, and draw more power than the final solution, but will take less time to hit the ground, and is simpler than the first solution. There is still the same problem about power, which is that a timer has to be checked constantly and therefore drains power. The final solution to deploy the parachute immediately is the simplest solution, since it requires more software, but it will take much longer for the satellite to reach the ground than the other two solutions. The accuracy of this solution should be excellent as well.

E. Selection

We have selected the altitude sensor based solution. This is because we need the satellite to reach the ground relatively quickly to save power. We believe the power draw from the software to check the altitude sensor would be less than the power lost from being stuck in the air for much longer if we had picked the immediate deployment method. Since we are already planning to use a GPS, most GPS sensors give an altitude measurement as well, so we can use existing hardware.

IX. Getting Unstuck from Obstacles

A. Options

Our three options were considering involve either adding another hardware component, using existing servos, or just using the treads. The first option is to add a mechanical arm which deploys to get the satellite unstuck from whatever it hit. The second option is to attempt moving different directions with the treads while moving the treads up and down with the servos they are attached to. The third option is to simply use the treads on their own and attempt moving different directions until the satellite can move.

B. Goals for Use

The goal of this system is to recover the satellite if the obstacle avoidance system fails and the satellite hits an obstacle. This is a difficult goal since we dont expect the obstacle avoidance system to fail, and if it does, the rover has encountered something we didnt anticipate. Hopefully, this system is never actually used.

C. Evaluation Criteria

These options will be evaluated on simplicity, likelihood of success, and space constraints. Simplicity is important since more effort may not be worth a slight improvement over another option. Likelihood of success is difficult to guess, but obviously it is very important since the solution needs to work. Space constraints are important since space is limited on the satellite.

Comparision of Evaluation Criteria for Getting Unstuck From Obstacles

	Simplicity	**Success**	**Space**
Mechanical Arm	Least	Most Likely	Uses Space
Moving Tread Servos	Less	Likely	Doesn't Use Space
Moving Different Directions	Most	Least Likely	Doesn't Use Space

D. Discussion

The mechanical arm option is the most likely to succeed since it could implement everything the other options implement as well, but it is by far the most complicated and uses the most space. Determining the best placement of the arm and how to use it also isnt obvious. The arm would be most useful when the treads are completely stuck, because the other options wouldnt be able to deal with this. The option which raises and lowers the treads would be the next most complicated, but it may be more likely to succeed than the last option. Without testing, however, its difficult to determine how helpful changing the height of the treads would actually be in getting unstuck. This option doesnt require any extra space, either. The last option to simply move the treads in different directions to get unstuck is the simplest solution, and requires no space. This option is also the least likely to work, since all other options implement this as well as something else.

E. Selection

We will use the simplest solution, which just attempts to move in different directions until the satellite can move. We picked this solution because it is the simplest to implement, and we werent sure how well the other solutions would actually work. The mechanical arm option would take up too much space, so its not an option, but the option which raises and lowers the treads would have worked. We just dont know if changing the height of the treads would actually help get the satellite unstuck, so for now we arent including it. If, in testing, we find that our simple solution doesnt work well, we may implement the ability to raise and lower the treads as well.

X. Find and Touch the Finish Pole

A. Options

Our options are based on either brute force, or sensors. The first option is to drive in a square pattern, like a lawnmower, in the general area of the pole until we hit it. The second option is to drive in a circular pattern outward until we hit the pole. The final option is to use sensors to detect where the pole is, and drive straight into it.

B. Goals for Use

Our goal is to hit the pole as quickly as possible. We must hit the pole to finish the competition, so this system needs to work correctly, otherwise we lose the competition. Speed is important to preserve battery life. The satellite will likely be low on power at this point, so finishing before it runs out of power is important.

C. Evaluation Criteria

We will evaluate these options based on speed, simplicity and accuracy. Speed is important to make sure we dont run out of power. Simplicity is important to save us time. Finally, accuracy is important because if we fail to hit the pole, we lose the competition.

Comparison of Evaluation Criteria for Finding and Touching the Finish Pole

	Speed	**Simplicity**	**Accuracy**
Square Pattern	Slow	Simple	More Accurate
Circular Pattern	Slow	Simple	More Accurate
Detect using Sensors	Fast	Complicated	Less Accurate

D. Discussion

A separate mode of finding the finish pole is needed because GPS isnt accurate enough. GPS is accurate to within a few meters, but the pole will likely only be a couple inches wide. The first option solves this problem by getting to where the GPS says the pole is an searching in a square pattern, like you would run a lawn mower. This is very simple to implement, and should be accurate. Unfortunately, it may take a long time to find the pole. The second option is just like the first, except the satellite searches in a circular pattern, instead of a square pattern. This has the same pros and cons as the first option. The last option is to search for the pole using the sensors used for the obstacle avoidance system, then driving straight into the pole. This option is the most complicated, and may be less accurate than the other solutions, but could be much faster.

E. Selection

We chose to go with the sensor approach. We believe power is going to be a very limited resource for this competition, and the more time we spend searching for the finish pole, the more power we drain running the motors. Although the other methods are guaranteed to hit the pole eventually, we may run out of power before hitting it which would make the accuracy pointless. We will do more tests with the sensors to ensure it is reliable enough.

XI. Navigating algorithms selection

A. Options

- Shortest path method: rover drives to the target pole directly based on the navigation module generated shortest path, without using any obstacle avoidance.
- Decision making method: rover drives to the target pole with obstacle detections. We set a boolean numbers in the program where 1 represents obstacle and 0 represents clear path. When there's an obstacle on the rover's path the obstacle avoidance module will be enabled. [**?**]
- Shortest path + Decision: this method is the combination of these two methods above. The navigation module keeps updating the new shortest paths after the rover went around the obstacle.[**?**]

B. Goals for Use

Navigation system is the one of the most crucial parts of any autonomous vehicles. The main goal for this part of the project is to get the rover drive towards it's target without any stuck,

C. Evaluation Criteria

We evaluated these options based on easiness of implementation, resource needed, Path length, and chances of getting stuck.

Comparison of Evaluation Criteria Navigating Algorithm Selection

	Simplicity	**Resources needed**	**Path length**	**Chances of getting stuck**
Shortest path	Most	Low	Short	High
Decision making	Less	Medium	High	very low
Shortest path + decision making	least	High	Medium	low

D. Discussion

The shortest path method is the easiest implementation for this task, but it has very high chances to hit an obstacle, may result being stuck. The decision making method is less simple, it takes reasonable amount of resources, and having a very low rate to get stuck, the only problem for this method is it might result the rover to travel a very long path, which takes a lot of battery. The combined method is the hardest one to implement but will give us the best outcomes.

E. Selection

The teams selection is we implement and test the decision making method first, if by the end we have extra time we will add the shortest path method on to it. Ultimately the combined method would be a cool feature and hopefully will make the impression.

XII. How to get unstuck if the rover landed sideways

A. Options

- Two wheel rover solution: Two wheel design is a great solution in order to prevent the rover falls sideways, because of its unique shape and weight distribution, the rover has a very low center gravity, that can guarantee itself won't fall sideways.
- Programming approach: Design an algorithm that makes the rover to move the tread opposite of its moving direction when it encounters this situation.
- Mechanical arm solution: By adding a servo driven arm onto the rover, it will get the rover unstuck when the rover landed sideways.

B. Goals for Use

The main goal of this task is to get the rover recovers itself in normal position from landed or fell sideways.

C. Evaluation Criteria

We evaluated these options based on feasibility, chances of success, and space requirement. Feasibility is the first thing to be considered, it would be a waste of time if we found the method we have chosen is less possible to archive after the implementation has started. Speaking on the chances of success, we want to ensure that the method we choose turns out has the best success rate. Per competition rules & requirements, payload's space is extremely limited, space requirement is also an important criterion.

Comparison of Evaluation Criteria for Getting Unstuck if the Rover Landed Sideways

	Feasibility	**Chances of success**	**Space taken**
Two wheel rover	low	High	Optimized
Program the Treads move clockwise	High	Medium	No space needed
Mechanical arm	Low	Medium	Takes space

D. Discussion

Two wheel rover is a great method, it has unbeatable success rate and it utilizes the most of space in a soda can, but definitely beyond our knowledge to implement at this time being. Programming solution is the most feasible method for us computer science major student, the success rate is relatively high and it doesn't require any additional component so saves up some spaces. Mechanical arm method is also great but beyond our knowledge to implement, its success rate is only marginally higher than the programming method.

E. Selection

We chose the servo driven mechanical arm option. Recalling our goal is to get the rover recovers itself in normal position from landed or fell sideways, with a mechanical arm it can guarantee that to get the rover unstuck when it on its side.

XIII. Payload fairing

A. Options

- Two half-shells solution[?] - This method is inspired by the SpaceX Falcon Heavy and Falcon 9 payload fairing design. We cut the soda can into two halves and put the rover into it, the can breaks into halves when it landed on the ground, in order to let the rover moves away freely.
- Wrapping solution - We cut the top and bottom of the soda can off, then cut the rest of it in a flat sheet, then wrap the rover in it and tape them together. Ultimately, this fairing method works the best with the two wheel rover design.
- Catapult solution - In this method, we first cut the top of the soda can off, put a spring in the can then press the rover into it, when the can hit the ground it triggers to release the spring that throws the rover out of the soda can.

B. Goals for Use

The major goal for this part of the system is get the rover out and move away from the fairing quickly, safely and without stuck or having any residues. This is one of the must done tasks in this project.

C. Evaluation Criteria

We evaluated these options based on feasibility, rover's safety, and probability to get stuck or having residues. Feasibility is the first thing to be considered since we want to get everything working in cost-effective ways. Rover's safety is a crucial point, if the rover is broken or damaged during the fairing stage then the whole competition is over. Probability of getting stuck or having residues is also important since the rover might also get damaged if something is jammed into its wheels or body.

Comparison of Evaluation Criteria for Payload Fairing

	Feasibility	**Rover's Safety**	**Probability of getting stuck or having residues**
Two half-shells	High	High	Medium
Wrapping	High	Medium	High
Catapult	Low	Low	Low

D. Discussion

Two half-shells option is a feasible, and relatively safe method, the only problem is it's possible to get the rover stuck in the can. Wrapping option is also a relatively feasible method, but because we break the can into too many pieces that we can't guarantee the rover's safety during its launching and landing stages, those pieces may also end up stuck in the rover. The catapult method has the least feasibility, it puts a lot of stress on the rover, also throw the rover out of the can might damage it. The only advantage of this method is it guarantees the rover can get out of the soda can, and without having any residues.

E. Selection

Upon the discussions, we made the final decision that we will implement the payload fairing by using the wrapping method. It was an easy decision for us since we made it clear in the discussion that this method is the most feasible and will give the team the best outcomes.

XIV. Conclusion

This document detailed our plans for different pieces of technology which could be used to complete our project. Finding multiple solutions for each technology allowed us to think more deeply about which solution would really be best for each technology. We believe the choices we have picked in this document will give us the best outcome for our project, although some of these decisions may change over the coming weeks, as we better understand what we are building.

XV. References

References

[1] jain.pk, “Using inline assembly in c/c,” Oct 2006. [Online]. Available: http://www.codeproject.com/articles/15971/using-inline-assembly-in-c-c

[2] D. Herity, “Modern c in embedded systems part 1: Myth and reality,” Feb 2015. [Online]. Available: http://www.embedded.com/design/programming-languages-and-tools/4438660/modern-c--in-embedded-systems---part-1--myth-and-reality

[3] “C vs c++.” [Online]. Available: http://www.mikrocontroller.net/articles/C_vs_C%2B%2B

[4] “How does lidar work? the science behind the technology,” 2016. [Online]. Available: http://www.lidar-uk.com/how-lidar-works/

[5] B. GmbH, “Ultrasonic sensors,” 2016. [Online]. Available: http://www.balluff.com/balluff/mde/en/products/overview-ultrasonic-sensors.jsp

[6] “What is the difference between ccd and cmos image sensors in a digital camera?” Apr 2000. [Online]. Available: http://electronics.howstuffworks.com/cameras-photography/digital/question362.htm

[7] “Types of sensors for target detection and tracking,” Nov 2013. [Online]. Available: https://www.intorobotics.com/types-sensors-target-detection-tracking/

[8] “Gps accuracy,” Oct 2016. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[9] R. MacLachlan, “Capacitive sensor introduction.” [Online]. Available: http://www.cs.cmu.edu/~ram/capsense/intro.html

[10] “Inductive proximity sensors.” [Online]. Available: http://www.balluff.com/balluff/mus/en/products/inductive-sensors.jsp

[11] “Arduino.” [Online]. Available: https://www.arduino.cc/

[12] “Raspberry pi.” [Online]. Available: https://www.raspberrypi.org/

XVI. Amendments

A. Methods of Object Recognition: 2/08/17

This amendment reflects a change in the methods of object recognition section of the document. The selection part of the section needed to reflect the changes in the obstacle detection system for the project. We have decided to not use an ultrasonic sensor for the obstacle detection so it needed to be removed from the section.

B. Parachute Deployment: 2/11/17

This amendment reflects a change in the parachute deployment section of the document. Originally we were going to use a timer, but we have decided to use the GPS altitude sensor instead.

C. Find and Touch the Finish Pole: 2/11/17

This amendment reflects a change in the finish pole section of the document. Originally we were going to drive in a circular pattern to hit the pole, but we have decided to use object recognition with a camera instead.

D. Getting Unstuck When It On Its Side: 2/12/17

This amendment reflects a change to the design of the system to get unstuck from it lands its side. We are no longer using the programmed self-rescue method, but instead a servo driven mechanical arm will be placed on top of the rover, when needed it will push the rover up.

E. Payload Fairing: 2/12/17

This amendment reflects a change to the design of the system on the payload fairing method. The original two half method is not cost effective in this case anymore, instead we will use the second choice plastic wrapped method, we believed that this method has the most cost-effective fashion and will get its job done.

VI. Weekly Blog Posts

VII. Poster

ARLISS A Rocket Launch for International Student Satellites

The Goal

Working alongside Electrical and Mechanical Engineering capstone teams, our task was to design and build a small rover that fits inside a standard 12 oz. soda can. This rover will be dropped from a rocket at 12,000' AGL, land safely on the ground and drive itself to a predetermined set of GPS coordinates.

As the Computer Science team, our job was to develop the software package for the rover. This included being able to lock on to and drive towards the GPS coordinates, as well as, an obstacle avoidance system that uses a camera to detect and avoid rough terrain or objects such as rocks that are in the path of our rover.

Implementation

We have implemented the following tasks:
- Parachute Deployment
- GPS Navigation
- Obstacle Avoidance
- Getting Unstuck From Obstacles
- Hitting the Finish Pole

In addition, many tasks were implemented utilizing the onboard camera, which we had to create obstacle detection software for

Parachute Deployment

Once the rover is dropped from the rocket, we had to determine at what point to deploy the parachute. We accomplished this by tracking the GPS coordinates height, and deploying the parachute once we dropped below a certain altitude. This way, the wind won't carry us as far as if we deployed the parachute immediately.

GPS Navigation

For the rover's GPS Navigation functions, we are using an algorithm that determines the shortest path between two given GPS coordinates. The GPS will also keep updating the new best route per request from the obstacle avoidance and unstuck from obstacles modules. This means that the GPS function has to work flawlessly with both of these modules to ensure the rover's safety and efficiency. How the rover behaves during it's driving is also critical, so the GPS function will check if the rover is off-course every few seconds and give route compensation if needed.

THE ARLISS PROJECT

Autonomously navigate to a target destination while using obstacle avoidance.

Obstacle Avoidance

The obstacle avoidance system ensures that our rover is not impeded on its way to the destination. Taking in filtered images from the obstacle detection software, this system does edge detection on the image to find objects in the rovers path, and then decides how to best get around the object. This is done by treating the filtered black and white image as a matrix of pixels, and summing the number of edges to the left, right or in front of the rover and adjusting the direction of the rover to travel where the fewest edges are found.

Getting Unstuck From Obstacles

In case the obstacle avoidance fails, and we end up hitting an obstacle, we've developed an algorithm to help us get unstuck from what we hit. It works by first attempting to back up the rover, and if the rover doesn't move, back up in different directions until it does move. It detects if the rover has moved by checking the GPS coordinates. This algorithm works best if just the rover's path forward is blocked, but it can still easily move backward.

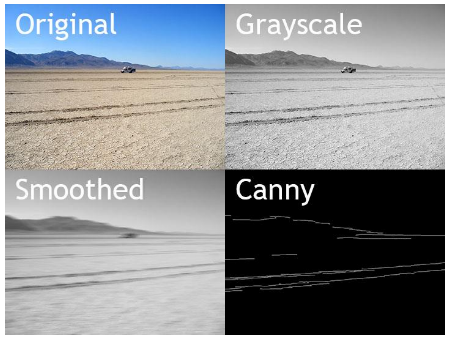
Find and Touch the Finish Pole

Once the rover get's within the GPS' error range of the finish coordinates, we have to search for the finish pole. This algorithm works by first searching for the finish pole by rotating in place and taking pictures. These pictures are used to detect a traffic cone by our imaging system. Once the cone is detected, the rover is oriented in the direction of the cone, and moves forward, making periodic course corrections along the way.

Computer Vision

Computer vision is used throughout the duration of the rover's expedition, but there are two separate and distinct functions which are being performed. During the stage of the expedition where the rover relies on GPS to direct it towards the target set of coordinates, computer vision will be utilized in the obstacle avoidance system. When the rover is approximately 8 meters away from the target, and the objective of the rover has switched from being directed by GPS to searching for the pole, computer vision will be used to locate the traffic cone at the base of the pole.

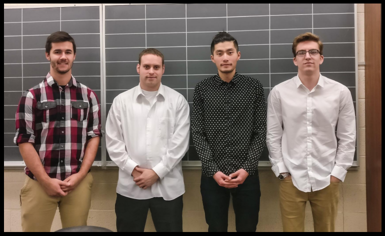
To detect obstacles in the path of the rover, each frame is run through a series of filters which are primarily intended to eliminate noise and to detect the edges of the obstacles.
1. Convert Original image to Grayscale
2. Smooth (e.g. blur) Grayscale image
3. Morphological Opening of image
4. Canny Edge Detection



To detect the pole at the end of the rover's expedition, our team has developed a complex system which utilizes machine learning and image processing. For the machine learning, our team trained a Haar Classifier which can be used to detect a traffic cone based on its features and geometry. Though effective at only detecting a traffic cone the majority of the time, machine learning classifiers are prone to returning false positives. The way we have avoided this is by verifying our results by checking the color.

Several points from the result, i.e. pixels, are converted to their HSV (Hue, Saturation & Value) equivalent, and are compared to values which would be typically observed on an orange traffic cone. Without the HSV values being confirmed, the image is rejected. This results in a higher rate of false negatives, but ensures accurate results.

Meet the Team



Team Members

- Zachary DeVita devitaz@oregonstate.edu
- Zhaolong Wu wuzha@oregonstate.edu
- Paul Minner minnerp@oregonstate.edu
- Steven Silvers silverss@oregonstate.edu

Our Client

Dr. Nancy Squires

Senior Instructor of Mechanical Engineering at Oregon State University

squires@engr.orst.edu

Sponsorship

This project was made possible through funding provided by Oregon State University AIAA. To find out more about AIAA, follow the QR code or visit http://groups.engr.oregonstate.edu/aiaa/home

Oregon State UNIVERSITY

VIII. Project Documentation

IX. Learning Sources

X. What did we Learn

XI. Appendix 1: Code Listings

XII. Appendix 2

Steven Silvers

Signature

Date

Zhaolong Wu

Signature

Date

Paul Minner

Signature

Date

Zachary DeVita

Signature

Date

Nancy Squires

____

Signature

____

Date