Senior Design Final Report: Autonomous Rover

ROGUE TWO

Team Members: **Brian Buu, Daniel Deaton, Rain Gopeng, Ryan Morris, Adam Olivera, Buse Ozsuca, Nicholas Payne, Madeleine Rasche, Jose Tomimatzu, Alvaro Valera-Rivera**

Sponsor: **Self-Funded**

Department of Electrical and Computer Engineering
San Diego State University

Spring 2018

TABLE OF CONTENTS

ABSTRACT

There are situations, such as during a natural disaster, when human eyes are needed, but it is not safe for a human to physically go to a scene. This problem calls for the development of new technologies, and this project aims to answer that call. The rover detailed in this report not only provides the crucial ability for a person to view a scene remotely via first person camera, but also has the capability to return to a designated home base autonomously when triggered to do so. Through careful planning and design, this rover brings forth new opportunities to see that which under normal circumstances, cannot be seen.

INTRODUCTION

The rover detailed in the report below aims to answer the question of how do you see something without being physically there to see it. This question arises in such situations as natural disaster or war, and during such times of stress and danger, technology becomes a crucial component in the safety and success of those working to keep others safe. The rover discussed here is designed to be a person’s eyes in these arduous scenes, with the added capability to return to a designated home based with triggered to do so, thus allowing a person to go perform other tasks while the rover comes home.

The rover is both functional and practical, with an simple user interface and easy to use controls. The scene can be viewed from a remote computer via the camera mounted on the front on the car. The car is then steered with a Xbox game controller, a choice made to give the user the maximum amount of control over the car rather than relying on the limitations of a keyboard. The car itself is designed to be durable and flexible, a strong platform holding up the CPU and other components, while wheels designed for traction are attached to springs to allow the car to take on rougher terrains.

This project was chosen with the hopes of creating something useful, while also learning valuable skills in the field of robotics and automation. The world as a whole is becoming increasingly automated, and there is an increased need for robotic technology. Thus, this project fits perfectly within the world's current demand, making it a perfect project for graduating engineering students.

Overall, this project aims to create a useful tool for the most difficult of situations, while also providing a means for learning.

BODY

Competition

This project was completed partly as a competition with another team creating another autonomous rover. Thus, there were certain agreed upon specifications and terms, both for the design of the rover itself and the rules and setup of the competition day.

First, the rover itself must not exceed 1.5 cubic feet in size, and must be under 50 pounds. The rover must be able to drive or be driven for a minimum of 15 minutes before needing a new battery. The challenge also specified that the maximum budget for the project is $1000.

On the day of the challenge, there will be a field of 8 boxes placed on a field that is 30 feet by 30 feet (see Figure 1). The boxes will be 12 inches in length and width, 18 inches tall, and at least 4 feet apart. They will be placed perpendicular to the starting line as well. Each team will have three attempts to make it across the field and back, with a timer running for each run. The car will be driven to from home base to the other other side of the field by a driver who is facing away from the field. Then the car will return autonomously to the home base. The person with the best time wins the competition.

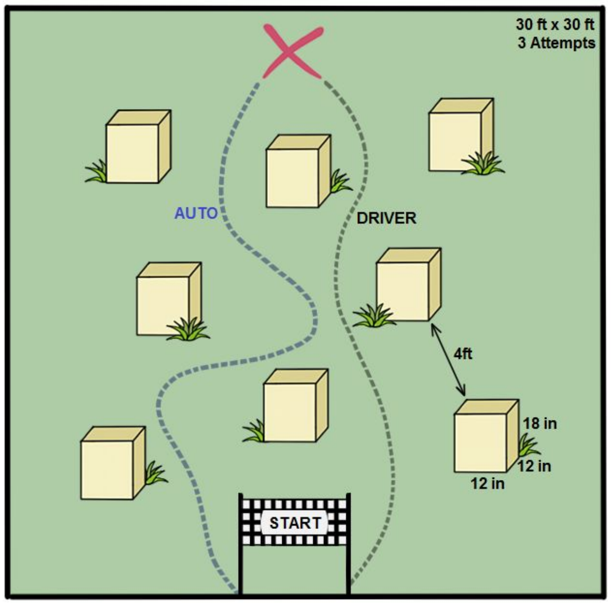


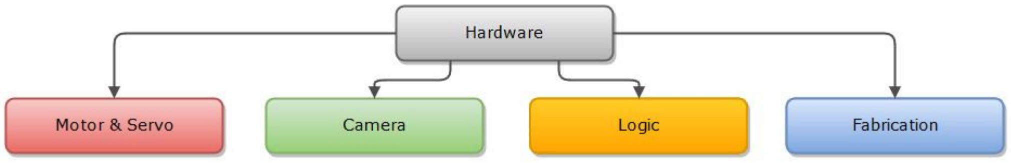
Figure 1: Field setup for competition

The goal of this competition is to inspire innovation, and to provide an added sense of motivation to succeed. By two teams building the same thing, there is a greater chance for new things to be discovered and learned, each approach bringing something to the table.

Hardware

General

The basis for the rover was a pre-built RC Car that was then restructured to fit the needs of the project. A Raspberry Pi 3+ is the primary CPU device, and additionally the other primary parts of the project are the Lidar sensors, the Xbox controller, and camera. Below is a basic diagram of both how hardware was broken down for assigning and building, as well as a block diagram of the basic hardware setup.



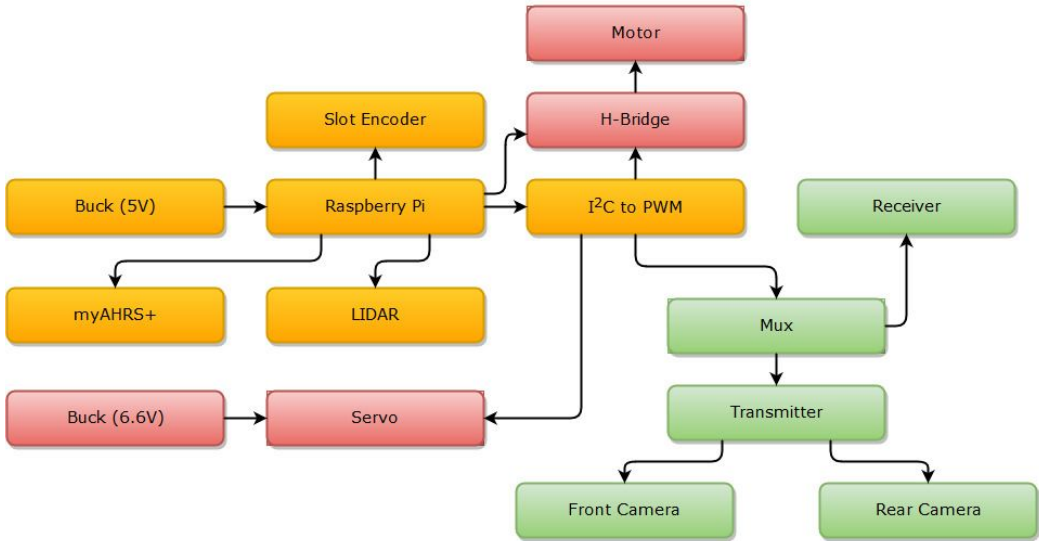


Figure 2: Hardware Block Diagram

Logic Section

Buck Converter

In order to obtain the various voltages that were necessary we utilized a step down buck converter. The selected buck module was able to handle input voltages of 4.5V - 30 V, output 0.8V - 30V (continuously adjustable) and supply an output current of up to 12A and 100W. The buck converter has short circuit protection (14A max) as well as thermal shutdown. The module has an aluminum radiator to dissipate heat, as well as a high frequency low resistance electrolytic condenser to make the output ripple lower.

Slot Encoder

In order to facilitate tracking we installed a Slot Encoder in the motor which operates at 16mA and can handle up 28 VDC input, it has a response speed of 3 kHz.

I^2C to PWM

We decided to use an adafruit 16-channel 12 bit PWM - I^2C interface module (PCA9685). The I^2C controlled PWM driver has a built in clock that allows it to run without having to send a continuous signal from the microcontroller. It has a 4 µs resolution at a 60 Hz update rate.

AHRS

The myAHRS+ is a high performance AHRS(Attitude Heading Reference System). Its attitude output is more stable to acceleration and magnetic disturbances. Communication and configuration are enabled via UART/USB interface for user applications. It features a Triple axis 16-bit gyroscope (± 2000 dp), Triple axis 16-bit accelerometer (± 16 g), Triple axis 13-bit magnetometer(± 1200 µT).

Lidar

For the project we used the Grove-TF Mini LiDAR, This product is based on ToF (Time of Flight) principle and integrated with unique optical and electrical designs, so as to achieve stable, precise, high sensitivity and high-speed distance detection. its operating principle is as follows: a modulated near-infrared light is sent from the sensor and reflected by an object; the distance to the object to be shot can be converted with the sensor by calculating the time difference or phase difference between the light sending and the light reflection, so as to produce the depth information.

Motor & Servo Section

Battery

The battery requirements for this robot where 5 V at peaks of 7.5 A. The amperage is high because it would drive the motors and servo of the robot and would not be continuous. The capacity of the battery was chosen as follows:

$$\frac{capacity\ (Ah)}{load\ (A)} = battery\ life\ (hr)$$

It was previous stated that the battery life would run for 30 or more minutes or 0.5Hr, and plugging in the peak load capacity →

$$\frac{capacity\ (Ah)}{7.5\ A} = 0.5\text{hr}$$

Solving for capacity (Ah) →

$$capacity\ (Ah) = (0.5\ hr)(7.5\ A)$$

$$capacity\ (Ah) = 3.750\ Ah$$

$$capacity\ (Ah) = 3750\ mAh$$

We need a battery that is 3750 mAh, but manufactures do not make batteries of that value so we rounded up. A 4000 mAh; 30C was selected. The C in the battery is a “C rating”. What that means is the current. The battery can safely discharge 120A, but generally one does not want to discharge at the max rate because there is internal resistance of the battery, thereby heating up the battery and have a lower overall efficiency, furthermore, lowers battery capacity at higher discharge rates.

Motor

The motor employed is an 80 turn class 540 brushless DC motor. By using a brushless motor with high number of turns we sacrifice speed for torque. We required higher torque in order to properly move the rover and its increased weight. The higher torque enabled us to maneuver through the rough terrain of the field.

H-Bridge

The purpose of an H-bridge is to switch polarity on the motor power inputs, and doing so reverses the direction of the motor spin. To do this, a set of switches are enabled at the same time another set of switches is disabled. These switches connect the output voltage of the H-bridge to the motor. This allows for forward and reverse driving of the rover. The H-bridge utilizes a full two half-bridge driver chip with low resistance N-channel MOSFET components. The configuration of these MOSFETs allows for minimal losses. The MOSFET driver chip allows for a hardware brake function and power feedback. A top tube connected to the half-bridge driver chip utilizes a bootstrap capacitor which allows for enough drive voltage of the MOSFET to perform quick opening of the channel. This also improves the acceleration of the motor and can also allow for quick braking. The driver can operate between zero and ninety-nine percent of the duty cycle of the input PWM.The physical dimensions of the H-bridge are 4.25” x 2.16” x 0.71” with a net weight of 2.01 oz. The H-bridge is capable of receiving two different PWM inputs to control two seperate motors, although our rover only needed one motor, so two of these pins are unused (the other unused pin is the direction pin for the second motor). The H-bridge pins utilized include a direction pin which drives the motor forward when the pin receives a logic 1 and feeds the motor with a pattern received by the PWM pin. The motor reverses when the direction pin has a logic 0. To enable the parking brake, the direction pin can be arbitrary, but the received PWM must be 0. Finally, the H-bridge contained a voltage input on the logic side (5V) as the high voltage reference, and output 7.2 V on the motor side. Two pins each were dedicated for connecting to the motor as the +/- terminals. A final pin was connected to ground on each side. The H-bridge is capable of handling 30A instantaneous currents and 15A sustained currents and also provides undervoltage protection.

Servo

The servo employed replaced the native servo to the chassis. Instead of a polymer gear mechanism, the new servo has full metal gear and an increased number of splines (25 at 6mm in diameter) as well as an improved gear ratio. The casing of the servo is lightweight aluminium which also assists in heat dissipation. Additionally, a higher torque rating was needed to overcome the static friction generated by the additional hardware that was mounted to the chassis. The servo possesses capability of 20 kg-cm torque output. A 6.6 V input is required for the servo and this was achieved through the output of buck two. The physical dimensions of the servo are 1.57” x 0.78” x 1.59”. The operating range of motion for the servo is 180 degrees, with a precision of 0.3 degrees. The servo has a three wire input with the black wire serving as ground, red for voltage positive, and white for PWM. The overall weight of the servo is 65 grams and can turn at a rate of 60 degrees per 160 milliseconds. It requires a 1A input in operation but only requires 100 mA in standby.

Camera Section

Transmitter and Receiver

We decided to use an TS832 transmitter and RC832 receiver for the first person view. In order to increase video and audio signal we used a Clover-Leaf style Omni-directional antenna.

Camera

We used two 700TVL 2.8mm 120 degree lens camera. Video switching from the front and rear camera was achieved through the video switcher(multiplexer). This was done by sending a PWM signal to the video switcher to switch from the front and rear facing cameras. The XBOX controller was used to send the PWM signal. A button the controller was used to toggle between 20% PWM and 50% PWM signal.

Fabrication and PCB

Custom fabrication of the aluminum chassis as well as the surrounding connector pieces was necessary in order to ensure that all of our hardware pieces could be properly mounted. We chose aluminum because it is light, inexpensive, and easy to manipulate. Over the center of the rover is an aluminum plane where both buck converters, Raspberry Pi, and I2C to PWM board have been mounted. In the center of this aluminum plane is an additional aluminum pole mounted in order to prop up the first person view camera as well as three antennas for external communication through wifi. If the first person view camera is not mounted about the car there would be some problems being able to reference where the actual rover may be under user operation. Keeping the antenna mounted above the rover also ensures less disruption over our wifi network.

In the front of the rover is another custom made piece of aluminum mounted with two LiDar sensors attached to both sides. The same configuration is present on the back side of the rover with a piece of aluminum supporting two additional Lidar sensors. The front two sensors are angled directly in front of the rover while the back two sensors are angled slightly outwards in order to detect objects approaching the rover at the side. The final pieces of aluminum are on the bottom side of the chassis where the two back wheels connect to the base. We replaced the springs on the two back wheels with two solid pieces of aluminum. The reason behind this was to ensure the rover stays in a fixed position as it is being remotely driven to our destination.

The PCB was developed using KiCAD software. In order to develop a PCB that would withstand the rated current loads, the trace width needed to be thick enough. Pictured below is the final design of the PCB including both top and bottom layers. The ground and power plane takes up most of the board so as to save space on the board and taking into consideration the limitations of the LPFK routing machine on campus at SDSU.

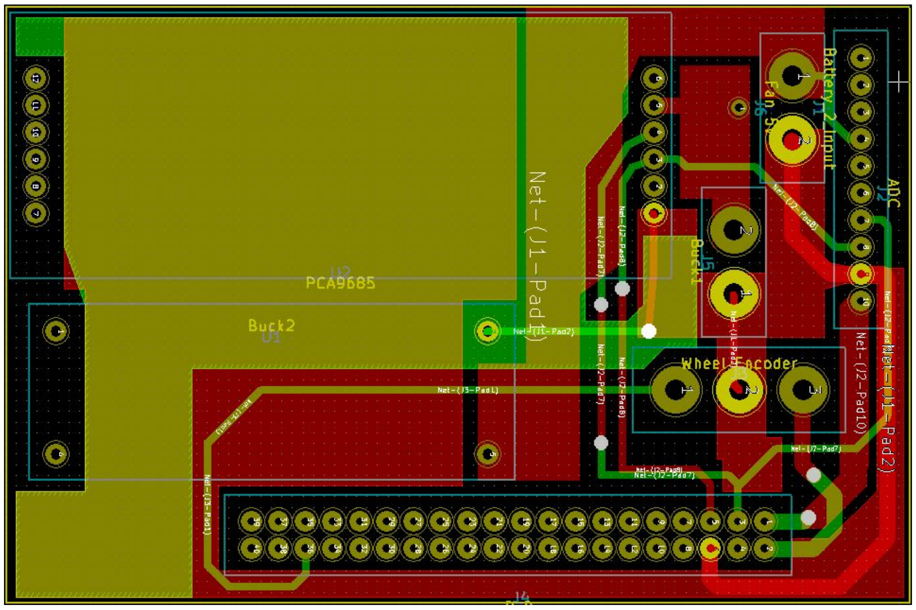


Figure 3: PCB Layout, Top and Bottom

Software

General

The software section was designed with automation in mind. Thus, one of the first choices made was to use the Robotic Operating System (ROS), which is “a flexible framework for writing robot software” [1]. It is ideal for the hefty task of automation, as “It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms” [1]. The system allows us to “publish” and “subscribe” to data, making data passing between nodes simple. It also allowed for a simulation of the rover to be created with ease. It runs on the Ubuntu Linux operating system, which is installed on our hardware of choice, the Raspberry Pi. This system gave us the tools and flexibility to aim for an elegant system, even as novice robotic designers.

Raspberry Pi Versus ODROID

One of the biggest obstacles to overcome during the course of this project was a misstep that was taken right at the beginning. A CPU device called an ODROID, model number XU4, was chosen to be the primary component of the project due to its high processing power. With this high level of processing power, we hoped that the automation would be improved, allowing for quicker response times and greater control. However, we ended up having to make a switch to the Raspberry Pi very last minute due to a lot of unforeseen issues with the ODROID.

For starters, the ODROID XU4 does not support PWM. PWM is crucial to the project, as it is the method by which the car is steered. A specific shifter board was purchased for the ODROID to help with the PWM issue, but since the system itself was not really intended to produce PWM signals, a lot of problems arose. To combat this, at one point an Arduino was attached to the ODROID to produce the signals, but it started to be apparent that it was going to take more than one Arduino to take care of it. The ODROID additionally does not have a large online support system the way other CPUs do, so research was difficult and information was hard to come by.

With this is mind, as the ODROID continued to make simple tasks complicated, a decision was made to abandon the ODROID and the Arduino for a Raspberry Pi 3+. While it does not have as much processing power as the ODROID, it can produce PWM signal, and in general is just a more user friendly, better documented, more frequently updated system. The transition was a fairly smooth one, as the Raspberry Pi 3+ supports ROS, and in general is just a user friendly device.

Software Architecture

Within ROS, the software was programmed in C/C++ as well as Python. C/C++ was used exclusively for navigation and perception, while Python was used for non-CPU intensive modules. In addition, since each node is node is ran as a separate process then all cores of the raspberry pi could be taken advantage of. This allows the linux operating system to assign the cores and choose the task scheduling. By choosing specific languages for specific tasks, the software was optimized to take advantage of the the differing benefits of each language. Additionally, Gazebo was also used for producing simulations of the rover, allowing the project to run with hardware and software working in parallel. The simulations created utilizing ROS allowed the project to be jump started, as well as for a wider range of freedom to make mistakes without damaging any hardware.

The general layout of the software can be seen below, consisting of a navigation system, a perceptions system, a controls system, and a hardware communication system (see Figure 4). ROS has a tool that can plot all the nodes in the system and the connections between them. To see the complexity of our system at a lower level please refer to Figure 5.

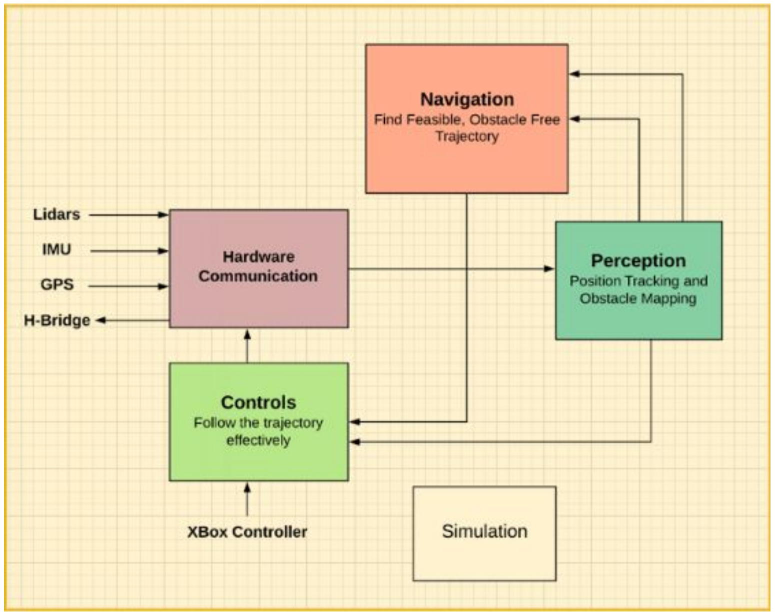

Navigation
Find Feasible, Obstacle Free Trajectory
Lidars
IMU
GPS
H-Bridge
Hardware Communication
Perception
Position Tracking and Obstacle Mapping
Controls
Follow the trajectory effectively
XBox Controller
Simulation

Figure 4: Software design block diagram

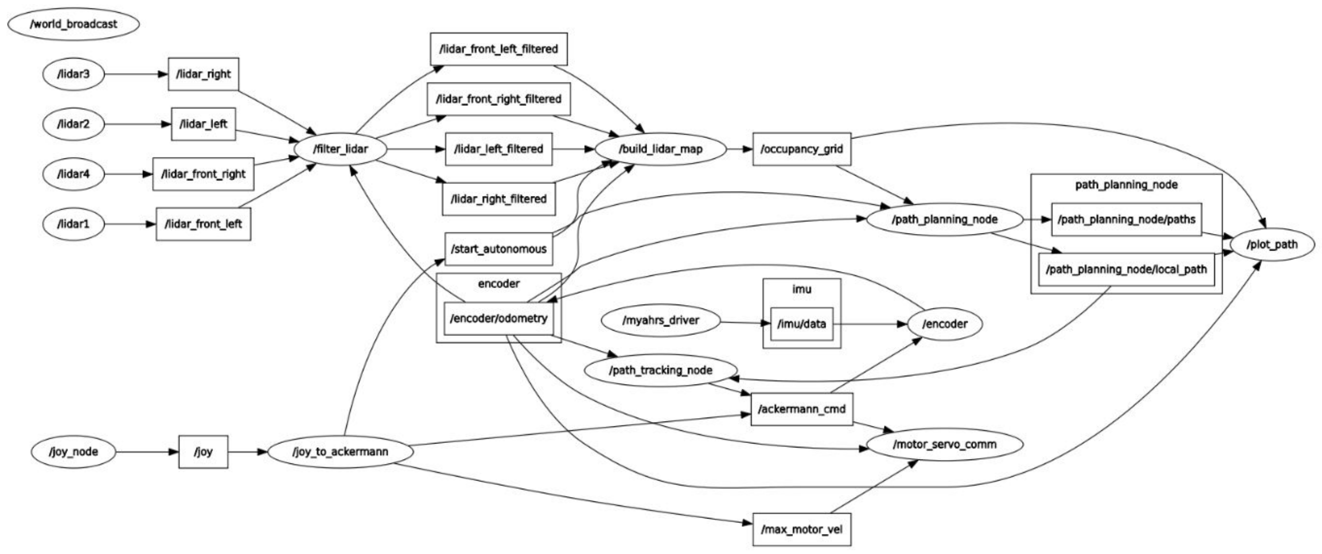


Figure 5. Low Level System Architecture

Dynamic Path Planning

To achieve obstacle avoidance dynamic path planning was used to planning an obstacle free trajectory given a map of the obstacles and the vehicle's current position. We wanted the vehicle to be able to handle any obstacle and to do so in an intelligent way. The dynamic window approach to collision avoidance was used. This approach creates a window of possible trajectories around the vehicle where the width is max yaw rate plus min yaw rate and the height is max velocity plus min velocity. For a visual of the dynamic window please refer to Figure 6 provided by [4].

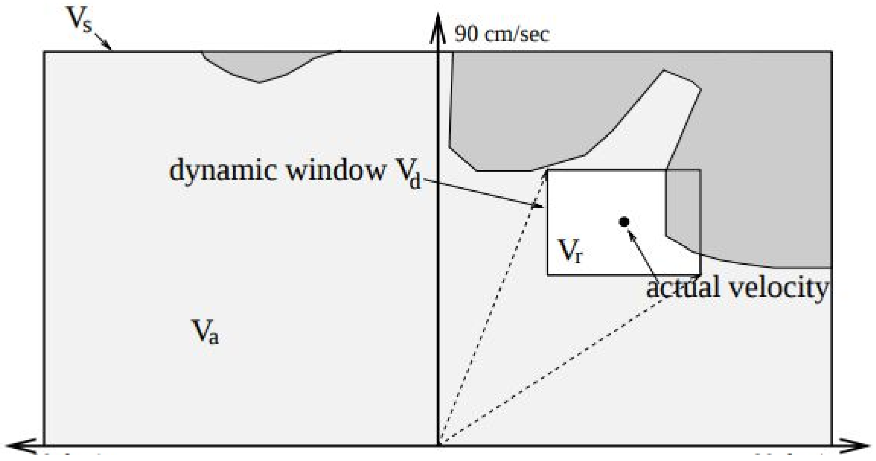


Figure 6. Dynamic Window

Once the dynamic window is calculated every possible trajectory within that window is weighed. The trajectory is created by estimating the robot's position some time ahead. Given the yaw rate and velocity from the dynamic window the robot's position at each time step can be calculated using Equations 1-3. The robot's x, y, and yaw position is stored into a vector for each time step. That vector is called the robot's trajectory.

Equations 1-3.

$$x \mathrel{+}= (velocity * cos(heading)) * delta\ time$$

$$y \mathrel{+}= (velocity * sin(heading)) * delta\ time$$

$$yaw \mathrel{+}= yaw_{rate} * delta\ time$$

Now that the robot's trajectory is generated, it must be given a cost. The total trajectory cost is the sum of three separate costs. The first cost is called distance to goal cost. The cost is calculated by adding the heading difference between the end of the trajectory and the goal point plus the distance from the trajectories end point to goal point. This cost forces the robot to continually head towards the goal. The second cost is called the speed cost. This is simply the difference between the velocity of the robot at the end of the trajectory minus the max speed. This cost makes sure that the robot will take trajectories that keep up. The final, and most important cost, is the to obstacle cost. This cost is the distance to the closest obstacle and it allows the robot to take swift maneuvers around the obstacles. Each cost is then weighed to give more importance to specific costs.

After looping through all the trajectories, the lowest cost trajectory is used as our path. For our robot we used the weights 0.1, 0.1, and 0.8 for the to goal cost, speed cost, and obstacle cost. This means our robot heavily favors trajectories that have the largest clearance from obstacles. We found from testing that decreasing the obstacle cost or increasing the to goal cost would make the robot clip the obstacles or turn to late. Also, when in the grass the vehicle doesn't respond well to small steering angles, so the robot had a hard time follow trajectories with conservative yaw values. Figure 7 shows the output of the path planner.

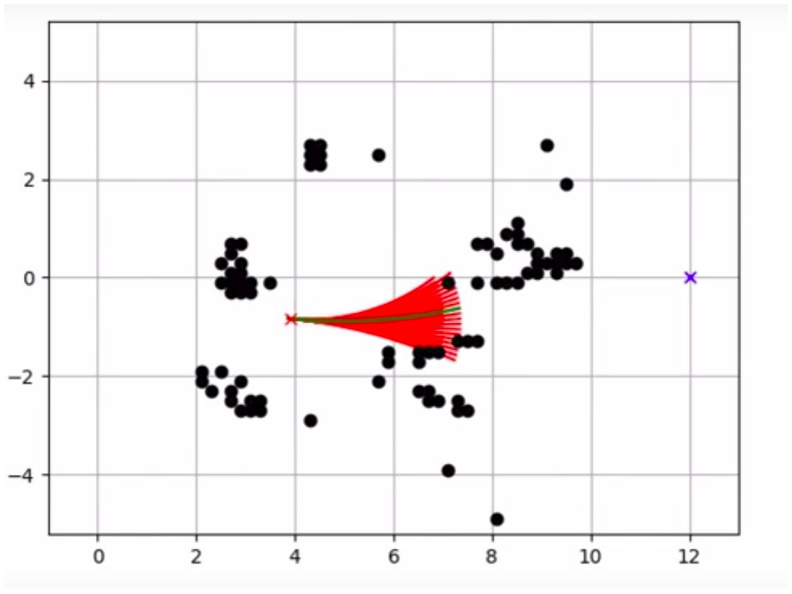


Figure 7. Red lines are possible trajectories. Green line is lowest cost trajectory.

Path Tracking

The path planner gives the path tracking node a vector of x, y, and yaw states of the vehicle. It is the path tracking node's job to compute a steering angle to follow that trajectory. The path tracking node using the Stanley Method, which is a nonlinear feedback function of the cross track error measure from the center of the front axle to the nearest path point. Equation 4 shows the steering control law equation.

Equation 4.

$$\delta(t) = \theta_c(t) + tan^{-1}(\frac{ke_{fa}(t)}{v_x(t)})$$

For a better visual of the control law please refer to Figure 8. The first part of Equation 4 simply keeps the wheels aligned with the given path by setting the steering angle to the heading error. The second part adjusts the steering angle so that the intended trajectory intersects the path. As the cross track error increases the wheels will be steering further towards the path.

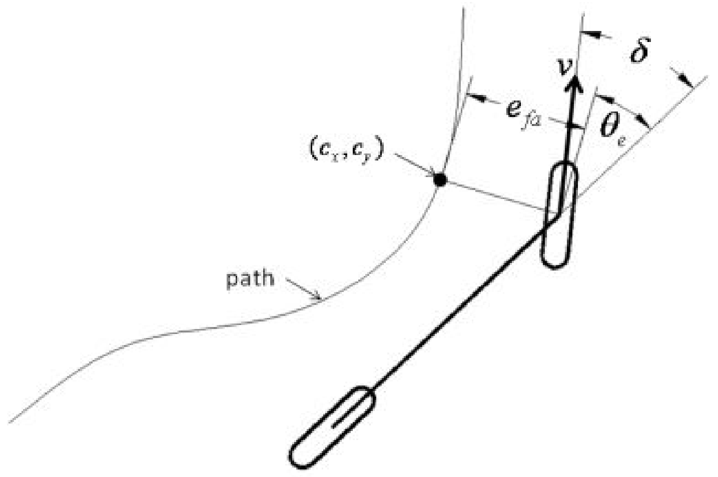


Figure 8.

Brute Force

The brute force method of autonomous navigation was developed in parallel with the dynamic path planning as a backup. It is a much less robust way of avoiding obstacles, but also much easier to code.

The program starts by using the Yaw and coordinates to orient the rover towards the home destination. The rover is sent a speed command to move forward, and checks for boxes with the front lidar. Once the front lidar detects a box, the rover turns based on which side is closest to goal.

Next, the program check for when the side lidar detect the same box. When the side lidar detect it, the rover is sent gradual turning commands to slowly correct itself. Finally, once the side lidar lose track of the box, the program considers the rover as being past the box. At this point, the program goes back to heading towards the goal. The rover stops when it is within half a meter of it's destination.

Hardware Communication

One of the crucial components to the automation scheme was the ARHS or Attitude and Heading Reference System. This system provides vehicle heading, pitch, and vehicle acceleration, all of which allow us to track the state of the rover. This system is similar to an IMU, however “The key difference between an inertial measurement unit (IMU) and an AHRS is the addition of an on-board processing system in an AHRS which provides attitude and heading information versus an IMU which just delivers sensor data to an additional device that computes attitude and heading" [7].

The software is used in part to steer the car, putting out a PWM signal of the desired frequency in relation to either the controller input or the sensor input to the servo, motor, and camera. A 1 ms pulse will result in the car turning left, a 1.5 ms pulse will result in the car staying straight, and a 2 ms pulse will result in the car turning to the right (see Figure 9). There are mid range degrees by which it can be turned but these serve as the primary directions that can be chosen from. The PWM is generated using an I2C to PWM board. This was a leftover piece from when we were trying to get the ODROID to properly output a PWM signal that we decided to keep as it allowed for some of the processing to be offloaded from the Raspberry Pi.

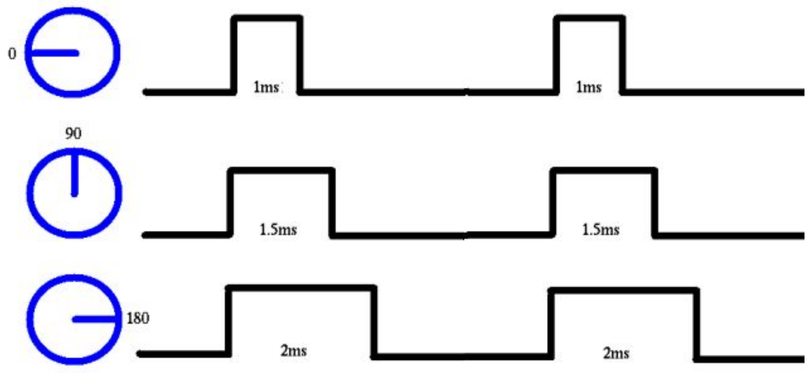


Figure 9: PWM signal to control servo motor [2]

Another crucial part of the automation process is interpreting the data from the Lidar sensors. This data is used to determine the next move the rover will make, turning away if it senses an object is near. This is discussed in detail in the section below.

Lidar Mapping

Detecting obstacles is the most vital part of the autonomous side of the rover. To do this we naively say that anytime our lidars read a value within a max and min distance then we will consider an obstacle. Once the mapping node gets the filtered lidar data then it converts the distance reading to x and y values from the vehicle's current position. To do this there are multiple transformations to account for.

The first transformation to account for is the vehicle's position. The vehicle's position comes from the encoder and the AHRS. The position is stored as a transformation matrix. Then we store the transformation of each lidar from the vehicle's center point. The dot product of the sensor's transformation from the vehicle and the vehicle's transformation from the start point gives us the senor's global transformation. Now that the sensor's global transformation is known we can take the lidar’s distance and using simple trigonometry to find the x and y distance the beam traveled can be calculated. With the x and y distance to the obstacle is known we store that information into a 2D occupancy grid. An occupancy grid is just a 2D matrix with width * resolution columns and height * resolution rows. The width and height is size of the map in meters and the resolutions is the size of each square in the grid. For our robot we used a width and height of 20 meters and a resolution of 0.20 meters. To symbolize an occupied square the index in the matrix is assigned a 100 and a 0 symbolizes empty space.

As the vehicle moves and more lidar readings come in, the map is updated with new obstacles. Then at a rate of 5 Hz, the map is published and the path planner is the subscriber.

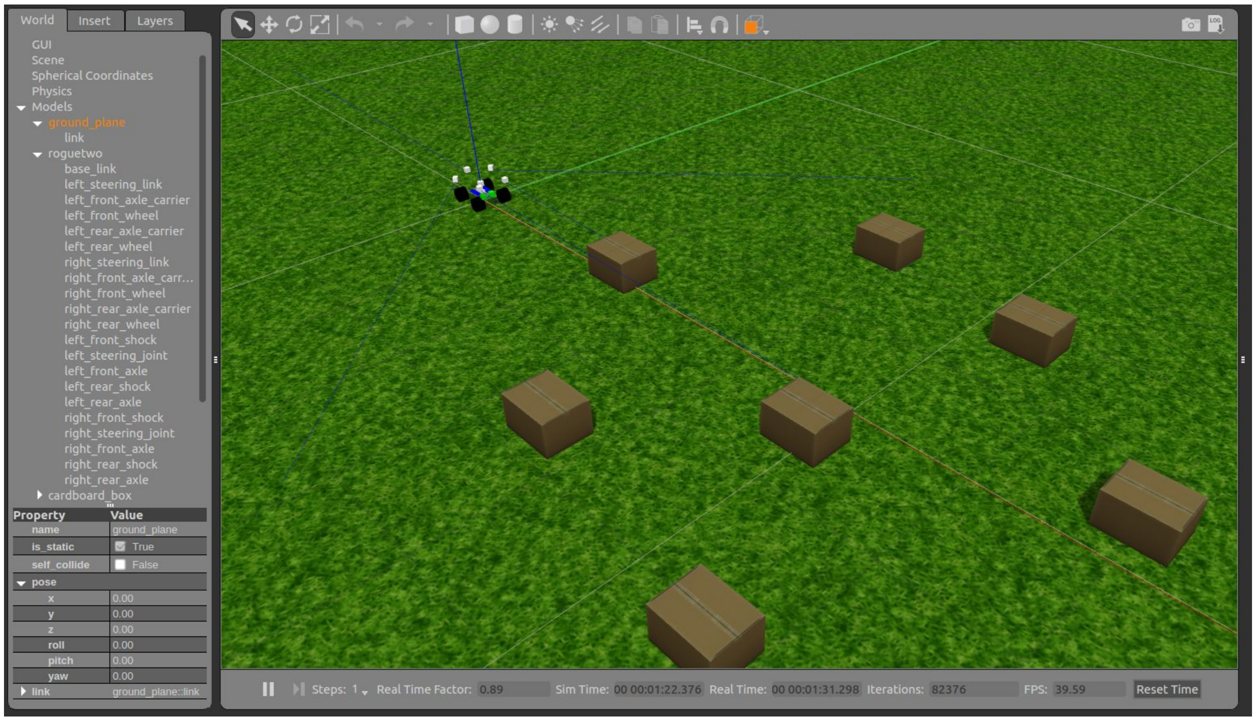
Lidar Sensors versus Ultrasonic Sensors

One of the major hardware changes occurred later in the project when the Ultrasonic sensors were switched for said LIDAR sensors. An Ultrasonic sensor “... measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back. By recording the elapsed time between the sound wave being generated and the sound wave bouncing back, it is possible to calculate the distance between the sonar sensor and the object.” [3] This technology however did not work perfectly for the project, as they also can only sense an object if it is directly in front of them, specifically perpendicular. This limits the cars ability to detect an obstacle and can lead to collisions. Additionally, Ultrasonic sensors has the added problem of only being able to run one at a time, slowing down the sensing speed. With this in mind, LIDAR was chosen as a replacement sensor.

Gazebo Simulation

A crucial aspect of our testing and development is the Gazebo Simulation software. The most beneficial aspect of this software, is the opportunity for the coders to test their program and logic without requiring an actual field test. This is very impactful during the initial stages of development, when we have no actual hardware to test with, allowing the software and hardware teams to work in parallel.

Within Gazebo itself, we are able to simulate the rover, sensors, cameras, and the boxes as can be seen in figure 10. The white boxes surrounding the rover represent the lidar, and the blue lines protruding outwards are the range. Gazebo also allows us to easily reposition the rover, and the boxes, allowing us to test quickly and efficiently, multiple times.



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
pose	
x	0.00
y	0.00
z	0.00
roll	0.00
pitch	0.00
yaw	0.00
link	ground_plane::link

Steps: 1 Real Time Factor: 0.89 Sim Time: 00 00:01:22.376 Real Time: 00 00:01:31.298 Iterations: 82376 FPS: 39.59

Figure 10: Gazebo Simulation software.

Position Tracking

At the very beginning, we have tried to use Visual Odometry. In order to use Visual Odometry, we had to determine the position and orientation of the rover. Thus, we had to analyze the camera images. Unfortunately, we ended up experiencing a few problems. Due the bumpy grass surface, the cameras couldn't handle the severe motion. As a result, they provided noisy values.

Instead, to track the position of the rover, we have used an encoder. The encoder allowed us to measure the travelled distance. In addition to that, we have used Attitude and Heading Reference System. Generally, AHRS is used to measure heading (yaw), pitch, and roll angles. In our case, we have used it to get the global heading. The calculations are shown below.

Equations 4-5.

$$x \mathrel{+}= distance * cos\,(heading)$$
$$y \mathrel{+}= distance * sin(heading)$$

For better position tracking, we have implemented Kalman Filter. The Linear Quadratic Estimation (LQE) allowed us to fuse the x and y coordinates from encoders with the IMU.

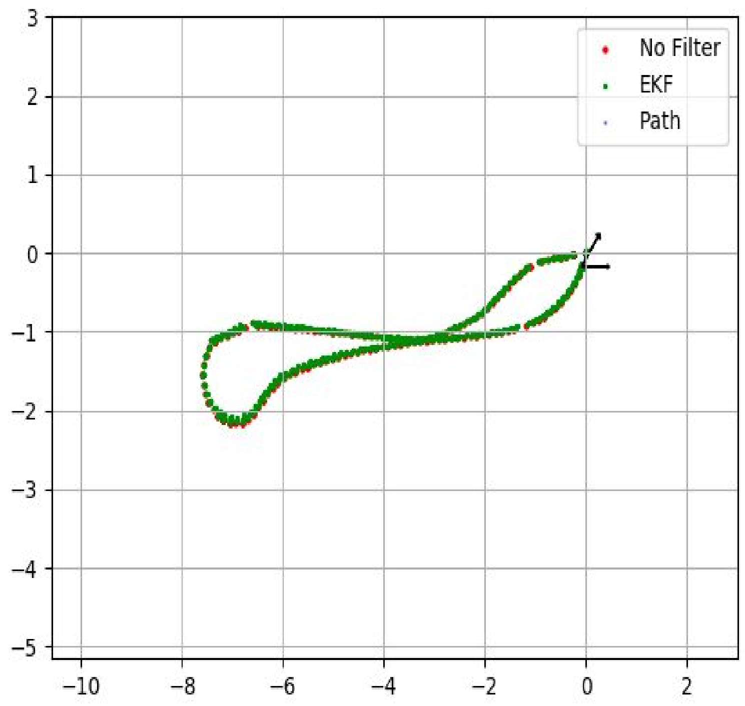


Figure 11: A visual representation of position tracking

Software Control

To make the user interface more friendly, we have implemented an XBox controller. This type of controller was used to give the driver the maximum amount of control. First of all, we have written a teleoperation node in C [5]. From the base station, the commands were sent over the wifi. The steering angle and velocity values were mapped to pwm range. Later, this range was sent to the motor and steering servo.

The XBox controls were programmed based on the video game *Need for Speed*, a car racing game, and though not all of the buttons do the tasks shown below (see Figure 12), it was still a logical choice to base our controls on the controls of a virtual car.

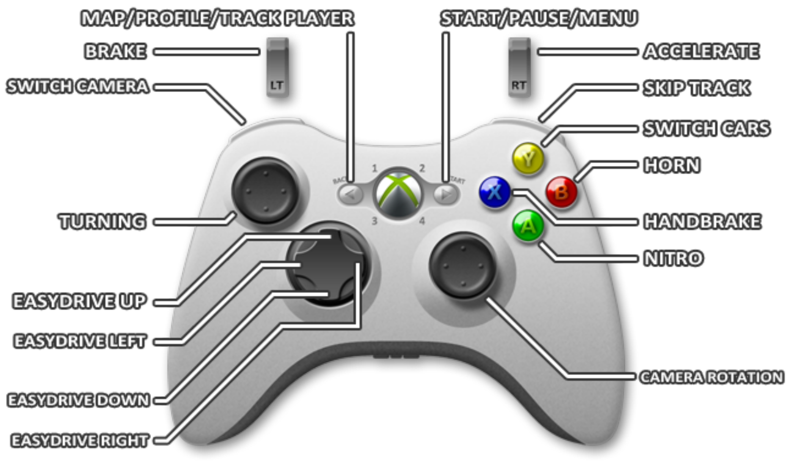


Figure 12: Need for Speed controls on Xbox controller

RECOMMENDATIONS & CONCLUSION

Overall this project was a success. A rover was successfully built and programmed to be driven with the Xbox controller via first person camera, and the rover did return to the home base autonomously. The precision of how close it gets to the home base varies, but it it always in line with the homebase, i.e. it makes it back to the homebase side of the field. Our rover makes in across the the field in a matter of minutes if not seconds when driven, and makes it back fairly efficiently when in autonomous mode.

The rover fulfilled the competition guidelines coming in at 18” x 14” x 17” and 5 pounds. The competition itself was a draw as ours had the better overall times, but did not come as close to homebase as the other team. This is a area where the rover could be improved, by finding a more precise method of identifying homebase. The rover could use some fine tuning in this regard.

Some recommendations for anyone trying to recreate the project include doing deep research early on on the chosen CPU. Make sure that everything actually does what it appears it will. If this had been done at a deeper level earlier on, some hardware changes would not have to have been made. It is also good to be flexible when those types of changes arise.

The rover could also possibly be suited to take on additional environments, making it waterproof, and perhaps able to take on even rougher terrains than just uneven grass. There also should be a means to make sure that the sensors do not get overwhelmed when there is a lot of data point (obstacles) around it.

This project was a learning experience for everyone involved, and provided a great set of skills and insights into the world of robotics and automation. There are so many potential applications for the skills and techniques in the project, including space exploration. NASA states “Technologies for autonomous planetary mobility enable the rovers to make decisions and avoid hazards on their own” [14]. This is incredibly important for space exploration as often it is not safe or feasible to send a human being to places like Mars (at least not yet). Besides all of the applications on Earth, this rover could quite literally be expanded upon to see the stars.

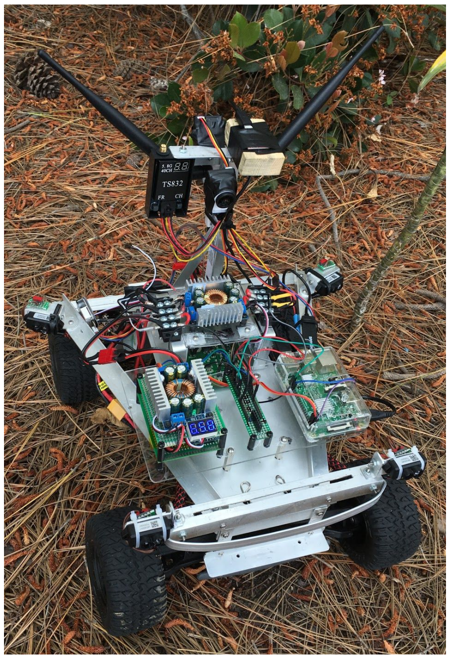


Figure 13: Final Rover

Demo: https://www.youtube.com/watch?v=zXMh1ZJNL9w&feature=youtu.be

SPECIFICATIONS
Size: 18” x 14” x 17”
Weight: 5 lbs
Robot Class: Remote controlled, Autonomous
Power: Powered by battery, able to supply 6.6 V and 5 V
Features: First Person View Camera, able to return home autonomously, avoid obstacles autonomously.

TEAM MEMBER PAGES

Brian Buu

Our team was divided up into software and hardware groups to split up the workload and to get tasks completed faster. After conference, the initial plan was made, and we were assigned our teams. For the senior design project I was the on the hardware team and had the job of purchasing.

One of the task I was assigned was finding a battery that met our requirements and specifications. We needed a battery for the robot that could run at 5 V and peaks of 7.5 A. The amperage is high because it would drive the motors and servos, but would not be continuous. The capacity of the battery was found by taking into account the 15 minute run time and doubling it to 30 minutes for a safety margin. The time multiplied by the amps gave us our capacity of 3750 mAh, but since manufacturers do not make batteries of that size, we elected to pick the 4000 mAh with a 30 C rating of; so the discharge rate could be at max 120 A.

Video switching was achieved by a video multiplexer (switcher) via a button the XBOX controller. The video switcher was tested by me and Jose. We attached the video switcher to the front and rear camera, the input to the RC832 receiver, and the output to the TS832 transmitter. We sent a PWM signal to the switcher at different PWM percentages via an Arduino code. The maximum PWM was 4136, so we kept changing the till we got the cameras to switch, which was around 830 and 2200, approximately 20% and 50 percent. The controller was programmed by Ryan to get the video to switch via a toggle button.

Another thing I did was the 3D prints for various parts of the rover, like the holders for the ultrasonic sensors and the camera holders.
Originally, the plan was to use the ultrasonic sensors to be use to avoid objects like the boxes in the field, but ultrasonic sensors could not be used because it was found out through testing that they could not see objects like the boxes, if they were at either a parallel or a right angle to the sensors. In the end, the LIDAR was chosen, they are more robust and fit our needs for object detection and object avoidance, but it came at a higher price cost.

Working on this project let me has the opportunity to work on a team and demonstrate the skills and knowledge I learned at San Diego State University. I learned how to budget time, work with others, and perform other task outside of my normal duties. I assisted in field and lab testing, and writing this report. I initially wrote the introduction and the abstract, but it was later rewritten by Madeleine. I also made the prototype for the competition map for the contest, but it was later redone with better graphics for the final presentation and competition by Buse.

Daniel Deaton

I am assigned to the hardware group and helped with fabrication, testing, design, and research. In the initial part of the project I helped with the decision of parts and design of the project. The project initially utilized an Odroid XU4 with ultrasonic sonars for object avoidance, which was later changed to a Raspberry Pi 3B and utilized lidar sensors for object avoidance. After the initial planning was finalized, I helped with designing and fabricating the aluminum frame which harnesses the GPS unit, forward and reverse cameras, two buck converters, H-bridge, Raspberry Pi 3B, power distribution board, and a servo driver.

The RC car's frame had very loose shocks. This posed a problem due to the extra weight that had to be placed on the vehicle due to the hardware and extra battery. Jose, Alvaro, and I replaced the back shocks with aluminum struts to make sure the car didn't sink to far with the added weight. The base of the frame is taken from the top part of a piece of aluminum raceway molding. Jose bought some aluminum L pieces to use for the side rails and the post for the cameras and AHARS. I helped fashion the rails, base, and post with Jose and Alvaro. I also performed research on the Odroid board and implementation of PWM.

I was tasked with creating test code to run the servo and the motor of the rover. My objective was to make sure the code and hardware would work well when the COMPE side integrated their code to the Odroid. To accomplish this task, I researched the specs of the Odroid and the implementation of PWM with the board to the servo and motor. After much research through google searches and message board inquires, I discovered that the Odroid does not have a hardware PWM and that a software PWM would task the Odroid's processor far too great while running the visual odometry code. I also discovered that the GPIO ports are 1.8Volts and the ADC inputs are limited to 1.8Volts. I suggested the Adafruit 16 channel servo driver as a solution to the PWM issue and the XU4 Shifter Shield for the low voltage of the GPIO ports. I began researching how to implement the servo driver with the Odroid board and ran into issues with the communication with the servo driver board through I2C protocol. The only tested and approved Linux version that works for both the Odroid and the ROS operating system is Ubuntu 14. I discovered the version of Ubuntu did not work with WiringPi and it caused issues with I2C communicate. After much deliberation with Jose, Ryan, and Alvaro we decided to go with the Raspberry Pi 3B. The Raspberry Pi 3B can use Ubuntu 16 with ROS, which worked well with our libraries and hardware. I wrote some skeleton code for the servo in C and found some example code in Python, but my code was not used. After the COMPE team finished the code for the servo and motor, we tested the rover out in the field.

I helped Jose, Alvaro, Ryan, and Rain performed 6 tests out of the twelve tests that were performed. I also helped test the range and accuracy of the lidars in the Senior Design lab. After a few tests, we noticed noise in the lidar data and discovered the lidars were pointing into the grass due to the rough terrain of the field. I helped calibrate the lidars after the results of the field tests. I also tested the servo driver and made sure that the Raspberry PI 3B had the ability to communicate with the servo via the Adafruit servo driver through I2C communication.

Rain Gopeng

During the initial planning stages of our rover, both the hardware and software groups split up to distribute tasks. In this stage, I decided to take on the bruteforce method of autonomous navigation for my role. Then, as we progressed, I also helped with other various tasks as required which included sonar and lidar integration, simulation tweaks, small hardware tasks, and testing.

Before I could even begin to work on the bruteforce code, the entire software team, aside from Ryan, had to learn about the environment we would be working in, the Robot Operating System, or ROS. Research on this subject took a couple of weeks, and I was able to gather a basic understanding of how it worked as well as get Ubuntu and ROS installed on my laptop.

After my research and initial testing of ROS, I cloned our github repository, and began coding for the bruteforce program. The first step, was to get familiar with Gazebo, and the topics I would be listening and manipulating. After I learned how the simulation behaved with the sensor data, and published commands, I began with my first challenge of how to get the rover to a goal, using our current information.

I solved this problem, by taking the coordinates of the destination (0,0), and the current coordinates of the rover (X,Y) and applying an inverse tangent function to these two. This gives me an angle to use for comparison, but one notable problem, was that the ROS coordinate system was not the same as a traditional coordinate system of north being positive Y and east being positive X. Instead, north in ROS is a positive X, and east is a negative Y. This is important, because to get an angle, you have to use a reference line, and I had to manipulate the function to take this into account. Lastly, I used four If statements to determine how the robot should turn so it does not do a full circle to face it's destination.

After I solved this problem, I was finally able to begin trying to dodge boxes in the simulation. The program first checks the front lidar, and if it detects a box, the rover turns to avoid the box. When the side lidar sees the box next, it begins to gradually turn back, and finally, once the side lidar loses track of the box, the rover is past the box and once again heads to the goal. Most of my time, was spent tweaking certain values to make sure the rover did not hit the box and dodged obstacles efficiently.

During my time working on this program, I also helped with sonar integration and testing. After Madeleine got the code to work, I was responsible for integrating it into ROS so our programs could read it's values, and test to make sure it worked. Afterwards, we ended up scrapping sonar and going with lidar instead.

At this point, I worked with the lidar code to make sure we got values through our new sensors. I also had to change our Gazebo simulation code to reflect this, and change certain features so that it reflected our change in real life. Then, I also worked on lidar integration within ROS, and tested to make sure it worked with our USB hub and USB to serial adapters.

In terms of hardware, I helped with some basic wire management and crimping, and was also there to help Ryan with field testing setup and breakdown.

Ryan Morris

For the senior design project I served as software lead. Initially I spent time on architecture design. I did a lot of research on how we could make our vehicle fully autonomous. Once I had a design I liked, I pitched it to the team. After discussing the components of the software stack with the whole team we decided on which parts we should all tackle. I was in charge of building the simulation, dynamic path planning, and vehicle tracking. In addition to this, I was what I like to call the "gluer." I would glue the nodes together that everyone was working and get it working into a coherent system.

The simulation was designed using gazebo. Gazebo is a fully featured physics based robots simulation program that allows building the robot models in a language called xarco. I was able to find a similar design to our rover online in xarco. I used that as a base then added all of our sensors. This allowed us to have a simulated rover that was really similar to our real rover. The simulation was crucial for making sure code was working right. However, the downside was it gave us false hope in how the rover would act in the real world.

My biggest mistake this semester was trying to use visual odometry for the vehicle tracking. Because the camera would bounce very rapidly in the grass visual odometry did not work well. Once figuring this out we had to pivot to using encoders which thanks to our electrical team happened very quickly. I added the encoder code to track the vehicle based on how many revolutions the encoder makes. I then use a kalman filter to try to reduce the encoder noise.

I also worked on the dynamic path planning. This is described in detail above under the dynamic path planning and lidar mapping sections. This was by far the most difficult part of the code I dealt with. I had to learn path planning over the course of a month, implement it in efficient C++ code, and thoroughly test it. I initially implemented the algorithms in python but python was too inefficient. The path planner has two O(n^2) components so the openmp library was used to parallelize the for loops.

Path tracking was another component that I did. Initially I used the pure pursuit controller for path tracking because it is the most simple. I had to moved to the Stanley Method because this method has quicker steering response to aggressive trajectories which is very important for dodging obstacles. This code was implemented in python since the control law is just O(1) which wouldn't justify taking the time to port to C++.

Adam Olivera

I was assigned to the hardware team and my specific role was to design the PCB. Initially, I was also tasked with researching H-bridges and to implement an H-bridge design. I researched H-bridge fundamentals and theory of operation. Additionally, I researched D-class power amplifiers since switching is crucial to the operation of an H-bridge. While initially I had called for a mechanical relay, the team decided that this would cause too much electrical and mechanical noise, and so a transistor bridge was to be designed. I chose high power MOSFETS for my design and utilized a p-type and n-type cascaded on top of one another as the side of the H, with bias voltages supplied by the motor battery and resistive dividers to split gate voltages. Ultimately, this subproject was cancelled in favor of a commercial H-bridge since we were short on time, and commercial products already implemented manufacturing processes (including high power efficiency ICs) beyond our available resources.

Additionally, I looked into different battery chemistries as we were deciding on a new battery for the logic circuit, although Brian ended up taking the lead on this task. Instead my main focus was then shifted toward the PCB.

The PCB layout was drafted in KiCad, although the schematic remained on paper for some time as components were regularly switched or moved in the circuit. Once the design was finalized, I performed the task of collecting data as to what specific pins on any given device were connected to, and at what voltage and ampacity. Once the circuit was finalized and verified, the schematic was rebuilt in KiCad. Trace spacings were taken into account as per the guidelines provided on blackboard, and each trace was given enough width to allow for 200 mA at the minimum. Pad mount CAD files for most of the components which required placement on the board were available for download on the retailers' websites, and these were imported into the KiCad file to provide dimensions for the drill. Other than specific traces and pad mounts, the rest of the board was filled with ground and power planes to provide optimal current distribution and to minimize manufacturing costs. The overall PCB accounted for a 40 pin ribbon mount which connected to the Raspberry Pi, a 10 pin connector for the ADC, voltage inputs for the buck converter 1 and battery 2, a voltage output pin for connection to an external fan, and mounts for the I2C/PWM converter module, the wheel encoder, and buck converter 2. The buck converter and I2C/PWM board dimensions were not available online, and Nico performed the task of modeling these components in KiCad, as well as the silk screen and other non-copper layers.

Pertaining to report contribution, I wrote the sections on the motor, servo, and H-bridge since I became familiar with and explained these components during the presentation.

Buse Ozsuca

For our senior design project, I was a part of the software team. Initially, I have learnt how to use Robot Operating System (ROS). Once I gained the knowledge that I need, then I have contributed the software side in two main sections.

First of all, I was assigned to implement an XBox controller for a friendlier user interference. In order to achieve this goal, I have studied about Robot Operating System (ROS) joy package. I have learned how to use joy_node. Then, I proceeded with configuring the joystick. By generating a teleoperation node in C, I have managed to create an interface between XBox controller and Robot Operating System (ROS) [6].

Secondly, I was assigned to work on opencv color thresholding. This task was given to me because we were worried about our rover wandering outside of the competition field. As a backup plan, we wanted our rover to recognize certain colors through image thresholding. Once camera recognized these predetermined colors, the rover would turn around and change its direction. Thus, I have researched about opencv color thresholding. I used raspberry pi's editor to implement my code. Currently, the mentioned code only detects the following colors: green, red and blue. Unfortunately, later in the project, this backup code was no longer needed. As a result, it was not implemented in the final version.

In addition to what listed above, I had two other main responsibilities. I was the Graphic Designer and the Poster Editor for our project.

As Graphic Designer, I was in charge of creating our visual aid. I was tasked to create appealing images and layouts. I have designed plenty of graphics, edits and digital picture illustrations for our reports and presentation slides. I have created our "Rogue Two" brand logo. In addition to that, I have also worked on coloring the car exterior for our rover. Unfortunately, due to a final decision, we have decided to abandon it.

As Poster Editor, I was in change of developing the overall layout of the posters. To create our eye-catching promotional flyers, I have used advanced illustrations programs such as Paint Tool SAI and Adobe Photoshop Elements 9. By using these programs, I have created two different posters designs for the design day.

I have also briefly researched about VNC on the Odroid, multiple machines for ROS and PWM on Raspberry Pi. Since we have decided not to use Odroid, the research wasn't used or applied in the final product.

In terms of report contribution, I have explained the software control and position tracking sections. I have also expanded the additional items section.

Nicholas Payne

My specific contribution was designing the PCB for the rover project. This PCB did not contain many components but rather played a bigger role in connecting the necessary hardware together. Without the PCB on the rover, the mess of wires that connected each and every module was overwhelming. Creating the PCB was a necessity in making the rover look more presentable and properly engineered.

The PCB was developed using KiCAD open source EDA software. The design includes a 40 pin connecting header for the Raspberry Pi as well as a 10 pin connector for the analog to digital converter (ADC). The ADC is used to measure voltage across both batteries and sending back I2C to our microcontroller through Raspberry Pi. This functions a basic battery fuel gauge. The ADC is connected to 3.3V pin 1 of the Raspberry Pi as well as pins 3 and 5 for proper I2C communication. Additionally there is a three pin connector for the wheel encoder which receives power from Raspberry Pi pin 2 and is also connected to GPIO16 and GND pin 6.

This kludge board is also meant to clear up some space on the top side of our rover by mounting the I2C to PWM board and buck converter two onto the board itself using through hole connectors. In order to do this, custom footprints were made resembling the two boards and placed directly on top of the kludge board. Creating custom footprints in KiCAD was challenging because any error in measurement means that the buck converter and PWM board would not fit properly onto the PCB.

Alos connected to the PCB are terminals for both the batteries and as well as a terminal for the fan. The fan was necessary in keeping the servo motor from overheating. Both of the battery terminals were routed onto the board for easy installing in order to create power rails that can be easily accessed by other components of our rover. Making sure that these power rails could handle the necessary current load invloed doing research into professional PCB design. I eventually deemed that by creating a large enough power plain on the top side of the board was thick enough to handle the rated current draws of out motor, microcontroller, and all other components.

The PCB was routed in such a way that our main power connectors, battery two and buck one output, have enough trace width to ensure sure enough current can safely flow through the board. Utilizing ground and power planes made this process a lot easier. Most of the kludge board is made up of ground and power planes to save space and to cut down on the amount of routing the LPKF machine needed to do.

Madeleine Rasche

For the project I was assigned to be report editor and website designer, and I was also the primary creator and editor of our presentation slides. I was assigned to the software part of our team, and was tasked with the hardware communication aspects of the project, specifically the PWM and the sensors. During the initial weeks of the project I spent a lot of time researching the basics of PWM, the ODORID, and ROS, watching tutorials, and downloading the needed tools. I also looked into an easy to use software to design the website and settled on Mobirise4, as it allowed me to have the ease of a plug and play layout, with the quick upload feature allowing it to be uploaded easily to the website. I also took all of the photos shown on the website. During Spring Break, I was in the lab for the majority of the week working on the PWM on the ODROID, a challenge to say the least, as the ODROID XU4 is not really designed to produces PWM waves. I found and modified the coded need to test the system, but problems began to occur [8]. Even with a shifter board attached, and an I2C to PWM board in place, I2C being something that the ODROID in theory should have been able to produce, the system refused to produce the correct output. In order to output the signal you needed to identify and use a specific address by calling the ODROID and asking for I2C pins. Though valid addresses appeared, all were tested and none worked. The code used for testing was run on a Raspberry Pi to see if it was a problem with the code, and it worked as expected, thus showing it was more of a problem with the ODROID than with what we were trying to do. After this bust with the ODORID one of the first solutions before the switch to Raspberry Pi entirely was use an Arduino for the PWM [9]. I found and modified the code to fit the needs of the project, and with Ryan's help we got the driving with this solution. Problems continued to arise though, as we had a lot of sensors to deal with, and the ODROID was not being user friendly with those. Thus, I began testing the sensors using the Arduino with the ultrasonic sensors [10], writing and modifying the needed code for all of them to run in a loop. Ultimately the Arduino was abandoned when we switched to the Raspberry Pi, and thus a new batch of code had to be developed for the Ultrasonics [11] and then the Lidar [12]. The ultrasonic code, was tested one sensors at a time, and then all five (the number we had at the time) were run in a loop. This code was eventually abandoned as well in favor of Lidar code, and this was tested and modified. One of the issues with the Lidar setup was there were some initial wrong wirings, so the code was fine, but the wrong pins were attached. There were also some issues with the differing libraries needed for the Lidar versus the ultrasonic.Some additional miscellaneous things I did were complete the write up for the Design Day pamphlet, proofread the poster before printing, and miscellaneous research for various bugs, including the Lidar sensing too much data. I attended every meeting for this project, and have written a large part of this report, and proofread and compiled the additional things needed to complete the report. I specifically wrote the intro, abstract, hardware communication, general software description, the Lidar versus Ultrasonic section, the Raspberry Pi versus ODROID section, the competition, planning, additional items, conclusion and recommendations, and part of software control.

Jose Tomimatzu

Working on this project has given me the opportunity to demonstrate the knowledge and critical thinking skills I have acquired during my stay here at San Diego State. During this project, I performed the role of project manager in addition to working closely with the hardware team. As a project manager, I maintained an overview of the project progress and made the crucial decisions for the team. One of the most important responsibilities I had as the project manager was to make sure the software and hardware team were able to run a task in parallel with one another. By maintaining parallel task, we were able to improve productivity and meet our milestones.

Working alongside the hardware team allowed me to work on the all the vital components of our project. One of the essential systems was the drive motor. Alvaro and I analyzed the current requirements of our 540 size 20 turn motor that came with the original RC car. We found that the motor required a peak starting current of about 16 amps. After researching multiple H-bridge modules, I choose one that was capable of delivering 15 amps continuous and 30 amp peaks. From our initial team meeting, we decide to utilize the Odroid XU4 board. After researching the datasheet, we determined that the current requirements of the Odroid XU4 were 6 amps under full load. By knowing the 6 amp requirement along with the distance sensors current needs, I was able to research and select an appropriate buck converter. I choose a buck converter that was capable of outputting 12 amps and has short-circuit, over temperature protection.

After purchasing the appropriate components, Alvaro, Dan and I began the fabrication process. We started by measuring the components and designed a support structure that could accommodate all the hardware. I decided to fabricate the rover body from aluminum because it is lightweight, affordable, and easy to work with. The entire rover frame was handmade. The fabrication process was a semester-long ordeal because of the multiple pivots in our design. Once the initial vehicle was constructed, we were able to test the hardware in the field. The testing made us realize that bought the motor and steering servo motor were underpowered and that tracking distance was going to be a problem. To solve the drive motor problem I decided to swap the 20 turn motor with an 80 turn motor. The high turn motor was ideal because it is capable of delivering higher torque. The steering servo motor was replaced with a high torque, metal geared motor, 25 spline servo motor. This high torque motor required a separate buck converter. Ryan and I solved the vehicle distance tracking problem by using a slot sensor to count the revolutions of the DC motor and converting that to a distance traveled.

Some of the non-engineering tasks I performed for this project were purchasing and embroidering the team shirts, along with organizing, buying and setting up the design day display both. I also assisted in almost all the field and lab testing.

Alvaro Valera-Rivera

My role in this project was Hardware Lead. During the initial planning of this project, we separated into two general aspects of the project: Hardware and Software. I was in charge of overseeing that the physical aspect of the rover operated correctly and was completed in a timely manner. I prioritized and assigned tasks, and created contingency plans in order to ensure that they would be completed. I was also involved, in the fabrication of the frame, mounting, and selection of the components, software research, as well as testing. After we obtained the RC Car, it was necessary to test the components that came with the car in order to determine what components could be used. After testing I decided that the best option would be to only use the motor, servo, and battery. We determined that under no load conditions the motor could spike up to 10 A and then operated under ~3 A. Initially the plan was to implement our project utilizing the ODroid XU4, after researching the specs we found that it required 5V and a max of 6A. I assigned Adam to find an H-Bridge, Brian to find a battery, Nico to find a buck converter, and Dan to write a simple test code to test the motor and servo. Jose and I continued testing and found an H-bridge that was able to handle 30 A spikes and 15 A continuous current, we also found a buck converter that was able to handle 12 A continuous. While we waited for parts to arrive, we started to create a base so we could start mounting components. The frame and mounting were custom made out of aluminum by me, Jose, and Dan. Using an Arduino I ran test code to ensure that everything was working correctly and there was no smoke coming from the rover. After testing we found out that the motor and servo needed more power so Jose, and I decided to replace the original motor and servo for an 80-turn DC brushless motor and a 20 kg/cm servo which solved our steering issues. Hardware was able to meet its deadline of having a working prototype by spring break. While Dan was doing research for the code, he found out that the Odroid XU4 had no hardware PWM. The solution brought up by dan was to utilize the built-in I2C of the ODroid and use an I2C to PWM module. Me, Jose, and Ryan decided that this was the best solution to the problem and began to implement it. The next challenge we encountered was getting the Odroid XU4 to communicate with the other modules. Because hardware team was done with the essential part, Jose, dan, and I volunteered to help the software team to search for solutions. Because the Odroid XU4 has little community support and documentation I started searching for alternative solutions. During the research, I found a version of raspberry pi that included the ROS software that we planned on using. Because of this image, I brought up that we should switch to the Raspberry PI to Jose, and Ryan. We all agreed that was the best approach due to time constraint. Since we had originally planned to use a board that required 6 A to operate switching to the Raspberry Pi which only required 2.5 A was easy. We decided to keep the I2C to PWM in order to save some of the CPU from the Raspberry Pi. Me, Dan, Jose, Ryan, and Rain spend the rest of the semester doing field testing, integration, and solving problems as they came. The biggest surprise we had from the hardware perspective was that we lost 2 servos, and our buck achieved thermal shutdown. I and Jose solved this issue by replacing the buck powering the servo with one able to handle larger currents.

REFERENCE

[1] “ROS,” <http://www.ros.org/>
[2] “Driving servo motor with PWM signal,” <https://electronics.stackexchange.com/questions/346603/driving-servo-motor-with-pwm-signal?rq=1>
[3] “What is an Ultrasonic Sensor?,” <http://education.rec.ri.cmu.edu/content/electronics/boe/ultrasonic_sensor/1.html>
[4] Fox, D., “The Dynamic Window Approach to Collision Avoidance,” <https://pdfs.semanticscholar.org/cd9e/0901a07262db18fb249efb82094e7c266527.pdf>
[5] “Writing a Teleoperation Node for a Linux-Supported Joystick,” <http://wiki.ros.org/joy/Tutorials/WritingTeleopNode>
[6] “joy,” <http://wiki.ros.org/joy>
[7] “Attitude and heading reference system,” <https://en.wikipedia.org/wiki/Attitude_and_heading_reference_system>
[8] “Using the Python Library,” <https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/using-the-python-library>
[9] “Read,” <https://www.arduino.cc/en/Serial/Read>
[10] “Multiple Ultrasonic Sensor With Arduino,” <https://www.theengineeringprojects.com/2015/02/interfacing-multiple-ultrasonic-sensor-arduino.html>
[11] “Using a Raspberry Pi distance sensor (ultrasonic sensor HC-SR04),” <https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/>
[12] “TFMini - Micro LiDAR Module Hookup Guide,” <https://learn.sparkfun.com/tutorials/tfmini---micro-lidar-module-hookup-guide>
[13] “TF Mini LiDAR,” <https://www.seeedstudio.com/Seeedstudio-Grove-TF-Mini-LiDAR-p-2996.html>
[14] "Automous Platentary Mobility,” <https://mars.nasa.gov/mer/technology/is_autonomous_mobility.html>

APPENDICES

Planning and Scheduling

During the initial weeks, a basic design and plan was laid out for the project. Assignments were also given to each team member. Purchasing then began, and we started testing and researching the parts purchased, specifically the RC Car. Software development then began in parallel along with hardware fabrication, assembly, and testing. The two portions were finally integrated and the final testing closed out the project. Below (Figure 1) the Gantt chart for the project is shown, showing the more specific parts of the project broken down.

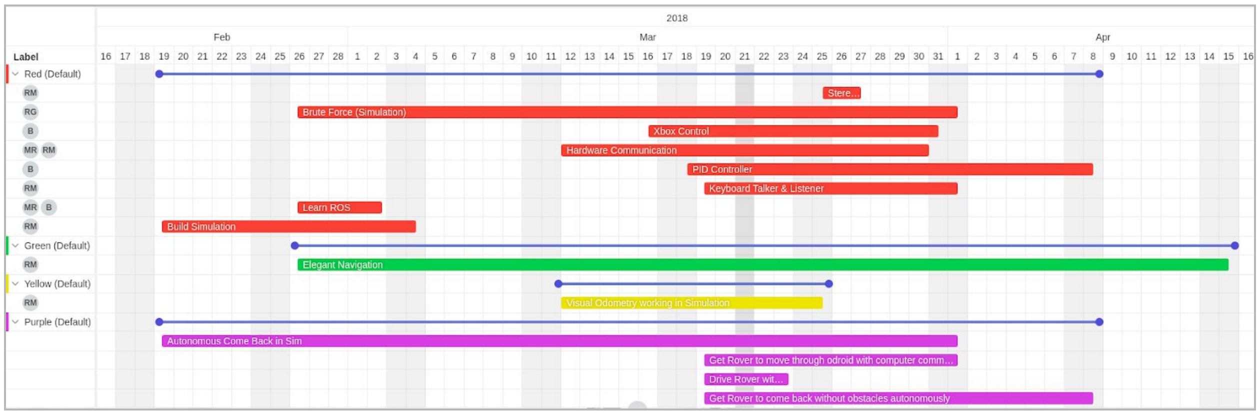


Figure 14: Gantt Chart

Budget

Total Limit	$1000
Amount Spent (For Rover)	$790.44
Total Amount Spent (Including Unused Items)	$1,178.94

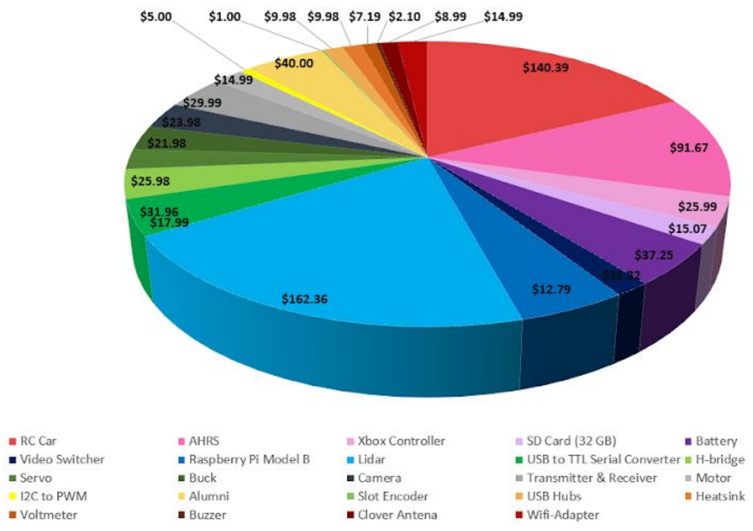
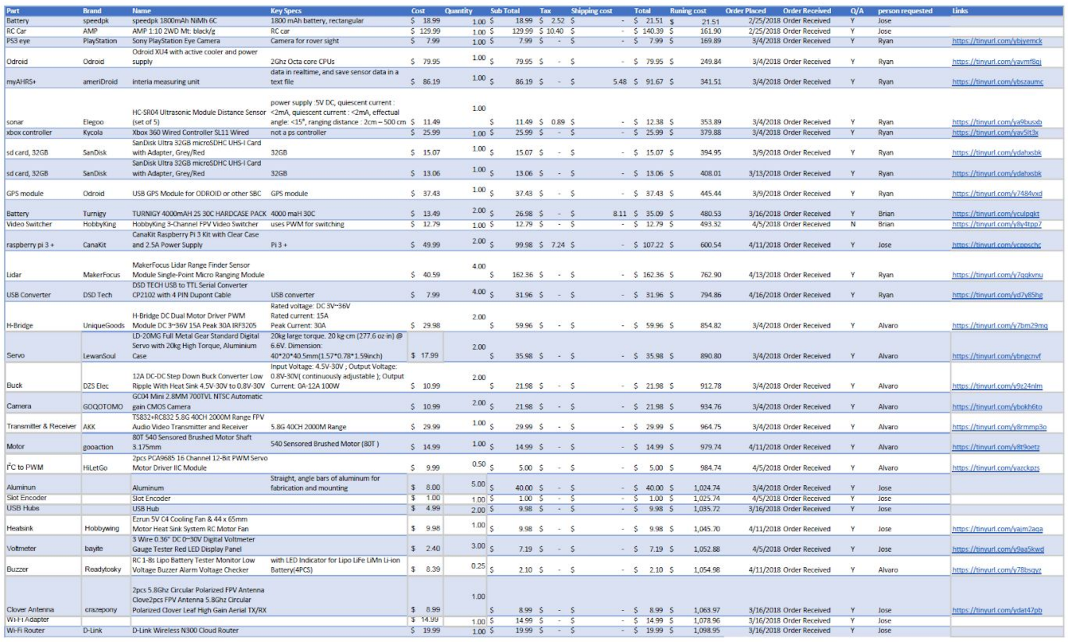


Figure 15: Budget breakdown

Complete Bill of Materials



Part	Brand	Name	Key Specs	Cost	Quantity	Sub Total	Tax	Shipping cost	Total	Runing cost	Order Placed	Order Received	Q/A	person requested	Links
Battery	speedpk	speedpk 1800mAh NiMh 6C	1800 mAh battery, rectangular	$ 18.99	1.00	$ 18.99	$ 2.52	$ -	$ 21.51	$ 21.51	2/25/2018	Order Received	Y	Jose	
RC Car	AMP	AMP 1:10 2WD Mt: black/g	RC car	$ 129.99	1.00	$ 129.99	$ 10.40	$ -	$ 140.39	$ 161.90	2/25/2018	Order Received	Y	Jose	
PS3 eye	PlayStation	Sony PlayStation Eye Camera	Camera for rover sight	$ 7.99	1.00	$ 7.99	$ -	$ -	$ 7.99	$ 169.89	3/4/2018	Order Received	Y	Ryan	https://tinyurl.com/ybjyemck
Odroid	Odroid	Odroid XU4 with active cooler and power supply	2Ghz Octa core CPUs	$ 79.95	1.00	$ 79.95	$ -	$ -	$ 79.95	$ 249.84	3/4/2018	Order Received	Y	Ryan	https://tinyurl.com/yavmf8qj
myAHRS+	ameriDroid	interia measuring unit	data in realtime, and save sensor data in a text file	$ 86.19	1.00	$ 86.19	$ -	$ 5.48	$ 91.67	$ 341.51	3/4/2018	Order Received	Y	Ryan	https://tinyurl.com/ybszaumc
sonar	Elegoo	HC-SR04 Ultrasonic Module Distance Sensor (set of 5)	power supply :5V DC, quiescent current : <2mA, quiescent current : <2mA, effectual angle: <15°, ranging distance : 2cm – 500 cm	$ 11.49	1.00	$ 11.49	$ 0.89	$ -	$ 12.38	$ 353.89	3/4/2018	Order Received	Y	Ryan	https://tinyurl.com/ya9busxb
xbox controller	Kycola	Xbox 360 Wired Controller SL11 Wired	not a ps controller	$ 25.99	1.00	$ 25.99	$ -	$ -	$ 25.99	$ 379.88	3/4/2018	Order Received	Y	Ryan	https://tinyurl.com/yav5lt3x
sd card, 32GB	SanDisk	SanDisk Ultra 32GB microSDHC UHS-I Card with Adapter, Grey/Red	32GB	$ 15.07	1.00	$ 15.07	$ -	$ -	$ 15.07	$ 394.95	3/9/2018	Order Received	Y	Ryan	https://tinyurl.com/ydahxsbk
sd card, 32GB	SanDisk	SanDisk Ultra 32GB microSDHC UHS-I Card with Adapter, Grey/Red	32GB	$ 13.06	1.00	$ 13.06	$ -	$ -	$ 13.06	$ 408.01	3/13/2018	Order Received	Y	Ryan	https://tinyurl.com/ydahxsbk
GPS module	Odroid	USB GPS Module for ODROID or other SBC	GPS module	$ 37.43	1.00	$ 37.43	$ -	$ -	$ 37.43	$ 445.44	3/9/2018	Order Received	Y	Ryan	https://tinyurl.com/y7484vxd
Battery	Turnigy	TURNIGY 4000mAH 2S 30C HARDCASE PACK	4000 maH 30C	$ 13.49	2.00	$ 26.98	$ -	$ 8.11	$ 35.09	$ 480.53	3/16/2018	Order Received	Y	Brian	https://tinyurl.com/yculpqkt
Video Switcher	HobbyKing	HobbyKing 3-Channel FPV Video Switcher	uses PWM for switching	$ 12.79	1.00	$ 12.79	$ -	$ -	$ 12.79	$ 493.32	4/5/2018	Order Received	N	Brian	https://tinyurl.com/y8y4tpp7
raspberry pi 3 +	CanaKit	CanaKit Raspberry Pi 3 Kit with Clear Case and 2.5A Power Supply	Pi 3 +	$ 49.99	2.00	$ 99.98	$ 7.24	$ -	$ 107.22	$ 600.54	4/11/2018	Order Received	Y	Jose	https://tinyurl.com/ycppschc
Lidar	MakerFocus	MakerFocus Lidar Range Finder Sensor Module Single-Point Micro Ranging Module		$ 40.59	4.00	$ 162.36	$ -	$ -	$ 162.36	$ 762.90	4/13/2018	Order Received	Y	Ryan	https://tinyurl.com/y7qqkvnu
USB Converter	DSD Tech	DSD TECH USB to TTL Serial Converter CP2102 with 4 PIN Dupont Cable	USB converter	$ 7.99	4.00	$ 31.96	$ -	$ -	$ 31.96	$ 794.86	4/16/2018	Order Received	Y	Ryan	https://tinyurl.com/yd7y85hg
H-Bridge	UniqueGoods	H-Bridge DC Dual Motor Driver PWM Module DC 3~36V 15A Peak 30A IRF3205	Rated voltage: DC 3V~36V Rated current: 15A Peak Current: 30A	$ 29.98	2.00	$ 59.96	$ -	$ -	$ 59.96	$ 854.82	3/4/2018	Order Received	Y	Alvaro	https://tinyurl.com/y7bm29mq
Servo	LewanSoul	LD-20MG Full Metal Gear Standard Digital Servo with 20kg High Torque, Aluminium Case	20kg large torque. 20 kg·cm (277.6 oz·in) @ 6.6V. Dimension: 40*20*40.5mm(1.57*0.78*1.59inch)	$ 17.99	2.00	$ 35.98	$ -	$ -	$ 35.98	$ 890.80	3/4/2018	Order Received	Y	Alvaro	https://tinyurl.com/ybngcnvf
Buck	DZS Elec	12A DC-DC Step Down Buck Converter Low Ripple With Heat Sink 4.5V-30V to 0.8V-30V	Input Voltage: 4.5V-30V ; Output Voltage: 0.8V-30V(continuously adjustable); Output Current: 0A-12A 100W	$ 10.99	2.00	$ 21.98	$ -	$ -	$ 21.98	$ 912.78	3/4/2018	Order Received	Y	Alvaro	https://tinyurl.com/y9z24nlm
Camera	GOQOTOMO	GC04 Mini 2.8MM 700TVL NTSC Automatic gain CMOS Camera		$ 10.99	2.00	$ 21.98	$ -	$ -	$ 21.98	$ 934.76	3/4/2018	Order Received	Y	Alvaro	https://tinyurl.com/ybokh6to
Transmitter & Receiver	AKK	TS832+RC832 5.8G 40CH 2000M Range FPV Audio Video Transmitter and Receiver	5.8G 40CH 2000M Range	$ 29.99	1.00	$ 29.99	$ -	$ -	$ 29.99	$ 964.75	3/4/2018	Order Received	Y	Alvaro	https://tinyurl.com/y8rmmp3o
Motor	gooaction	80T 540 Sensored Brushed Motor Shaft 3.175mm	540 Sensored Brushed Motor (80T)	$ 14.99	1.00	$ 14.99	$ -	$ -	$ 14.99	$ 979.74	4/11/2018	Order Received	Y	Alvaro	https://tinyurl.com/y8t9oetz
I^{2}C to PWM	HiLetGo	2pcs PCA9685 16 Channel 12-Bit PWM Servo Motor Driver IIC Module		$ 9.99	0.50	$ 5.00	$ -	$ -	$ 5.00	$ 984.74	4/5/2018	Order Received	Y	Alvaro	https://tinyurl.com/yazckpzs
Aluminun		Aluminum	Straight, angle bars of aluminum for fabrication and mounting	$ 8.00	5.00	$ 40.00	$ -	$ -	$ 40.00	$ 1,024.74	3/4/2018	Order Received	Y	Jose	
Slot Encoder		Slot Encoder		$ 1.00	1.00	$ 1.00	$ -	$ -	$ 1.00	$ 1,025.74	4/5/2018	Order Received	Y	Jose	
USB Hubs		USB Hub		$ 4.99	2.00	$ 9.98	$ -	$ -	$ 9.98	$ 1,035.72	3/16/2018	Order Received	Y	Jose	
Heatsink	Hobbywing	Ezrun 5V C4 Cooling Fan & 44 x 65mm Motor Heat Sink System RC Motor Fan		$ 9.98	1.00	$ 9.98	$ -	$ -	$ 9.98	$ 1,045.70	4/11/2018	Order Received	Y	Jose	https://tinyurl.com/yajm2aqa
Voltmeter	bayite	3 Wire 0.36" DC 0~30V Digital Voltmeter Gauge Tester Red LED Display Panel		$ 2.40	3.00	$ 7.19	$ -	$ -	$ 7.19	$ 1,052.88	4/5/2018	Order Received	Y	Jose	https://tinyurl.com/y9aa5kwd
Buzzer	Readytosky	RC 1-8s Lipo Battery Tester Monitor Low Voltage Buzzer Alarm Voltage Checker	with LED Indicator for Lipo LiFe LiMn Li-ion Battery(4PCS)	$ 8.39	0.25	$ 2.10	$ -	$ -	$ 2.10	$ 1,054.98	4/11/2018	Order Received	Y	Alvaro	https://tinyurl.com/y78bsqyz
Clover Antenna	crazepony	2pcs 5.8Ghz Circular Polarized FPV Antenna Clove2pcs FPV Antenna 5.8Ghz Circular Polarized Clover Leaf High Gain Aerial TX/RX		$ 8.99	1.00	$ 8.99	$ -	$ -	$ 8.99	$ 1,063.97	3/16/2018	Order Received	Y	Jose	https://tinyurl.com/ydat47pb
Wi-Fi Adapter				$ 14.99	1.00	$ 14.99	$ -	$ -	$ 14.99	$ 1,078.96	3/16/2018	Order Received	Y	Jose	
Wi-Fi Router	D-Link	D-Link Wireless N300 Cloud Router		$ 19.99	1.00	$ 19.99	$ -	$ -	$ 19.99	$ 1,098.95	3/16/2018	Order Received	Y	Jose	

Figure 16: BOM

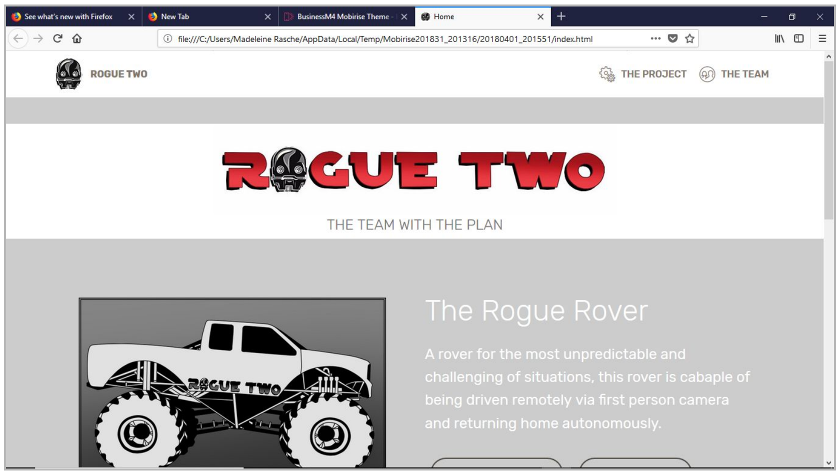
Complete BOM with Links: https://tinyurl.com/ydhupog6

Additional Items

We have used Trello to organize create our Gantt Chart. We have used Slack to communicate with each other. We have shared our files through Google Drive. In addition, we have used professional drawing programs such as Adobe Photoshop Elements 9 and Paint Tool SAI to create our visual illustrations. Furthermore, we have used Github to share our code files.

The website was designed using Mobirise4 software and contains the general concepts of the project. Additionally the website contains a photographic timeline of the progress of the project. The software used to create this website allows for quick and easy design, as well as fast updates.

Website URL: https://www.roguetwo.sdsu.edu.


See what's new with Firefox
New Tab
BusinessM4 Mobirise Theme
Home
file:///C:/Users/Madeleine Rasche/AppData/Local/Temp/Mobirise201831_201316/20180401_201551/index.html
ROGUE TWO
THE PROJECT
THE TEAM
ROGUE TWO

THE TEAM WITH THE PLAN

The Rogue Rover

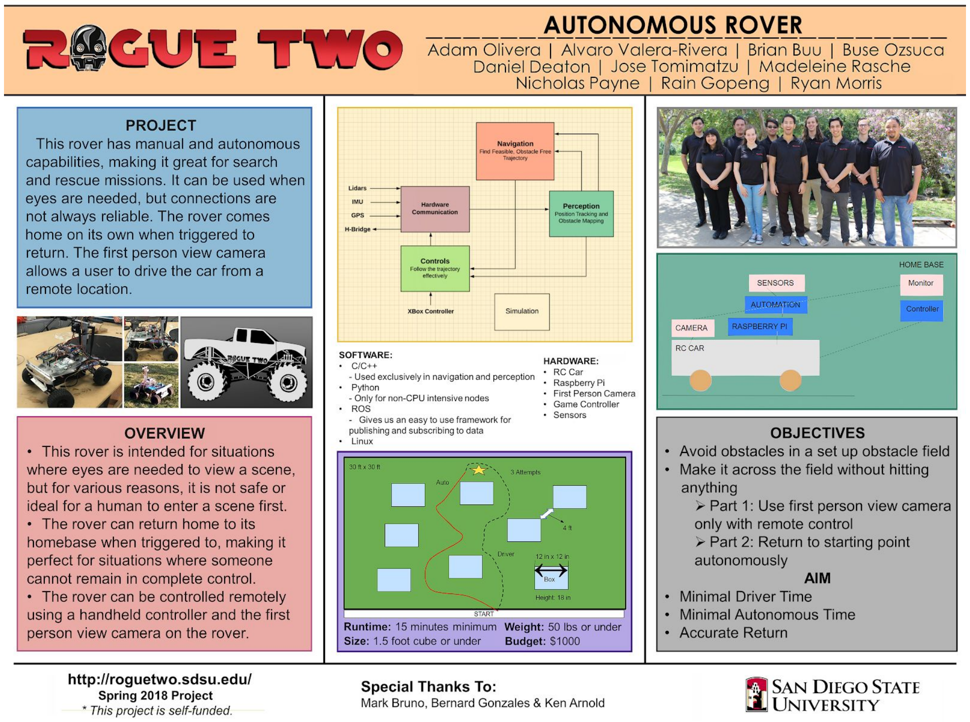
A rover for the most unpredictable and challenging of situations, this rover is cabaple of being driven remotely via first person camera and returning home autonomously.

Figure 17: Website

The presentation was completed and presented on May 1, 2018 and can be found at the link below. It goes into a general overview of what is discussed in depth within this paper.

Presentation: https://goo.gl/7enMqe

Poster was completed and submitted on April 23, 2018. The poster images can be found down below.



AUTONOMOUS ROVER

Adam Olivera | Alvaro Valera-Rivera | Brian Buu | Buse Ozsuca
Daniel Deaton | Jose Tomimatzu | Madeleine Rasche
Nicholas Payne | Rain Gopeng | Ryan Morris

PROJECT

This rover has manual and autonomous capabilities, making it great for search and rescue missions. It can be used when eyes are needed, but connections are not always reliable. The rover comes home on its own when triggered to return. The first person view camera allows a user to drive the car from a remote location.

OVERVIEW

- This rover is intended for situations where eyes are needed to view a scene, but for various reasons, it is not safe or ideal for a human to enter a scene first.
- The rover can return home to its homebase when triggered to, making it perfect for situations where someone cannot remain in complete control.
- The rover can be controlled remotely using a handheld controller and the first person view camera on the rover.

Navigation
Find Feasible, Obstacle Free Trajectory
Lidars
IMU
GPS
H-Bridge
Hardware Communication
Perception
Position Tracking and Obstacle Mapping
Controls
Follow the trajectory effectively
XBox Controller
Simulation

SOFTWARE:
- C/C++
  - Used exclusively in navigation and perception
- Python
  - Only for non-CPU intensive nodes
- ROS
  - Gives us an easy to use framework for publishing and subscribing to data
- Linux

HARDWARE:
- RC Car
- Raspberry Pi
- First Person Camera
- Game Controller
- Sensors

Runtime: 15 minutes minimum **Weight:** 50 lbs or under
Size: 1.5 foot cube or under **Budget:** $1000

OBJECTIVES
- Avoid obstacles in a set up obstacle field
- Make it across the field without hitting anything
  - ➢ Part 1: Use first person view camera only with remote control
  - ➢ Part 2: Return to starting point autonomously

AIM
- Minimal Driver Time
- Minimal Autonomous Time
- Accurate Return

http://roguetwo.sdsu.edu/
Spring 2018 Project
** This project is self-funded.*

Special Thanks To:
Mark Bruno, Bernard Gonzales & Ken Arnold

San Diego State University

Figure 18: Poster 1

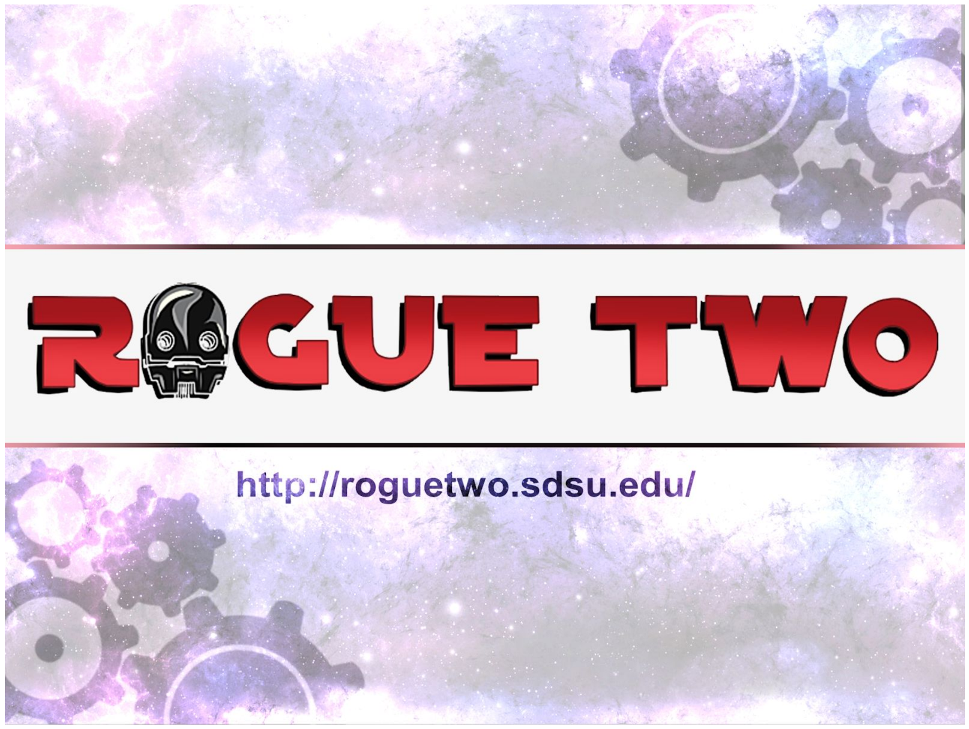


Figure 19: Poster 2

PROJECT ARCHIVE

Contents of Project Archive

1. Report PDF
2. PCB Files
3. Presentation Slides
4. Complete Code
5. Bill of Materials

Link: https://drive.google.com/open?id=1oytF7EUAHKur8Zpz9nEyIsL8_qzzt7oh