

6-10-2016

## RSL Rover

Patrick Barone  
*Santa Clara University*

Giovanni Briggs  
*Santa Clara University*

Aaron Burns  
*Santa Clara University*

Hesham Naja  
*Santa Clara University*

Zoe Demertzis  
*Santa Clara University*

Follow this and additional works at: [http://scholarcommons.scu.edu/idp\\_senior](http://scholarcommons.scu.edu/idp_senior)

 Part of the [Computer Engineering Commons](#), and the [Mechanical Engineering Commons](#)

---

### Recommended Citation

Barone, Patrick; Briggs, Giovanni; Burns, Aaron; Naja, Hesham; and Demertzis, Zoe, "RSL Rover" (2016). *Interdisciplinary Design Senior Theses*. Paper 24.

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Interdisciplinary Design Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact [rscroggin@scu.edu](mailto:rscroggin@scu.edu).

**SANTA CLARA UNIVERSITY**

Departments of Computer Engineering and Mechanical Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED  
UNDER MY SUPERVISION BY

Patrick Barone, Giovanni Briggs, Aaron Burns,  
Zoe Demertzis, Hesham Naja

ENTITLED

**RSL ROVER**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF

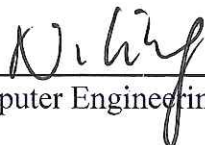
**BACHELOR OF SCIENCE**  
IN  
**COMPUTER SCIENCE AND ENGINEERING**  
**MECHANICAL ENGINEERING**



Thesis Advisor

6/10/16

date



Department Chair (Computer Engineering)

6/9/16

date



Department Chair (Mechanical Engineering)

6/10/2016

date



# RSL ROVER

By

Patrick Barone, Giovanni Briggs, Aaron Burns,  
Zoe Demertzis, Hesham Naja

## **SENIOR DESIGN PROJECT REPORT**

Submitted to  
the Departments of Computer Science and Engineering and Mechanical Engineering  
of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements  
for the degree of  
Bachelor of Science in Computer Science and Engineering and Mechanical Engineering

Santa Clara, California

Spring 2016

# RSL Rover

Patrick Barone, Giovanni Briggs, Aaron Burns,  
Hesham Naja, Zoe Demertzis

Departments of Computer and Mechanical Engineering  
Santa Clara University  
2016

## ABSTRACT

The goal of this project was to design and implement an unmanned vehicle that can assess the air quality and general state of a post-fire environment. To do this, we equipped Santa Clara University's Polaris 6x6 Ranger with appropriate sensors and cameras to determine how safe the environment is for humans to enter. We also used GPS and laser scans to generate a 3D map that operators can use to define certain zones as particularly dangerous. Finally, we incorporated partially-autonomous sensing capabilities that will allow the operator to easily drive the vehicle. The result was a rugged, advanced, and intuitive vehicle that can be used to protect fire responders from any lingering hazards during the investigation of a post-fire environment. This vehicle is accompanied by a powerful operating system and localization techniques that will allow any future research groups to help this vehicle evolve into a fully autonomous system.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Literature Review . . . . .	3
1.3	Vehicle Background . . . . .	6
1.4	Problem Statement . . . . .	7
<b>2</b>	<b>Systems Level Design</b>	<b>8</b>
2.1	Customer Needs . . . . .	8
2.2	Key Requirements . . . . .	9
2.3	System Level Sketch and Use Cases . . . . .	10
2.4	Functional Analysis . . . . .	12
2.5	Benchmarking Results . . . . .	14
2.6	System Level Issues, Trade-off Analysis . . . . .	17
2.6.1	LIDAR Physical Configuration . . . . .	17
2.6.2	Sensor Physical Configuration . . . . .	18
2.7	System Level Architecture . . . . .	19
2.8	Team and Project Management . . . . .	20
<b>3</b>	<b>Subsystem: Environmental Sensing</b>	<b>23</b>
3.1	Air Quality Assessment . . . . .	23
3.1.1	Payload Requirements . . . . .	23
3.1.2	Component Selection . . . . .	24
3.1.3	PCB Design . . . . .	26
3.2	Cameras . . . . .	28
3.3	Sensor and Camera Layout . . . . .	29
<b>4</b>	<b>Subsystem: Sensor Housing</b>	<b>30</b>
4.1	Need for Housing . . . . .	30

4.2	Materials Used . . . . .	30
4.3	Initial Design . . . . .	31
4.4	CFD Analysis and Iterative Work . . . . .	33
4.5	Final Design . . . . .	36
<b>5</b>	<b>Subsystem: Operator Control and User Interface</b>	<b>38</b>
5.1	User-Interface Design . . . . .	38
5.2	RobotWebTools and the ROS Control Center . . . . .	39
5.3	Improving the ROS Control Center . . . . .	41
5.3.1	Rendering the LIDAR Point Cloud . . . . .	43
5.4	Network for Internet Communication . . . . .	44
<b>6</b>	<b>Subsystem: Communications</b>	<b>46</b>
<b>7</b>	<b>Subsystem: Power</b>	<b>48</b>
<b>8</b>	<b>Subsystem: Localization/Mapping</b>	<b>49</b>
8.1	Sensors . . . . .	49
8.2	Coordinate Frames . . . . .	52
8.3	Kalman Filter . . . . .	53
8.4	Hector SLAM . . . . .	55
8.5	3D Visualization . . . . .	56
<b>9</b>	<b>Construction Plan</b>	<b>58</b>
<b>10</b>	<b>System Integration Testing and Results</b>	<b>59</b>
10.1	Range Requirement Testing . . . . .	59
10.2	Latency Requirement Verification . . . . .	60
10.3	GPS Testing . . . . .	61
10.4	Localization and Mapping Testing . . . . .	62
10.5	Environmental Sensor Package Testing . . . . .	63

10.6 Blind-spot Testing . . . . .	65
<b>11 Costing Analysis</b>	<b>68</b>
<b>12 Commercialization Plan</b>	<b>69</b>
12.1 Introduction . . . . .	69
12.2 Goals and Objectives . . . . .	69
12.3 Description . . . . .	70
12.4 Potential Markets . . . . .	71
12.5 Competition . . . . .	72
12.6 Sales and Marketing Strategy . . . . .	75
12.7 Manufacturing Plan . . . . .	76
12.8 Product Cost and Price . . . . .	77
12.9 Service and Warranties . . . . .	79
12.10 Financial Plan and ROI . . . . .	80
<b>13 Engineering Standards and Realistic Constraints</b>	<b>81</b>
13.1 Ethics . . . . .	81
13.2 Health and Safety . . . . .	81
13.3 Manufacturability . . . . .	82
13.4 Environment . . . . .	83
13.5 Society . . . . .	83
<b>14 Summary and Conclusions</b>	<b>85</b>
<b>Appendix A Design Requirement Flowdown</b>	<b>A-1</b>
<b>References</b>	<b>A-1</b>
<b>Appendix B Market Survey</b>	<b>B-1</b>
<b>Appendix C Tradeoff Analysis</b>	<b>C-1</b>



Appendix D Budget	D-1
Appendix E Gantt Chart	E-1
Appendix F Power Budget	F-1
Appendix G Drawings	G-1
Appendix H Code	H-1
H.1 Cameras Launch File . . . . .	H-1
Appendix I Safety Protocol	I-1
Appendix J Conference Slides	J-1

# List of Figures

1.1	Polaris 6x6 Ranger . . . . .	3
1.2	Fire Investigation Flowchart [?] . . . . .	4
2.1	Use Case Scenario Illustration . . . . .	11
2.2	Software Component Block Diagram . . . . .	12
2.3	Mech/Elen Component Block Diagram . . . . .	13
2.4	Argo J5 Mobility Platform . . . . .	14
2.5	Northrop Grumman’s Andros F6 . . . . .	15
2.6	Northrop Grumman’s Remotec Wheelbarrow Mk9 . . . . .	15
2.7	Elimco UAV-E300 . . . . .	16
2.8	Sensefly’s UAV EBee . . . . .	16
2.9	LIDAR Types . . . . .	17
2.10	Sample Sensor Types . . . . .	19
2.11	System Layout Architecture . . . . .	19
3.1	MQ-Series gas sensor . . . . .	24
3.2	Sharp air particulate sensor with air flow . . . . .	25
3.3	Air quality PCB schematic . . . . .	26
3.4	Printed circuit board layout . . . . .	27
3.5	Logitech c615 camera . . . . .	28
3.6	Example of OpenCV People Detection Application . . . . .	29
4.1	For the gas sensors to work, air must flow over the sensor . . . . .	32
4.2	For the air particulate sensor to work, air must flow through the sensor	32
4.3	The solid designed for the initial CFD analysis that shows the initial design . . . . .	33
4.4	CFD streamline analysis for our first design . . . . .	34
4.5	CFD streamline analysis for our second design . . . . .	35
4.6	CFD streamline analysis for our final design . . . . .	36

4.7	The final sensor housing design equipped with our sensors, mounted to the vehicle . . . . .	37
6.1	Communication Subsystem . . . . .	46
8.4	Novatel ProPak-LB GPS System . . . . .	51
8.5	RSL Rover LIDAR Sensors . . . . .	52
10.3	3D visualization of vehicle during mapping activity . . . . .	63
12.1	The RSL Rover monitoring air quality around a controlled burn in California . . . . .	71
12.2	Argo J5 Mobility Platform . . . . .	73
12.3	Northrop Grumman’s Andros F6 . . . . .	73
12.4	Northrop Grumman’s Remotec Wheelbarrow Mk9 . . . . .	74
12.5	Elimco UAV-E300 . . . . .	75
12.6	Sensefly’s UAV EBee . . . . .	75
12.7	Excel table showing Team RSL Rover’s estimated financial plan and return of investment . . . . .	80
A.1	Requirements Flowdown . . . . .	A-1
A.2	Test Plan . . . . .	A-2

# List of Tables

2.1	Key Requirements . . . . .	10
2.2	Current Product Comparison . . . . .	14
4.1	The properties of acrylic made available from the UL company . . . . .	30
10.1	Range requirements and verification results . . . . .	60
10.2	Latency requirements and verification results . . . . .	61
12.1	Current Product Comparison . . . . .	73
12.2	Component cost breakdown . . . . .	78
12.3	Labor breakdown . . . . .	78
12.4	Cost per Vehicle . . . . .	78
B.1	Customer Needs . . . . .	B-1

# 1 Introduction

Robotic systems are used to enhance even the most mundane aspects of daily life. Robots can clean our floors, dispense our soft drinks, and, pretty soon, drive us around town. Powerful and precise, robots are also used to make up for the qualities humans lack. They can do and see what humans cannot, and are resilient in situations where humans physically could not be. As one might imagine, robots are used to perform great feats.

Recently, autonomous systems have been of great interest to robotics experts. The race to fully autonomous driving is at its peak with GM, Mercedes-Benz, Audi, Nissan, BMW, Renault, Tesla and Google all expecting to sell at least partially autonomous vehicles by 2020[?]; however, the applications for autonomous systems are expanding past commercial uses. Researchers are now looking to use unmanned and partially autonomous vehicles, both ground and air, to aid in providing relief services in the event of a natural disaster.[?]

Gathering information is a critical part of providing disaster relief services. This is true both immediately following a natural disaster, and for prolonged periods of time afterwards. Relief service personnel and investigators place themselves in dangerous environments in order to assist those affected by a natural disaster. To avoid placing human lives in unnecessary danger, relief services have begun to rely on unmanned vehicles to gather information.

Drones, in particular, have been utilized to collect this information; however, the environment after a natural disaster is not always conducive to drone activity. Extreme weather, dust, and smoke are all conditions that make it difficult for drones to operate and gather information to aid relief services [?]. In certain situations like wildfires, having drones in the air actually severely impacts the ability of firefighters to combat wildfires. While a drone is in the air, other manned aerial vehicles cannot enter the airspace, preventing them from performing vital operations [?].

Furthermore, after the initial relief service efforts, the area can remain hostile

for indefinite periods of time. Investigators looking to determine the root cause of a disaster – building collapse, wildfire, flood, etc. – have to be aware of:

- building structural integrity
- smoke
- electrical, chemical, or biological hazards
- other health risks [?].

This investigatory period can be as dangerous as the initial relief service operations. This poses the question: how can robots be used to allow post-disaster investigators to safely determine what types of risks lurk within the disaster zone?

## 1.1 Motivation

The motivation behind our project is to provide an unmanned vehicle that can examine an environment after a natural disaster so that humans do not have to place their lives in danger to do so.

- Santa Clara University possesses a 2004 Series 11 Polaris Ranger 6x6 that is ideal for traversing a rugged environment. (See Figure 1.1 on page 3)
- Members of our team have personal relationships with those who serve as fire responders.
- Other members have access to the types of sensors and testing locations that are ideal to implement a vehicle that can aid in any post-fire investigation efforts.

So, we were motivated to design an unmanned vehicle that can specifically examine a post-forest-fire environment and ultimately keep fire responders out of hazardous situations.



Figure 1.1: Polaris 6x6 Ranger

## 1.2 Literature Review

The sources reviewed have revealed a need for an unmanned vehicle that can assess the hazards, particularly pertaining to air quality, that lurk in a post-fire environment. "Guide for fire and explosion investigations" by the National Fire Protection Association gives a detailed procedure that fire responders follow while investigating the cause and damage of a fire[?] and "Evaluation of hazards in the post-fire environment" by The Inter-agency Board reveals the specific risks fire responders are exposed to while investigating a fire[?]. More information can be found in the remainder of this section. These two sources in particular helped inspire and shape the functionality of our vehicle and provided the motivation behind using an unmanned vehicle to assess a post-fire environment. There is a need for an unmanned vehicle that can help determine whether a post-fire zone is dangerous for fire servicemen, law enforcement officers, and various other people to enter, even with protective clothing and masks.[?]

After a fire, there are very deliberate steps taken to gather information on the state of the environment. This is mainly done to determine whether the post-fire zone is safe for humans to enter, whether it be for "victim recovery, salvage and overhaul, origin and cause investigation, or criminal investigations." [?] The general post-fire assessment steps and concerns are highlighted in Figure 1.2.

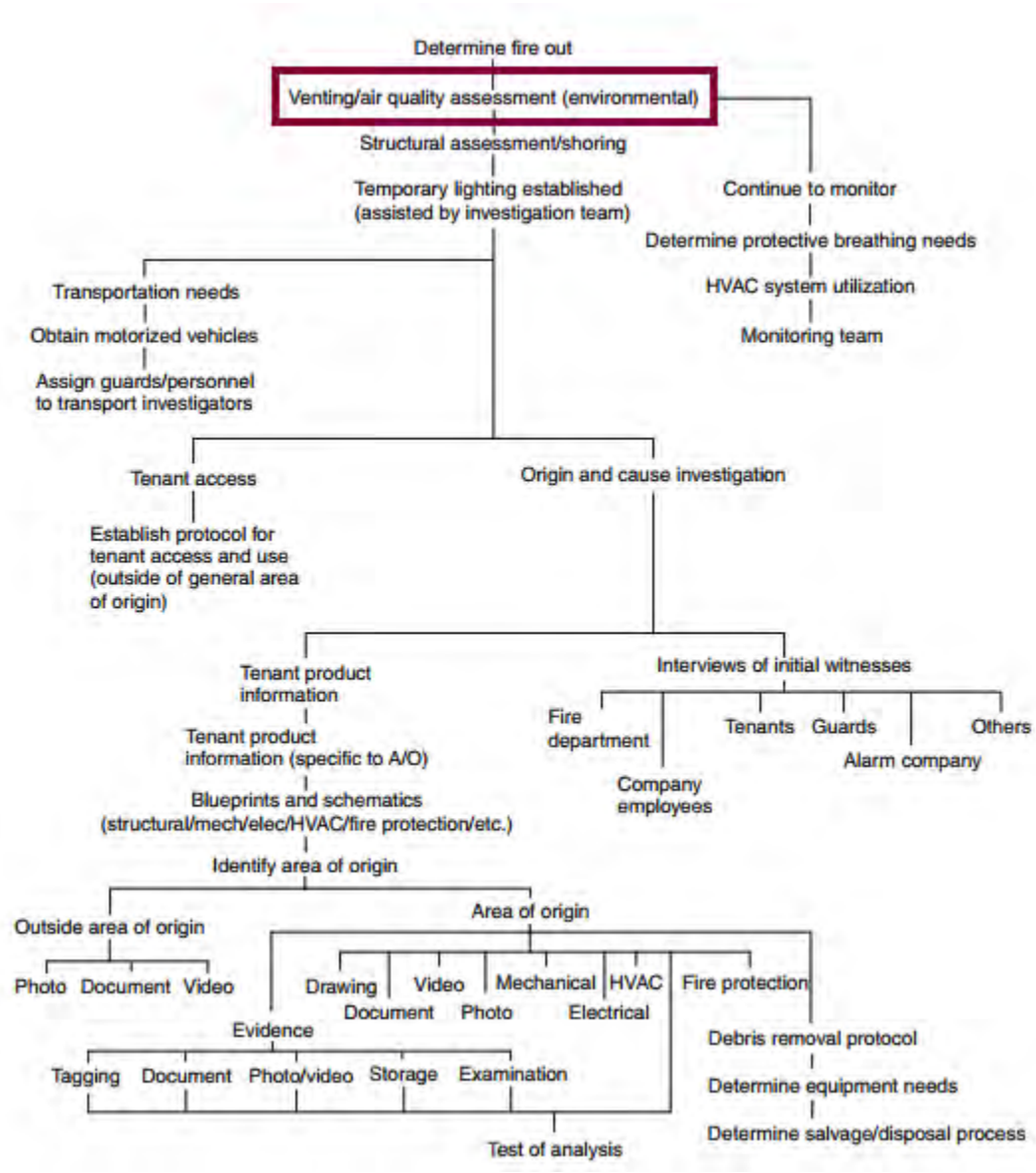


Figure 1.2: Fire Investigation Flowchart [?]

During the "Venting/air quality assessment step" in Figure 1.2, there is a sub-



step that prompts the fire investigators to "determine protective breathing needs." [?] This is the primary focus of our vehicle's functionalities.

Currently, in order to assess the types of air quality risks within a post-fire environment (and therefore the types of protective breathing needs/clothing), a responder must enter the environment equipped with some degree of protective gear and various sensors. These sensors include, but are not limited to, multi-gas detectors and air particulate detectors. [?] Once the environment is assessed and the proper protective gear is chosen, the post-fire procedure can commence. Therein lies the problem. In order to assess the harmful gasses and particulates in a post-fire zone, a person must enter said zone. This person could potentially be exposed to excessive: carbon monoxide, carbon dioxide, air particulates, and toxic chemicals from the fire and fire retardants. [?]

We have used the sources we have found to develop our vehicle in such a way that it can perform all of the tasks that a responder would, as well as provide additional data that can be used to further ensure that no humans will be harmed during the post-fire assessment/investigation/recovery procedure. Our vehicle has been equipped with multiple useful sensors that reveal air quality conditions to operators who are controlling the vehicle from a safe distance. The operators also receive a live video stream as the vehicle traverses the environment. All of the relevant information can be visualized and analyzed through our web-based user interface.

In addition, our vehicle gathers data from which 3D maps can be generated, a capability that is beyond what can be collected by a human responder. This 3D map can serve as a resource for other people who will enter the post-fire environment. The operators will be able to identify "hot spots" and other notable zones. When assessing post-fire environments, it is useful to keep track of particularly hot or dense, gaseous zones in order to maximize safety for the humans involved. [?]

The sources we found have provided information on and inspiration for the exact role our vehicle can serve in aiding those who assess post-fire environment. Our

vehicle will help operators determine the state of the environment and keep humans out of any particularly hazardous situations.

## 1.3 Vehicle Background

The vehicle was built for the 2004 and 2005 DARPA Grand Challenge competitions by a private team called Team Overbot [?]. Unfortunately, the vehicle did not meet requirements for the challenge either year and was then donated to University of California Santa Cruz. Students at UC Santa Cruz worked on the vehicle trying to expand its autonomous functions. UC Santa Cruz then donated the vehicle to Santa Clara University for use in the Robotic Systems Laboratory. The vehicle was in a state of disarray after being passed around for ten years.

The vehicle is a 2004 Series 11 Polaris Ranger 6x6 [?]. It contains a single-cylinder carbureted gasoline engine which produces approximately 40 horsepower. The two rear axles have fixed differentials, forcing the four wheels to always turn at the same speed. It has a rated top speed of 40 mph and the bed is a tilting unit and raises up allowing contents to be dumped out of the vehicle.

Two previous Senior Design teams worked on the vehicle in 2014 and 2015. The 2014 Senior Design team focused on creating an autonomous vehicle testbed. They successfully developed a software control system to operate the vehicle via a remote console, redesigned the wiring of the vehicle, and implemented safety features. They used Arduino microcontrollers to control steering, throttle, brakes, transmission and the parking brake [?]. All of these microcontrollers and emergency stop safety features are still present and in use in our current system.

The 2015 Senior Design team attempted to build on the system developed by the 2014 Senior Design team and add sensors and microprocessors to give the vehicle autonomous capabilities [?]. At the end of their year of work, the 2015 Senior Design team successfully implemented obstacle detection using a LIDAR unit. When the

vehicle detected an object in its path, it would make an emergency stop in order to avoid collision [?]. Currently, the vehicle still houses the LIDAR unit but no longer performs obstacle detection, as we have developed our own software using the LIDAR.

## 1.4 Problem Statement

Our goal was to design and implement an unmanned vehicle that gathers and relays information on potentially hazardous environmental conditions back to its operators.

To do this, we planned to accomplish the following:

- equip the Polaris 6x6 Ranger with appropriate sensors and cameras to determine how safe the environment is for humans to enter
- use GPS and laser scans to generate a 3D map that operators can use to define certain zones as particularly dangerous.
- incorporate partially-autonomous sensing capabilities that help the operator successfully drive the vehicle by remote control

The result is a rugged, advanced vehicle that can be used to protect fire responders from any lingering hazards during the investigation of a post-fire environment.

## 2 Systems Level Design

### 2.1 Customer Needs

Even after wildland fires have been contained, they present many potentially hazardous environments for people. Residual smoke can make large areas of land unsafe for human traffic, especially for prolonged periods of time. Fire fighters, however, have to continue to work in these areas after the initial fire is contained, exposing themselves to unsafe levels of contamination. This creates a need for an unmanned vehicle that can be deployed in backcountry areas.

In order to better understand the needs of the current market, our team interviewed some potential users. Three persons were interviewed: Antoine "Chidi" Untamed, a Forest Firefighter at ASI Arden Solutions Inc., Max Reese, a Volunteer Search and Rescue First Responder, and Matthew Perez, Lieutenant Firefighter, Santa Clara Fire Department.

When interviewed, interest was expressed for a product that could provide ground level environmental sensing to firefighters. Mr. Untamed talked about how after a fire, it is the responsibility of people to go in and analyze the area. Though they are provided with aerial maps taken by Unmanned Aerial Vehicles, they oftentimes are still put in dangerous situations that an unmanned rover could be in instead. The dangers these people face include unsafe air to breathe, the chance that the fire is not entirely out, and "snags," which are different branches, logs, etc., that can get caught on the protective equipment of the firefighters and that can cause the protective equipment to tear, or even to cause trees to fall down on or near them.

Mr. Untamed also expressed concern over the efficacy of drones because of FAA regulations that limit usability, unpredictable wind conditions, and limited visibility from smoke. He stated that especially during the mornings, smoke has a tendency to sit on top of the canopy and spread across the canopy, rather than simply rise from the source of the fire. This causes the aerial view from the UAV's to not provide

usable data. He says that in one situation, people tried to use this data to pinpoint the location of remaining fire that a helicopter could then air-drop water on the fire, and the helicopter completely missed the source of the fire. When asked what kind of data a person going into a post fire situation would have to find, he responded, "They would need to find forestry trails, if and where the fire was still going, and see if the air was safe to go into the area without a mask. [Also] the temperature would have to be reported."

Mr. Perez talked to us about his experience as a fireman, and even though his work mostly dealt with residential fires, our project interested him, and he saw a lot of good uses from it, including air quality sensing. He was especially concerned with combustible gasses in the air that could ignite and quickly restart a fire. Mr. Perez also suggested having a larger vehicle that, instead of being towed to a scene taking up space, could be driven to the location carrying the gear of the firefighters traveling with it. Mr. Perez was interested in a vehicle that could provide data to multiple sources, including fire fighters and trained search-and-rescue teams. The summarized result from these interviews is shown in Appendix B. The resultant design requirements and customer needs are listed in Appendix A

To fill those needs, the prototype vehicle has been outfitted with an array of environmental sensors to monitor air quality, LIDAR units, marine batteries, and a forward looking infrared camera. Although multiple passenger capabilities is a requirement, the prototype vehicle is unable to provide space for multiple passengers.

## **2.2 Key Requirements**

In order to ensure that we met all of our customer's needs, we identified and sought to meet a series of requirements. The full list of requirements is included in Appendix A. The most critical requirements are outlined in 2.1.

We chose these Key Requirements because they specifically address the needs of

Description	Requirement
Range - Remote Operator Console	150 meters
Range - WiFi Network	150 meters
Latency - Cameras to Onboard Laptop	1 second
Latency - Cameras to User Interface	1 second
Latency - Vehicle State Data to User Interface	1 second
Mapping - GPS Accuracy	< 2 meters
Visuals - Cameras Blind-Spot Testing	360 degrees

Table 2.1: Key Requirements

the customers we interviewed. We sought to design an intuitive, long-range vehicle that disseminated critical data in close to real-time and provided accurate mapping and localization information. These Key Requirements heavily influenced our decisions during the design and development of our vehicle. These requirements were verified through a series of tests we designed, which is discussed in detail in Chapter 10.

## 2.3 System Level Sketch and Use Cases

The primary use case for the vehicle is during the mop-up phase of forest fire fighting operations. As forest fire fighters are carrying out mop-up operations, unknown environmental hazards, such as carbon monoxide, low oxygen levels and high air particulate concentrations, present a real and imminent danger to fire fighters.

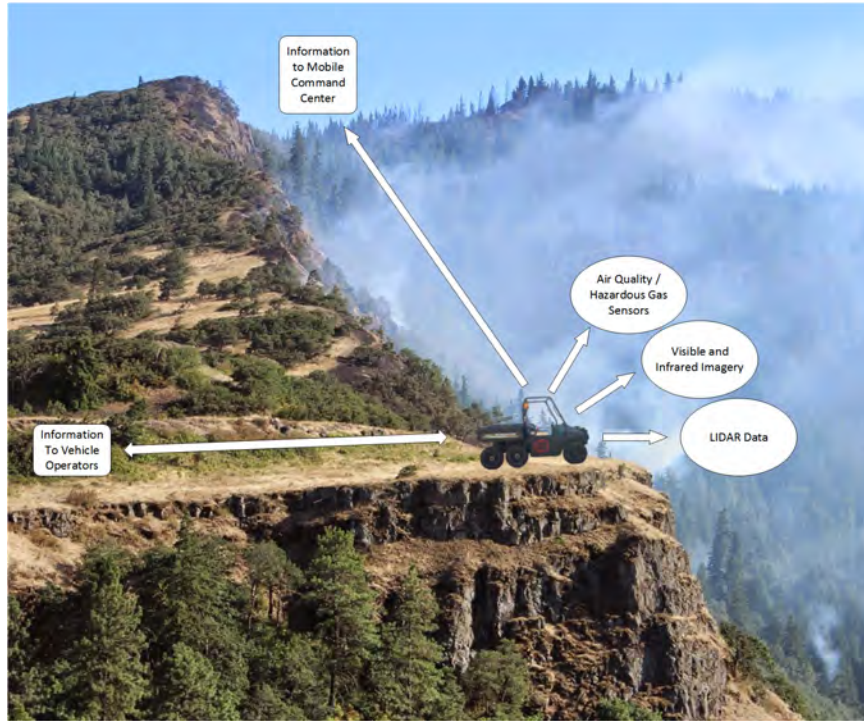


Figure 2.1: Use Case Scenario Illustration

For example, while conducting fire fighting operations, the fire fighters make the decision to send the rover into an area which has been flagged as potentially hazardous. The operators manually operate the vehicle to position themselves and their gear near the danger zone. The operators can then quickly dismount and setup the teleoperation command center. The vehicle then proceeds into the potentially toxic environment, measuring the concentrations of potentially toxic, airborne substances and relaying that information back to the operators. While one operator focuses on driving the vehicle, an analyst focuses on the sensor streams, including the environmental sensors, monitoring harmful gas concentrations, and the LIDAR data, identifying physical obstacles which may be difficult to see in a visually impaired environment. Both operators will also relay, via a satellite communications link, images, videos and data to the forest fire fighting mobile operations center where strategists can best decide how to proceed with fire fighting operations. (See Fig 2.1)

## 2.4 Functional Analysis

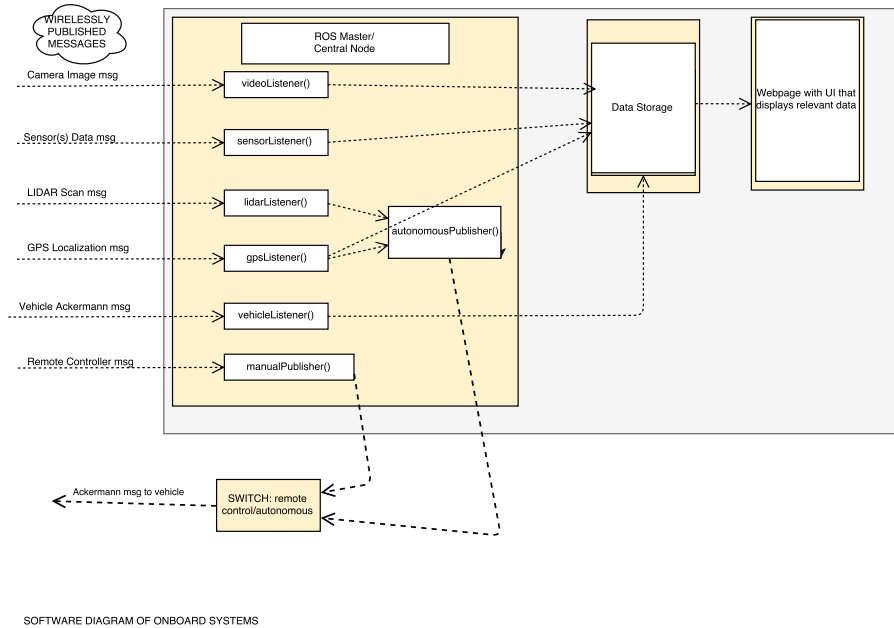


Figure 2.2: Software Component Block Diagram

The vehicle is categorized into five subsystems: environment sensing, operator control and user interface, communications, power, and localization. The breakdown of the two primary systems, hardware and software, can be seen in figures 2.3 and 2.2 and show how these subsystems fit together. The environment sensing subsystem includes the sensor packages, cameras, and LIDAR units. The operator control and user interface subsystems are the systems that control the vehicle remotely as well as the data presentation components. The vehicle must communicate back and forth with the operator, receiving driving commands and sending data readouts. This is categorized under the communications subsystem. All powering is classified by the power subsystem. The final subsystem is labeled localization and it includes the ROS architecture involved with determining vehicle position for accurate mapping.



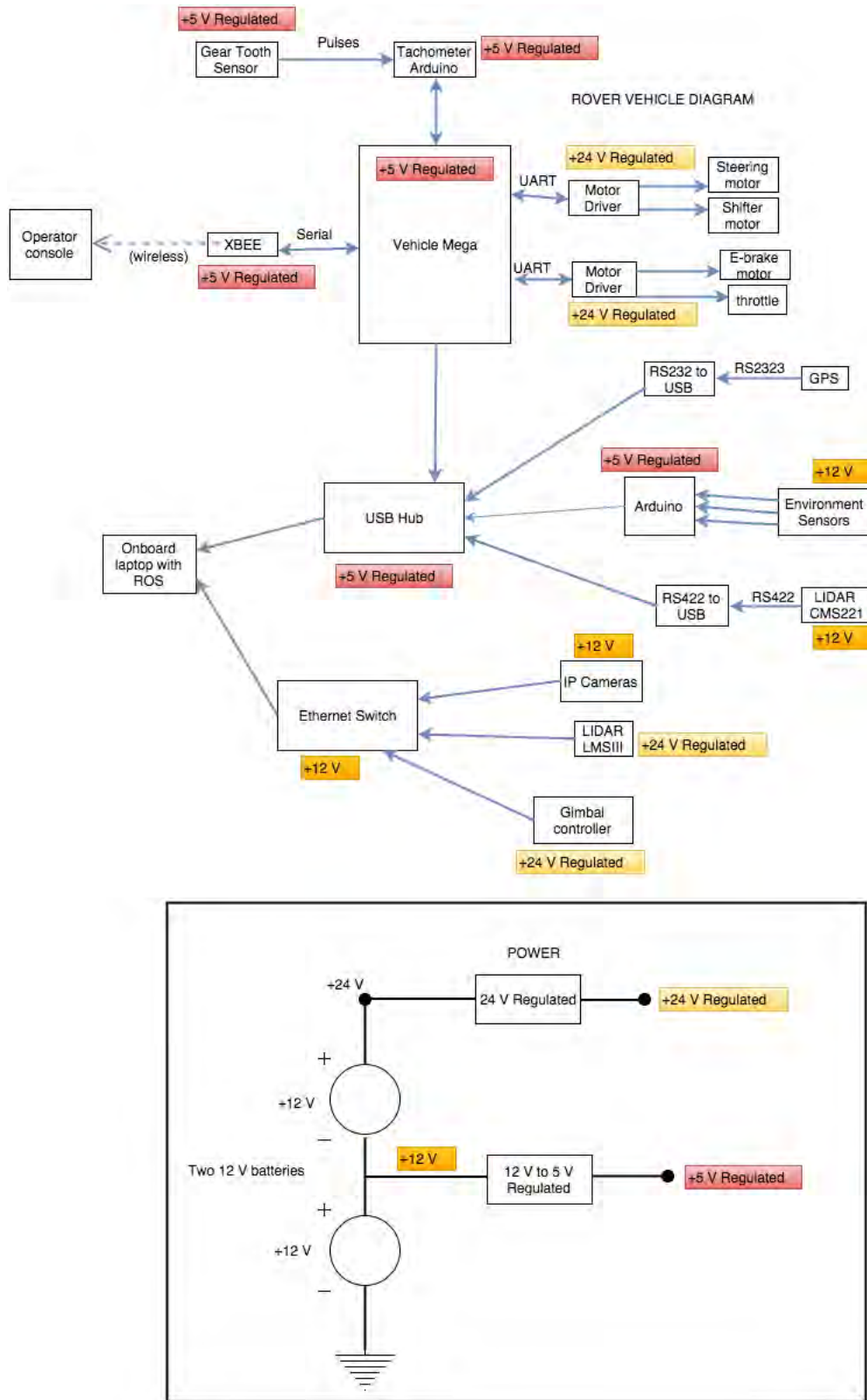


Figure 2.3: Mech/Elen Component Block Diagram

## 2.5 Benchmarking Results

Many all-terrain, unmanned vehicles on the market are for military contracts exclusively. However, Northrop Grumman and Argo Robotics both have products that fit the category of an unmanned, all-terrain vehicle that can be equipped with sensor packages for non-military purposes. Some of the specs for these products can be found in Table 2.1. These are the Argo J5 Mobility Platform (Fig 2.4) at thirty-nine-thousand dollars, the Northrop Grumman Andros F6 (Fig 2.5), and the Northrop Grumman Wheelbarrow Mk9 (Fig 2.6). Prices for the two later vehicles are not available without a direct quote.

Product	Height(m)	Width(m)	Length(m)	Sensor Packages	Features
J5 Mobility Platform	0.83	1.38	1.52	1 Camera Mount	Amphibious
Andros F6	1.486	0.445	1.32	5 Cameras	Extendable Arm
Wheelbarrow Mk9	1.24	0.63	1.24	Up to 10 Cameras	Extendable Arm

Table 2.2: Current Product Comparison



Figure 2.4: Argo J5 Mobility Platform



Figure 2.5: Northrop Grumman's Andros F6



Figure 2.6: Northrop Grumman's Remotec Wheelbarrow Mk9

While these products that are on the market are the most similar to the product we wish to produce, the issues we wish to address are not solved by these rovers. Neither of Northrop Grumman's rovers have the housing needed for sensor packages necessary to monitor atmospheric conditions in forest fires. The Argo rover also does not house the necessary data packages, and while it is amphibious, inclusion of a

sensor housing is more important.

Other aerial drones are also in the market for unmanned vehicles capable of relaying data such as thermal imaging and video back to the user. Two such products are the Elimco UAV-E300 (Fig 2.7) and the Sensefly's EBee (Fig 2.8). Both of these products have the capabilities necessary to map and relay usable data, and are marketed as disaster response vehicles. However, there are some inherent weaknesses to their aerial nature. Mr. Untamed has pointed out that images provided by aerial vehicles can oftentimes be unusable as smoke can sit on top of the canopy of a forest, especially in the mornings. They also can only provide overhead data, and may not be able to map certain elements disaster responders could need, such as air quality at ground level or mapping paths that are safe. Our vehicle seeks to get under this layer of smoke to provide data from the ground level.



Figure 2.7: Elimco UAV-E300



Figure 2.8: Sensefly's UAV EBee

## 2.6 System Level Issues, Trade-off Analysis

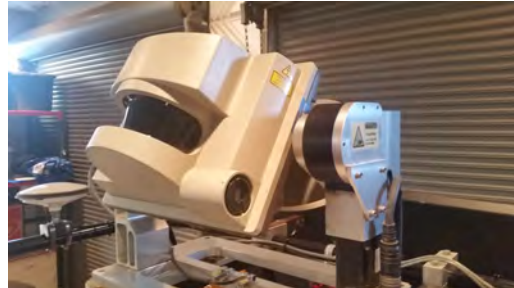
The design process for this vehicle required that we opt for certain configurations and components over alternatives. The biggest design decisions we had to make were on the LIDAR layout and environmental sensing systems.

### 2.6.1 LIDAR Physical Configuration

Two LIDAR units, a SICK LMS221 on a pivoting mount (Fig 2.9b) capable of generating a 3D point cloud and SICK LMS111 2D linescan LIDAR (Fig 2.9a) capable of producing 2D obstacle maps, were made available to our team through Santa Clara's Robotic Systems Lab. One design challenge was to determine the optimal location for these two sensors on the vehicle. The total combined field of view, blind-spots, total weight, material cost, view height and vehicle vertical clearance were all taken into account when deciding on the configuration.



(a) LMS 111: 2D LIDAR



(b) LMS 221: 3D LIDAR

Figure 2.9: LIDAR Types

A final configuration was chosen based on the Concept Scoring Tradeoff Analysis in Appendix C. The final configuration has the 3D, articulated LIDAR (LMS221) mounted on the top of the roll cage where it has a full forward and rear field of view. The 2D LIDAR (LMS 111) has been mounted on the front grill of the vehicle where it can mitigate a blind spot immediately in front of the vehicle.

## 2.6.2 Sensor Physical Configuration

The hazardous environment sensing requirement has been filled by three redundant sensor packages placed on the hood and each side of the roll cage (See Figure 2.11). The sensor packages are enclosed in a box with a forced air inlet to continually monitor the air for changes in composition while also protecting the components. The enclosed array of sensors are able to detect temperature and concentrations of: carbon monoxide, methane, carbon dioxide, hydrogen, air particulate, and liquefied petroleum gas. The cost of an oxygen sensor is prohibitive, so oxygen concentrations must be extrapolated from alternative sensors. These sensors could be expanded as needs arise. The vehicle is equipped with redundant packages for a multitude of reasons. If a sensor fails, there are two other identical sensors to fill that role. Additionally, having the sensors spread out on the vehicle, it can be determined if concentration readings are representative of the greater area, certain elevations, or a single point. The data from the redundant sensors could be analyzed in three ways; the greatest reading could be displayed, the reading most common among sensors, or all three could be displayed. The first option, outputting the highest value, is the most conservative option while also being concise, but it makes it possible that false data could be transmitted. The most thorough method would be to transmit all three data streams and allow the operator to decide which to trust.

The location of the air quality sensors and cameras was the next critical design decision. See Figure 2.10 for examples of air quality sensors and cameras. For the cameras, the field of view, mounting costs, implementation time and robustness, among other factors, were considered when deciding on a final design. Additionally, for the environmental sensors, the cost, implementation time, robustness and proximity to the vehicle's exhaust systems were considered when deciding on a sensor configuration. Appendix C features a complete tradeoff analysis for the sensor configurations.



(a) Carbon Monoxide Sensor



(b) IP Camera

Figure 2.10: Sample Sensor Types

A final configuration was chosen where the cameras are mounted on all 4 sides of the roll cage, allowing full coverage of the surroundings while also allowing the operator to see over obstacles. The redundant sensor packages are located on the front of the roll cage and on the front of the vehicle.

## 2.7 System Level Architecture

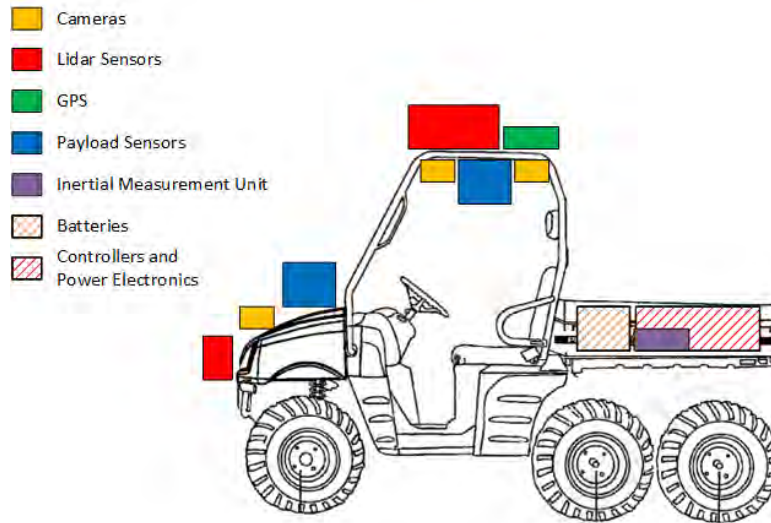


Figure 2.11: System Layout Architecture

Figure 2.11 shows the physical layout of the subsystems of our design. There is a large LIDAR unit on a gimbal mounted on the top as well as a smaller, stationary LIDAR unit on the front bumper of the vehicle. These two LIDAR units work in unison to gather data to create 3D maps of the environment. The top roll bar also houses the GPS antenna, environmental sensors, and cameras. The redundant environmental sensors are mounted on the front bumper of the vehicle. The controlling electronics and power are housed in the trunk of the vehicle to protect them from hazards.

## 2.8 Team and Project Management

As the RSL Rover project is a continuing project through Santa Clara's Robotic System's Lab, we were provided with a workshop right off of the engineering quad which is outfitted with computers, tools and workspaces. We were also provided some financial assistance in purchasing some tools which will stay with the RSL and will continue to be used by teams in the future.

One of our major constraints was our budget. We were awarded a grant of \$2500 from Santa Clara's School of Engineering (Undergraduate Programs Funding). We used this funding to purchase sensors and raw materials for the environmental sensing packages, repair the vehicle and purchase various electronic components in order to support the new computing and sensing packages. A complete, line-item budget can be found in Appendix: D.

Due to the scope of our project, an accelerated timeline was required. During fall quarter, we successfully integrated GPS and LIDAR sensors, including power and communication systems, onto the vehicle and successfully integrated these sensors, along with driving functionality to the ROS (Robot Operating System) software platform. During winter quarter we designed the environmental sensing packages, and integrated cameras and an inertial measurement unit on the vehicle and began de-



velopment on a web-based GUI application to present this data to first responders. During spring quarter we successfully integrated all of our subsystems and ran extensive field testing to validate our design against our engineering requirements. See Appendix E for a complete Gantt chart summarizing our schedule.

As a team we dedicated ourselves to this task, in the hopes of completing the listed goals by the end of the year. We held each other accountable by meeting at least twice a week, once under the supervision of our advisor and once on our own. At each meeting, we went over the work we had done in the previous week and talked about tasks, and who would be best suited to handle each task. Patrick Barone has served as the primary leader of the group, however, each of us has taken initiative in leading the group on different occasions or when undergoing certain tasks. Despite the fact that we are split between two majors, we have been able to always communicate effectively about our projects, successes, failures, and insights to others. We have all contributed to the project as a whole and to the research process in order to inform our design.

To begin our design process, we first assessed the state of the vehicle as it had already been modified by two senior design teams in previous years. From there, we assessed how the rover could be changed, fixed, or left as is in order to make it best suited for disaster response, especially in the case of forest fires. During this time, we researched what was actually needed in these disasters by interviewing disaster responders and reviewing literature; in addition, we researched current solutions and similar market products, paying close attention to their strengths and shortcomings. We then created a criteria for ourselves and set goals and a timeline. We then began to design our system layouts, while assessing each possible design choice to find the best choice.

This project, as a full sized, unmanned vehicle that will enter disaster sites, is subject to many risks, but we sought to manage these risks as we feel this project is worth undertaking. First, as this project is unmanned, we run the risk of working with

technology that the public has not yet fully accepted. Many see robotic technology as something to fear. In order to address this fear, we designed our system as a whole to make our vehicle not only functional, but also intuitive and aesthetically pleasing.

Also, as an unmanned vehicle, there is no human to make snap decisions, such as braking. In order to mitigate these risks, we leveraged the already-existing emergency stop button system that can quickly and effectively stop the vehicle in an emergency. This allows human controllers to make snap decisions to stop the rover before anything bad can happen. For example, if the vehicle is not driving as anticipated and it is on course to hit an object, or even a person, we can quickly stop the vehicle.

As the vehicle is fully-sized, our team ran the risk of causing injury to ourselves and others as we put the car up on jacks to work with it; consequently, we established safety guidelines in order to ensure that we kept ourselves and guests working on the machinery with us safe. The approved safety protocol is included in I. As the vehicle itself is also a few years old, we ran the risk of it breaking down on us. To prevent this, the team performed extensive work, through preventative maintenance, to ensure that the rover is kept in a well maintained state. As for the risk of us putting the rover into unstable disaster sites, we ran the risk of the environment harming our equipment. We conducted research on heat shielding in order to protect key features on our rover; however, the solutions discovered were both cost prohibitive and also not critical to the robotic and sensing functionality that we aimed to prove through this project. That being said, we housed most of the delicate electronic equipment under a layer of protection. The roll cage, the hood, and the trunk cover all provide a sufficient level of protection for sensitive equipment.

## 3 Subsystem: Environmental Sensing

The environmental sensing subsystem includes the air quality sensors and video imaging array. The environmental sensing subsystem is an integral component of the fire monitoring functionality of the RSL Rover. It is the system that allows operators to gather visual and atmospheric information, an essential component for assessing fire-scene safety. The air quality sensors must display accurate information because they determine what gear is required to enter the area. The video imaging array is critical as it allows the operator to successfully see where the vehicle is within the environment, thus making it easy for the operator to drive through and visually inspect the environment.

### 3.1 Air Quality Assessment

The air quality sensing subsystem consists of three redundant packages distributed on the front and top of the vehicle. Using three independent but redundant packages improve the accuracy of results since operators then have multiple values to compare against if there is a spike or other anomaly. It also allows a safety buffer if one or more sensors were to fail. Within each package there is an array of eight gas sensors, two temperature sensors with different temperature range sensitivities, an air particulate sensor, and a humidity sensor. The eight gas sensor array detects atmospheric concentration of the following: liquefied petroleum gas, carbon monoxide, carbon dioxide, natural gas, and hydrogen gas.

#### 3.1.1 Payload Requirements

The list of important gases to detect was derived from the National Fire Protection Association handbook as they are the gases produced by a fire most hazardous to human life. Carbon dioxide is used as an indicator for fire because it is a primary product of complete combustion while carbon monoxide is a product of incomplete

combustion. There are numerous components to smoke, most of which are hazardous, so the air particulate sensor is responsible for the detection of the smoke hazard indicator. In addition to the hazards produced from combustion, forest fires also increase the risk of residential gas leaks and explosions. Some jurisdictions use liquefied petroleum gas for residential heating and cooking while some use natural gas. Thus, liquefied petroleum gas and natural gas must both be detected to limit risks from gas fires. Atmospheric air temperatures can increase dramatically from large forest fires so it is important that the sensor package be able to detect fine changes at lower temperatures while still being able to detect high temperatures that are dangerous for firefighters.

### 3.1.2 Component Selection

Gas sensors on the market are incredibly expensive, detect only a limited range of gases, and require complex, expensive calibration equipment. Instead of trying to implement this into the RSL Rover, we started from scratch and utilized the MQ-series sensors pictured in figure 3.1.



Figure 3.1: MQ-Series gas sensor

These sensors use a reactive filament that changes resistance according to the concentration of gases it is sensitive to. This makes for a robust, low cost solution

that can easily be calibrated to show presence of a wide variety of gases. These sensors are sensitive to humidity, so a humidity sensor is also incorporated into the sensor package to allow for humidity corrections in the field. In addition, two temperature sensors are included to provide accurate temperature readings up to 250° Fahrenheit, which is beyond the safe limit for firefighters. The final sensing component was the air particulate sensor. For this, we utilized a Sharp GP2Y1010AU0F smoke and dust sensor picture in figure 3.2.

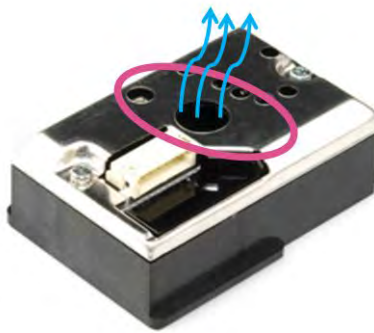


Figure 3.2: Sharp air particulate sensor with air flow

The device works by pushing air through the center hole where it reflects infrared light off the smoke or dust. The more light that is reflected into the infrared receiver, the higher the smoke concentration is. The small voltage from the receiver is then amplified to a range of 0 to 3.3 Volts where 3.3 Volts is the maximum detectable concentration.

Each of these sensors requires its own circuitry and mounting hardware; to avoid large amounts of wiring and to give the sensors a place to sit, we created a printed circuit board that interfaces with each sensor. The board does not process the readings; an Arduino Mega 2560 is used to read and scale the analog voltages and digital readings from the sensor array. This is a low cost solution that allows for future modification as well as the planned modularity of the sensor package. The current printed circuit boards could be swapped out for a new circuit board with different sensing capabilities without having to change additional hardware. The data collected by the

Arduino is processed on the onboard computer using ROS, limiting the impact on the Arduino.

### 3.1.3 PCB Design

We used kicad, a free CAD program, to design the entire circuit board. The first step was designing the schematic shown in figure 3.3.

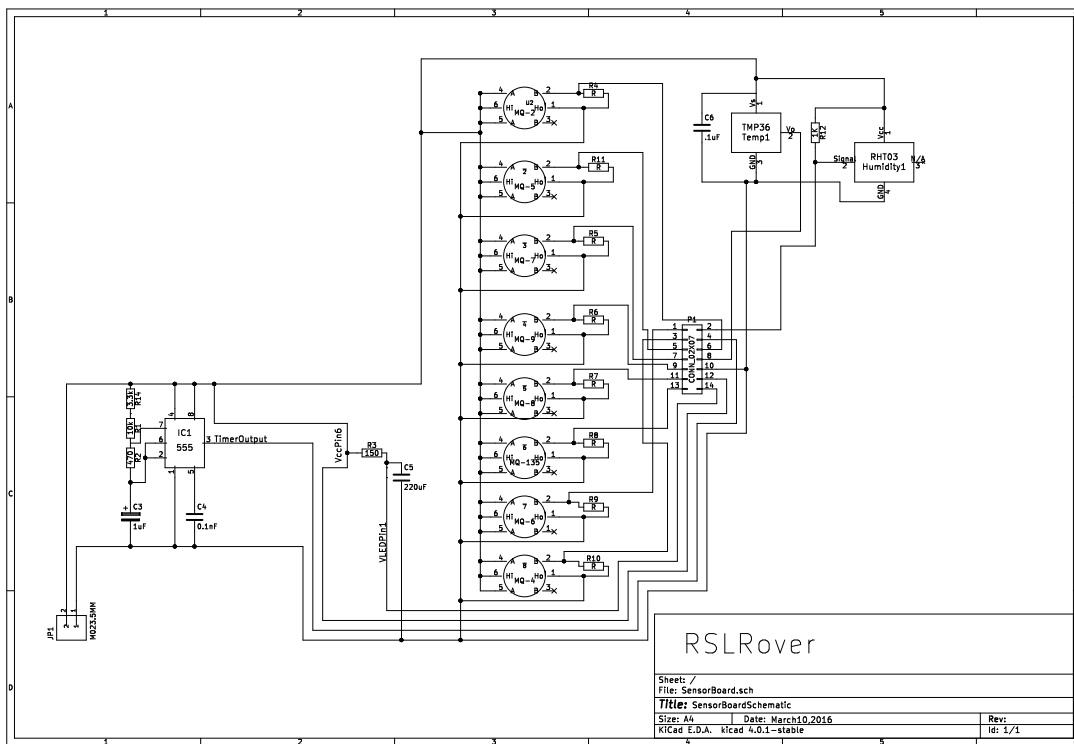


Figure 3.3: Air quality PCB schematic

The schematic includes the timing circuit for the air particulate sensor, MQ sensor circuits, humidity hookups, and temperature circuits. Originally, a 12 Volt to 5 Volt regulating circuit was included in the schematic, but it was removed due to an overheating problem during testing. Instead, an external pulse width modulating regulator was added. The next step in designing the printed circuit board was laying

out the circuits and component footprints in the layout editor. The final layout schematic is shown in figure 3.4.

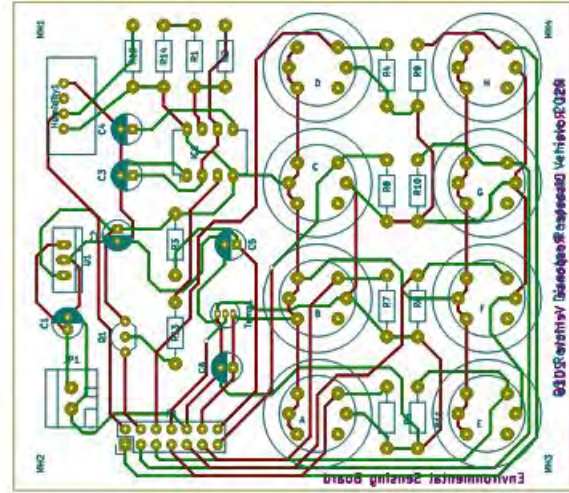


Figure 3.4: Printed circuit board layout

The design was sent to a board house where it was manufactured. We then populated the board with our sensors and tested. The first iteration had several problems, the most pronounced was the overheating of the voltage regulator. The regulator had to dissipate nearly 10 watts which caused it to get dangerously hot. We solved this by implementing a pulse width modulating voltage regulator and using a jumper wire across the original voltage regulator footprint. The next big problem we encountered was with the timer circuit for the air particulate sensor. The data sheet for the sensor showed an inverse graph of the required pulse form. To solve this, we simply removed the transistor that inverted the output from the 555 timer. All of these changes are shown in the schematic in figure 3.3 but not the circuit board layout shown in figure 3.4.

## 3.2 Cameras

The camera system is a four camera array that provides a front facing view for the operator as well as a roughly 260° view for observation of the surroundings. The initial design included four IP cameras, all of which ran as a unique node within the ROS network. ROS handles IP cameras over RTSP, the Real Time Streaming Protocol. This introduced an unacceptable amount of latency, about 4 seconds, to the camera stream. The final design includes four Logitech c615 usb cameras that are connected to a Raspberry Pi 2 Model B. A photo of the selected model of Logitech camera is included in Figure 3.5. The dedicated Raspberry Pi runs an image of Ubuntu with ROS. The usb camera images are captured through ROS and sent through an OpenCV "person detection" application that recognizes bodies within the camera stream and outlines them in a green box so that they are visible to the operator. For an example of this capability, see Figure 3.6.



Figure 3.5: Logitech c615 camera





Figure 3.6: Example of OpenCV People Detection Application

### 3.3 Sensor and Camera Layout

The location of the air quality sensors and cameras was a design decision. For the cameras, the field of view, mounting costs, implementation time and robustness, among other factors, were considered when deciding on a final design. Additionally, for the environmental sensors, the cost, implementation time, robustness and proximity to the vehicle's exhaust systems were considered when deciding on a sensor configuration. Appendix C features a complete tradeoff analysis for the sensor configurations. A final configuration was chosen where the cameras are mounted on all 4 corners of the roll cage, allowing for a 360 degree view of the surroundings while also allowing the operator to see over obstacles. The redundant sensor packages are located on both sides of the front bumper and the center of the roll cage.

# 4 Subsystem: Sensor Housing

## 4.1 Need for Housing

Our vehicle, being equipped with multiple sensors, including sensors for air particulates, natural gas, petroleum gas, carbon monoxide, carbon dioxide, and hydrogen gas, needs to protect the sensors from physical elements that could damage them while also ensuring they are installed such that the sensors are exposed to the elements they are meant to detect. We decided to house these sensors together in one housing unit. This unit serves to shield the sensors from any physical interferences.

## 4.2 Materials Used

The material for the housing unit needed to be relatively sturdy to protect the sensors, cheap and easy to machine for economic efficiency, and thermally resistant to protect our sensors from high heat. After researching many materials, acrylic was decided upon for the material to construct our housing structure. See Figure 4.1 for the favorable characteristics of acrylic.

Acrylic has relatively low thermal conductivity, meaning it can insulate our sensors effectively, and it's melting temperature is far above the maximum of 250deg  $F$  that we expect our vehicle to experience. Though we do not expect our unit to be under a great deal of mechanical stress, as the most stress our housing unit will be in is under the stress of its own weight, the compressive and tensile strengths are both well above the weight of the system. The price of acrylic is relatively cheap compared to

Property	Associated Value
Thermal Conductivity	1.3 to 1.5 $\frac{Btu-in}{hr-ft^2-deg F}$
Melting Temperature	464 to 473 deg F
Compressive Strength	14100 to 18000 psi
Tensile Strength	5420 to 10700 psi

Table 4.1: The properties of acrylic made available from the UL company

other plastics of comparable properties, and it is very easy to machine and assemble through the use of adhesives.

### 4.3 Initial Design

The first question that we addressed in the design of the housing was how do we force the air into the housing unit? Though our housing unit is designed to protect our sensors from many of the outside elements, our sensors require exposure to air. We decided to use a PWM fan for its speed control and relatively small size. The model we used is the Artic F9 Case fan, which reaches a maximum air flow of  $31 \text{ ft}^3/\text{min}$ . For our initial iterations of the design, our team decided to use the fan as an outlet for airflow, pulling air through our housing unit.

The next question facing our team concerned the means by which air could flow over and through our sensors. Figures 4.1 and 4.2 show the unique air flow needs of the sensors we used. Most of our sensors simply needed air to flow over them, as shown in Figure 4.1. Our air particulate sensor, however, needed air to rise through a very specific segment, as demonstrated in Figure 4.2. Due to this, our housing design needed to account for these means of air flow. Our initial design had two main inlet holes, one on the front of the unit, and one on the underside. The front inlet was designed to be the major inlet of air to fill our housing unit. Our bottom hole was designed specifically to intake air, pushing air directly through the air particulate sensor.

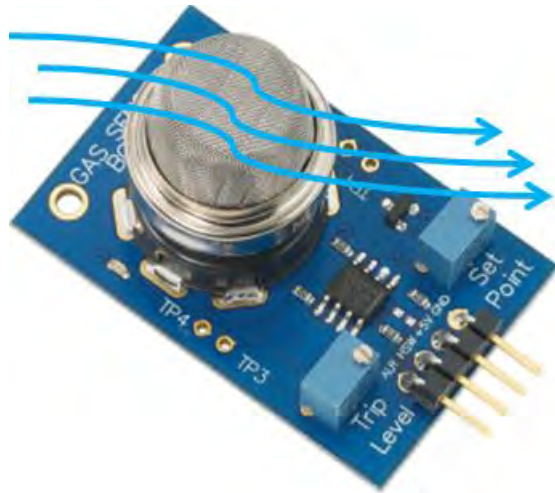


Figure 4.1: For the gas sensors to work, air must flow over the sensor

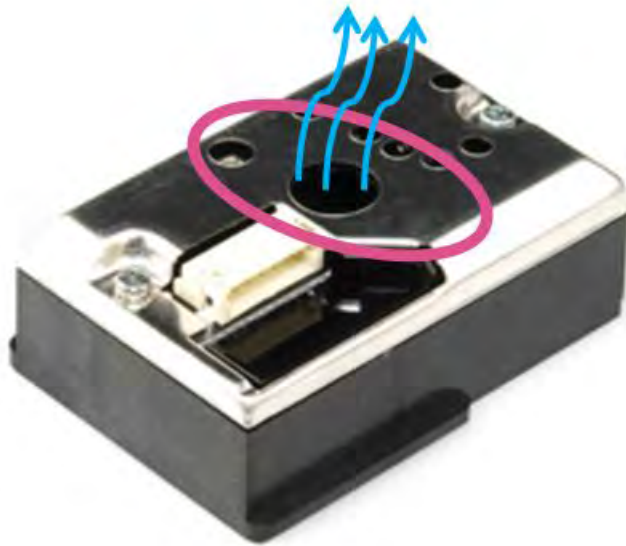


Figure 4.2: For the air particulate sensor to work, air must flow through the sensor

The major question we were left with was, will the air flow where we want the air to flow? For this a computational fluid dynamics (CFD) test was performed on our design.

## 4.4 CFD Analysis and Iterative Work

Ansys Workbench was used to perform a series of tasks designed to complete a CFD analysis. The first task was to design the internal space of the feature. This means designing a part that would represent only the area inside the housing unit. Figure 4.3 shows the design of said part.

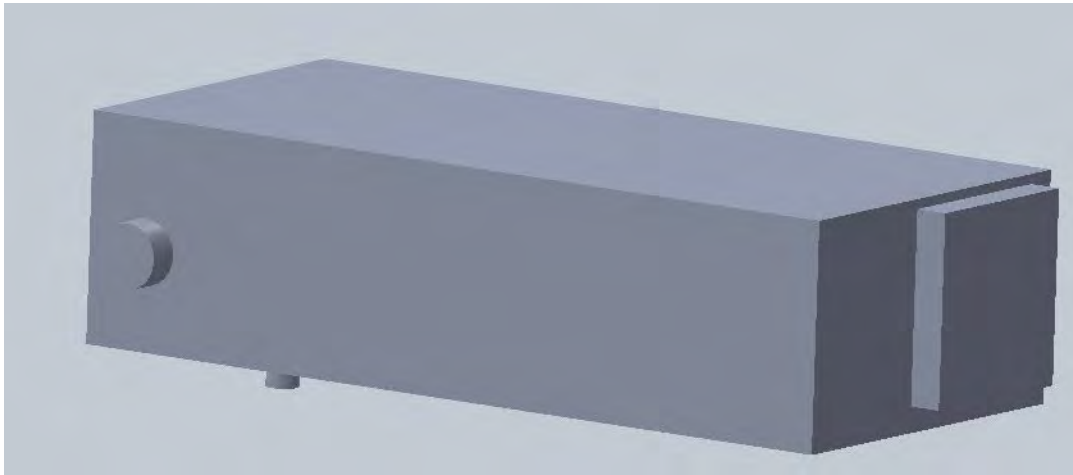


Figure 4.3: The solid designed for the initial CFD analysis that shows the initial design

After this internal space was designed, a mesh was created around the part using Ansys Workbench. Once the mesh was created and reformatted to be as precise as possible, boundary conditions could be set. No-slip conditions were set on all of the walls of the part. The inlets of the part were set at ambient pressure of 1 atm. The outlet was given a constant outlet velocity, calculated from converting the 31 ft<sup>3</sup>/min of the fan into a usable velocity of 2.6 m/sec; then this was computed by dividing the flow rate by the area of the fan and converting feet to meters. From there, a streamline map was developed for the system in order to show how successful our design was.

Our streamline map in Figure 4.4 pointed out some key failings of our initial design. Though we were successful in driving air up the hole dedicated to feeding

our air particulate sensor, we were unable to fill the area of the housing unit where most of our other sensors would be placed. Instead of the air filling up the empty space, only a few streamlines reached the center of our housing unit; many of our streamlines clung to the edge of our design.

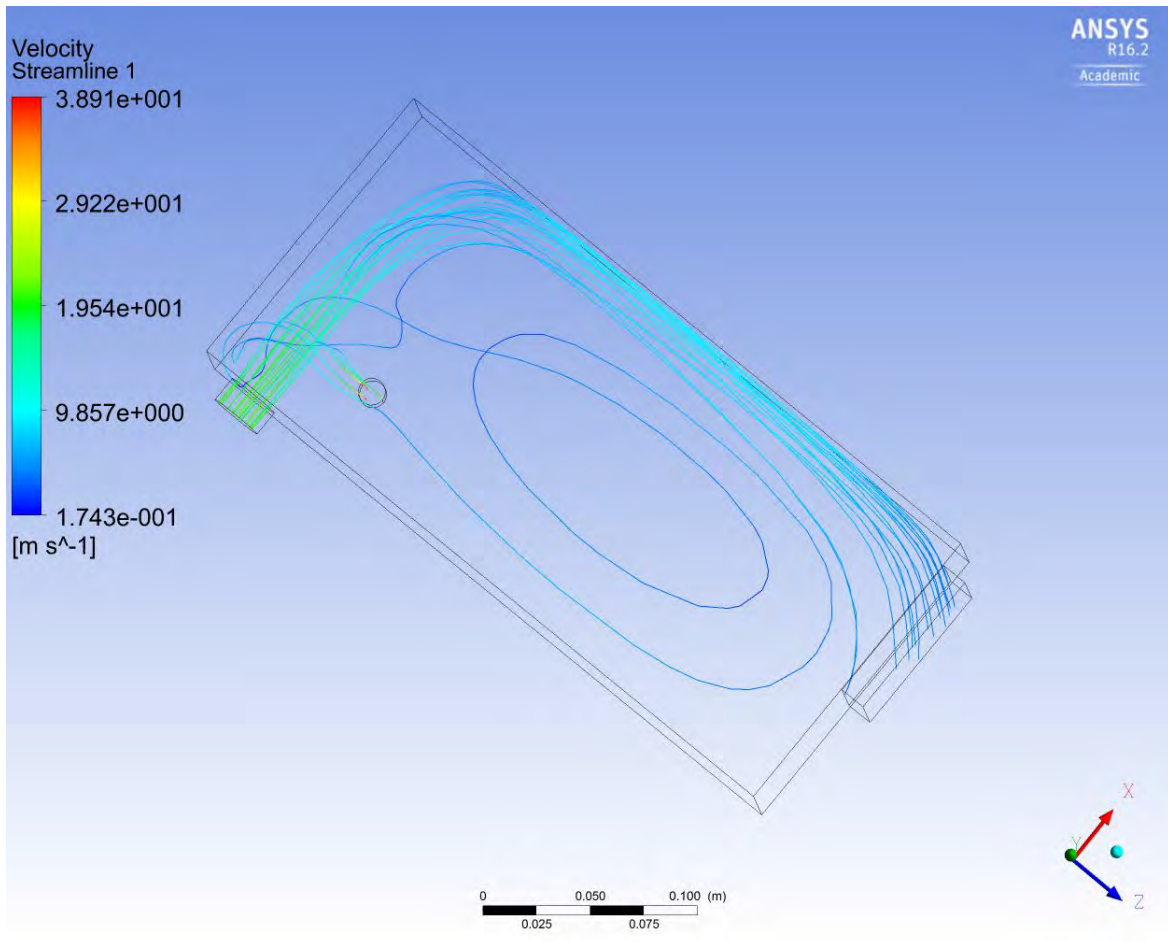


Figure 4.4: CFD streamline analysis for our first design

In our second iteration, we attempted the same tests while adding a second inlet hole on the opposite side of the housing unit to our primary inlet. This was added to push air streams into the streams currently clinging to the wall of our unit in order to send air through the central area of our housing unit, where the gas sensors would most likely be. While this did move air more successfully through our system than the prior design, Figure 4.5 shows there were still deficiencies in the streamlines as

vortices were created as the streams collided.

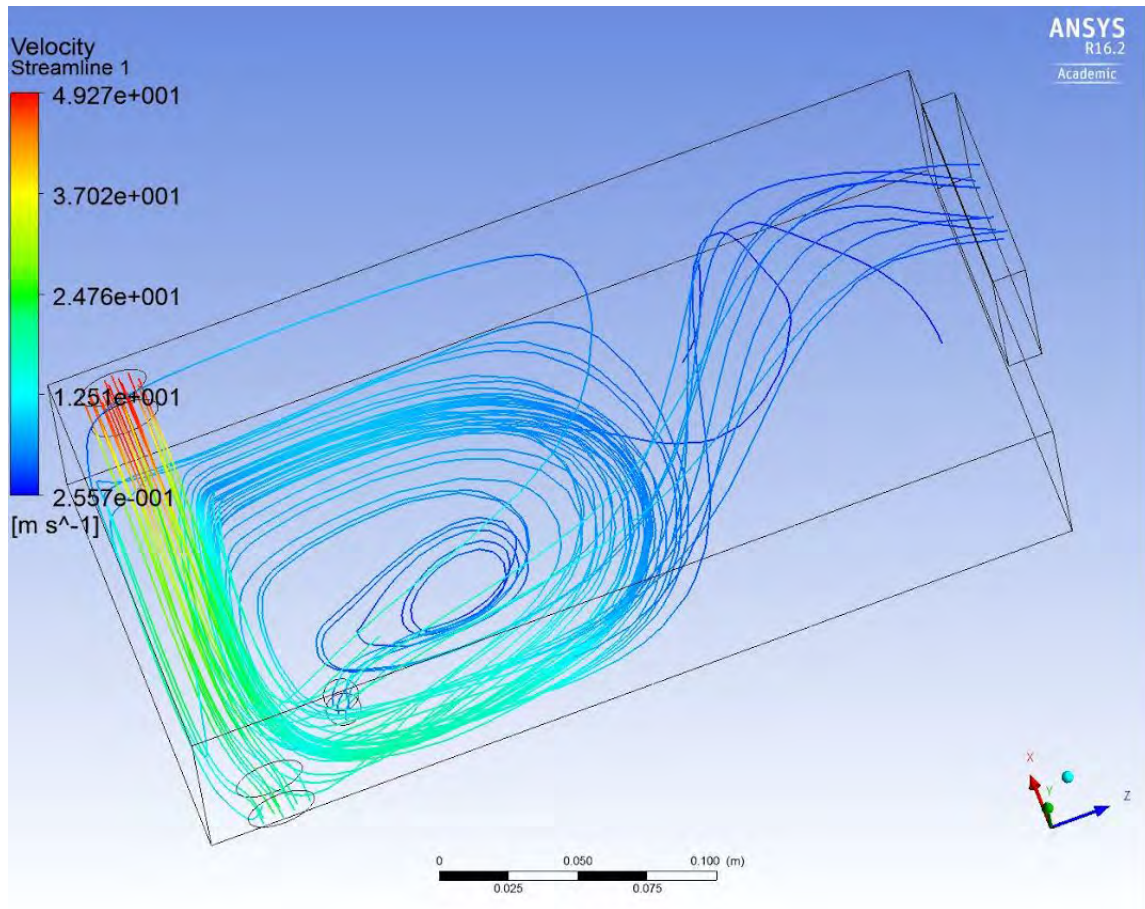


Figure 4.5: CFD streamline analysis for our second design

In our final iteration, we attempted to use the PWM fan as an air inlet instead, and we received positive results from our CFD analysis. The use of the fan as an inlet caused the streamlines to be smooth throughout most of the system until exiting through the three outlet holes, the two primary outlets and the secondary outlet dedicated to the air particulate sensor. The streamlines shown in Figure 4.6 show the air exiting directly through the air particulate sensor still, while also driving air smoothly over the area where our gas sensors would be located. From this design, we built our acrylic prototypes.

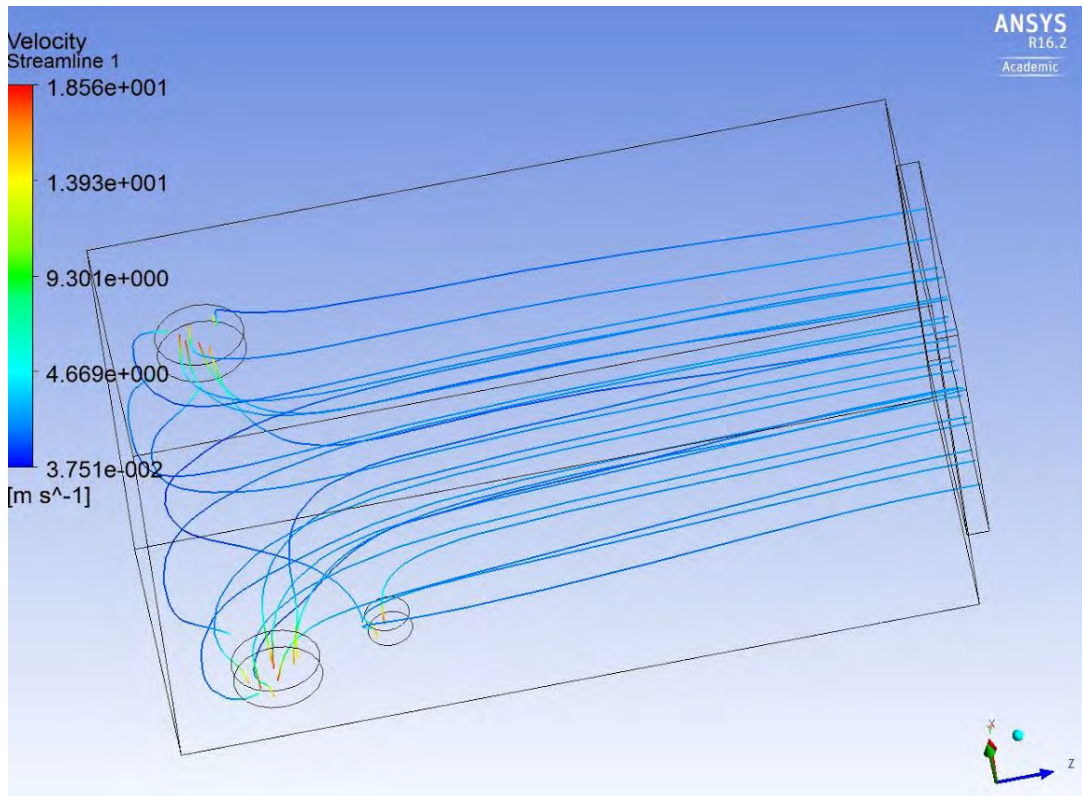


Figure 4.6: CFD streamline analysis for our final design

## 4.5 Final Design

Three housing units were made and nicknamed Larry, Curly and Moe. Each was fitted with our dedicated PCBs, which had been fitted with our gas sensors, the air particulate sensor, an Arduino unit to process the data, and the PWM fan. These fully equipped sensor units were then fitted to the roll cage of the vehicle. Two units, Larry and Moe, were installed on the right and left side of the upper bar of the front cage, shown in Figure 4.7. Allowing air to be pulled in on both sides of the vehicle. The remaining unit, Curly, was placed at the center-front of the overhead roll cage to draw air in from the front of the vehicle. This mounting strategy protected the sensor cases and allowed air intakes from all three directions of concern around the vehicle. It also created an aesthetically pleasing look.



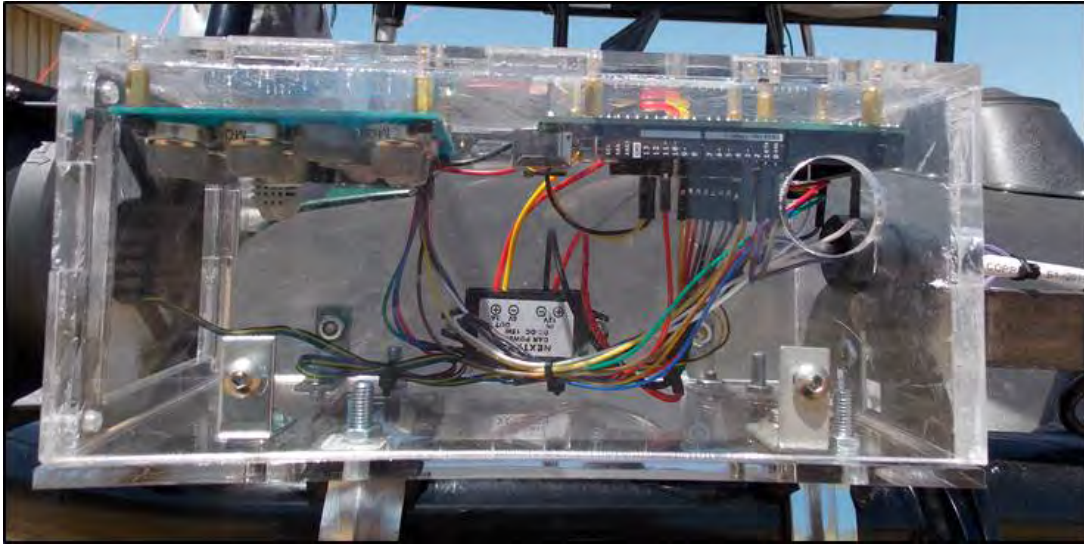


Figure 4.7: The final sensor housing design equipped with our sensors, mounted to the vehicle

# 5 Subsystem: Operator Control and User Interface

## 5.1 User-Interface Design

The inherited user interface uses a large black-box with a joystick, a series of buttons and switches, and a small LED display. The box provides all major functionality that an operator could need to interface with the vehicle. An operator can throttle, break, steer, change gears and force the vehicle to an emergency stop all from the box. This box; however, is not the best solution for our design.

Our design requires displaying a large amount of data to an operator. The small LED display on the box will not work for this goal. The black-box is also not desirable because it is hard to extend. There are a finite number of buttons and switches, many of which have already been programmed for essential functionality. Our design needs a user-interface that is:

- easy to use
- can handle displaying large amounts of information to the operator but does not overwhelm them with the info
- can be extended to provide extra functionality in the future
- can display the most critical information in a single page view . . .

Given these requirements, we decided to implement a web-based user-interface. Operators interact with the vehicle through a web-page. This web-page displays the incoming information from the sensor packages on the vehicle and incoming information about the state of the vehicle itself. Most operators are familiar with how to navigate a web-page from interacting with them on a regular basis on their own, so the assumption is that the interface should feel intuitive and natural to most operators. There is also a vast amount of research on how to develop friendly and

easy-to-use web-pages that our team leveraged when implementing our web-based user-interface. The legacy black box control system is still operational and required when operating the vehicle. The new user interface hosts the video feeds and other vehicle information.

## 5.2 RobotWebTools and the ROS Control Center

Initially our team thought about replacing the black-box entirely and allowing users to drive the vehicle by using a gamepad controller. Google Chrome and Mozilla Firefox both natively support gamepad interactions via the Gamepad API; however, we abandoned this goal to focus more on visualizing the incoming sensor data. The black-box already contains the functionality for driving the vehicle and the Gamepad API is still in active development and is not at a state where we felt comfortable using it for driving a vehicle.

After a large amount of research on how to visualize ROS data via a web-page we discovered the RobotWebTools group [?]. This group was actively developing tools for connecting to ROS from a website. The architecture works by running a webserver with ROS, and then having HTML pages connect to that webserver via JavaScript. The JavaScript library is meant to replicate the ROS architecture. It subscribes to nodes, and when that node publishes an event, the webserver captures it, converts it into a JavaScript Object Notation (JSON) message, and sends that object to the JavaScript library. JSON is a standard data format for web applications, so this makes it very easy to work with the data coming from the rover in other JavaScript libraries.

The RobotWebTools provide the low end infrastructure for building a web-based user-interface with ROS. We started to design our user-interface to be a single page view of the state of the vehicle and visualize all of the sensor data. We decided on this because a user needs to be able to focus on driving the vehicle, and forcing them

to have to click through tabs to find the data that they needed seemed unreasonable. However, the sheer amount of data that we process makes this difficult. We have a handful of sensors and cameras, the LIDAR, and vehicle state and location data to display in one page. Furthermore, we need our interface to be flexible and dynamic so that it can subscribe to all of the necessary nodes immediately without needing to be reconfigured. If we add new nodes for new sensors on the vehicle, then we want our user-interface to be able to identify and display that data with little effort.

While researching how to accomplish our user-interface, we discovered an open source web-based user-interface that was built on top of the RobotWebTools libraries. The project is called the ROS Control Center and is an AngularJS project which provides a template for how to build a web-based user-interface to visualize ROS data [?].

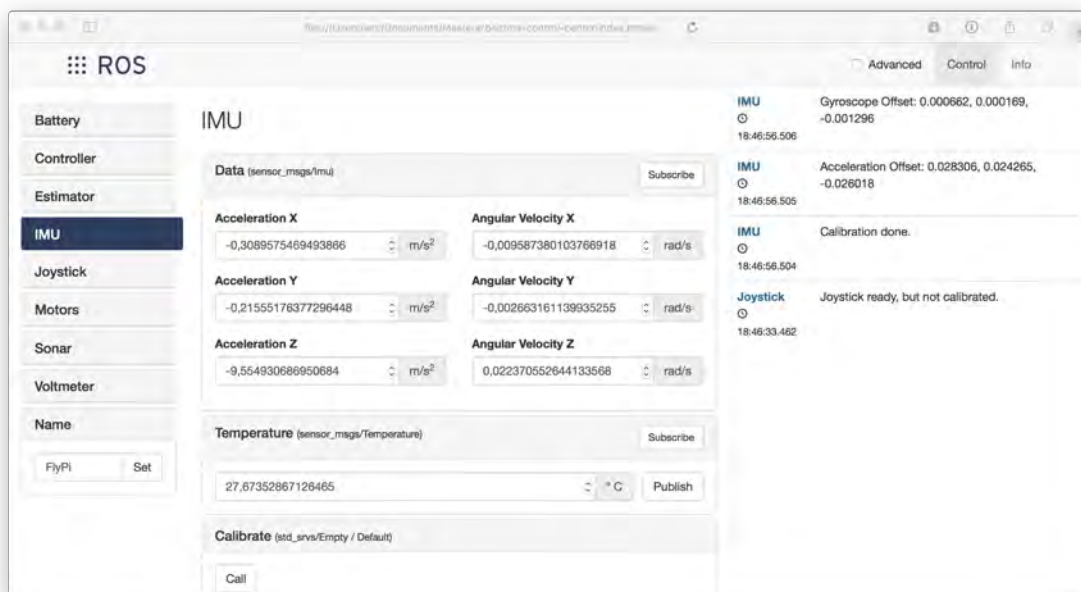


Figure 5.1: ROS Control Center

As seen in Figure 5.1, the ROS Control center comes prepackaged with visualization for a variety of default ROS packages, but these visualizations are limited to text displays of incoming data. The ROS Control Center works by asking the

RobotWebTools web server for all nodes that are currently publishing messages. These node-message type pairings create unique names as part of the ROS architecture. The ROS Control Center leverages this feature and creates an associative array where each node is a key and each value is the message type. When the ROS Control Center recognizes a node, it adds the node to the left-side navigation bar. Then, the ROS Control Center embeds the HTML code for visualizing the individual message types into the page. This way, a user can click on a particular node's name, and then see all of data for all of the messages that belong to that node. Writing these HTML files for each message is fairly simple to do because the ROS Control Center handles all of the data processing.

The ROS Control Center is not a perfect solution. It does not provide a one page view of the vehicle's most critical systems. It also does not come pre-packaged with any graphing ability. The ROS Control Center also does not provide any way to handle the idea of thresholds and warnings. Many of the environmental hazards we are trying to detect have safe and unsafe thresholds. We want our user-interface to be able to provide feedback to the user when the unsafe thresholds are exceeded to quickly alert the user that there is a problem.

These are all features that can be added to the ROS Control Center. Since each message is given it's own HTML page, we can write the JavaScript logic in those pages to handle alerts for safe and unsafe conditions and build live-stream plots in those pages too. We can also build a page that uses the data from multiple messages and displays them in one page. Adding these features on top of the ROS Control Center meets our design requirements and goals.

## 5.3 Improving the ROS Control Center

We made several changes to the ROS Control Center in order to implement the features that we wanted. The largest issue with the ROS Control Center framework

is that it doesn't provide any easy way to display data from multiple nodes in one page. In order to implement this feature we force the ROS Control Center to load a fake Dashboard topic. This topic is always the first one to load and display and subscribes to the different topics that we need it to. Currently the Dashboard topic subscribes to the three different environmental sensor packages, the LIDAR point cloud, the four cameras, and the vehicle state information. Not all of the data from each message is displayed on this page. Instead we identified what we believe to be the most critical parts of each topic.

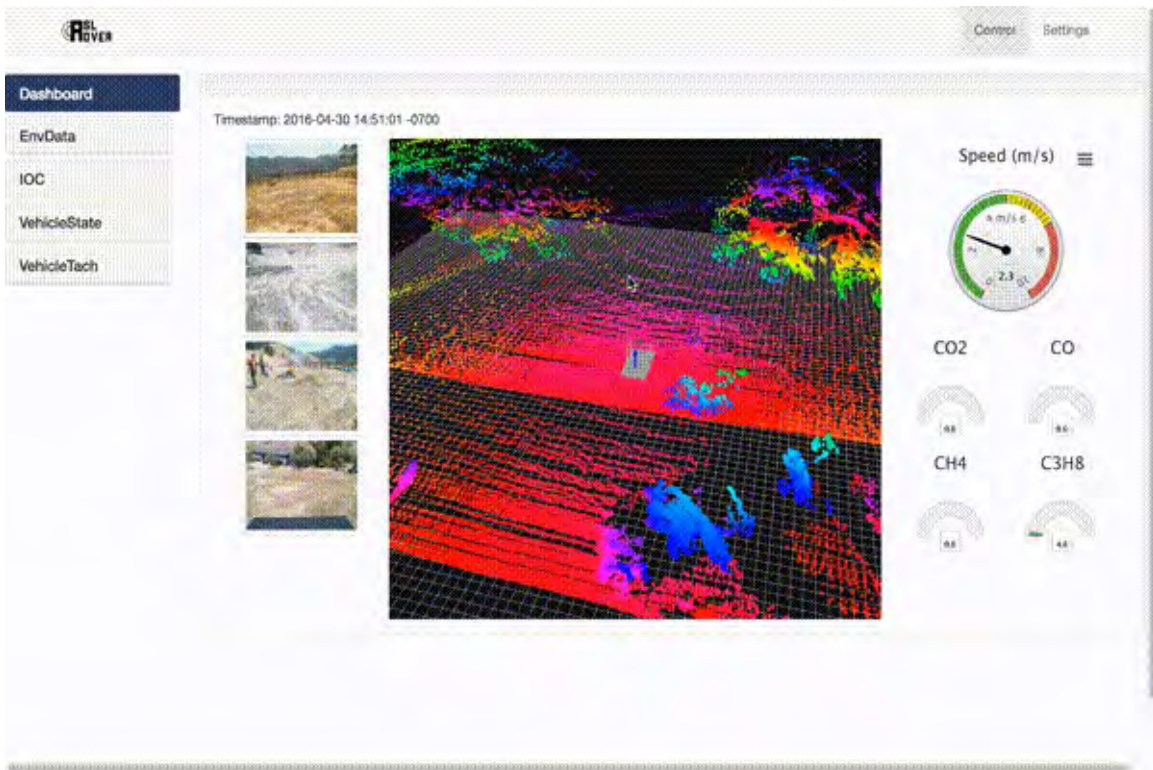


Figure 5.2: Screenshot of the User Interface during our testing session.

Figure 5.2 displays a screenshot of the this main dashboard view. On the left side of the screen are the camera feeds - front, right, left and rear. In the center of the page is the visualization of the LIDAR point cloud using the `ros3djs` library provided by the RobotWebTools. Part of that LIDAR visualization also includes rendering the model of the rover within the cloud. The reason that the point cloud

is the prominent feature on the page is that the cameras don't always provide the most reliable view of the environment. Factors such as smoke will make it difficult for an operator to drive the vehicle while using the cameras. The LIDAR point cloud however will always generate a reliable scan of the environment and give operators a clear view of where the vehicle is. The right side of the page is a combination of the vehicle state information (the speed of the vehicle) and the environmental sensors.

All of the charts are created using the highcharts-ng AngularJS library which provides a convenient way to use Highcharts in AngularJS.

### 5.3.1 Rendering the LIDAR Point Cloud

Rendering the LIDAR point cloud presented a special challenge. The data coming off of the LIDAR is simply a line of what the LIDAR is scanning. As the LIDAR moves up and down it publishes these lines, and it is then the responsibility of the visualizing software to aggregate these lines and display the resulting cloud.

There are two potential solutions to this problem:

1. Assign each point a timer, and when the timer expires, remove the point
2. Only hold so many points in the scene at once

For our implementation, we chose to only hold so many points on the screen at once. The ros3djs library allows a maximum number of points to be set for display on the screen, but it assumes that it already has a full point cloud. We added the ability to have the scene aggregate the points during render time. Essentially, we keep track of the last index that we wrote points to, and on the next message, we start writing at that location. When we reach the end of the array, we start back the beginning and overwrite old points with new information. We currently have the scene set to hold 75,000 points at once, which seems to hold about three to five seconds worth of point cloud data, which is plenty of time to visualize the surrounding environment as the vehicle moves through it.

We also needed to assign color to the points. If we didn't assign colors, the entire cloud would be gray which can makes it hard to distinguish what's in the cloud. We use a quick linear-interpolation to determine color of a given point by using its y-axis value. We chose the y-axis value because it gave us the best clarity. The colors progress from red to green to blue as you move upward in the scene. This interpolation isn't perfect though. It actually cycles through the color wheel before reaching the max y-axis value. This was done partially on purpose to provide greater clarity. Our first attempt at coloring the scene made people hard to distinguish. A standing body takes up a very small portion of y-axis values, and so they were appearing as single colored blobs. We allowed the colors to cycle in order to allow small objects to be more visible; however, we may have allowed the colors to cycle too much. Now, there are almost too many colors on the screen which can make it hard to process.

## 5.4 Network for Internet Communication

The RobotWebTools libraries and the ROS Controller Center assume that there is an internet connection between ROS and the device running the ROS Controller Center. The rover is meant to be deployed in rugged, off-road terrain where an existing internet connection is not necessarily available. To deal with this problem, we purchased a high-power router that sits on the vehicle. On the vehicle we run the RobotWebTools `rosbridge_server` package and a small HTTP web-server. The `rosbridge_server` package handles translating ROS messages to a JSON format so that the web-client can use the data. The vehicle network architecture is explained further in Figure 5.3.



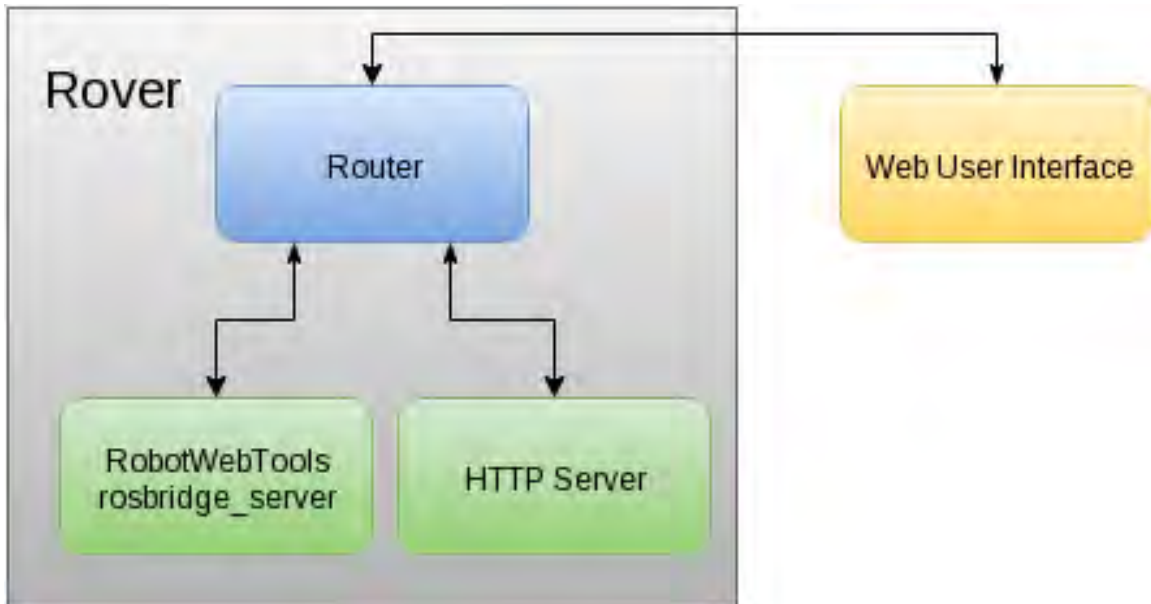


Figure 5.3: Network architecture to enable the user-interface

Ideally, we would want the rover to be able to serve up the HTML pages and JavaScript files necessary to run the UI. Unfortunately, all of the computer systems currently on the rover are already overburdened. Our current solution is to just download our modified version of the ROS Control Center on a laptop and then load the page from a browser locally. This has worked well enough for our use cases, although in the future, it would be nice to have the vehicle serve the files. This would require the least amount of setup for accessing the user interface in the field.

## 6 Subsystem: Communications

In order to meet our design objectives, our vehicle needs to be able to relay information back to its operator and receive commands from the operator. In order to achieve this, we need to design a strong communication link between the vehicle and the operator.

Our current implementation uses a peer-to-peer wifi network to receive feedback from the vehicle and an XBee PRO for vehicle driving commands. Peer-to-peer wifi networks are easy to setup and maintain and provide the high bandwidth required to transmit the camera and sensor data from the vehicle to the operators. However, the main problem with the peer-to-peer wifi network is that it has a relatively short range. The connection is not good at long distances, and the actual distance varies depending on if there are any objects in between the network nodes. For safety and reliability reasons, we chose to keep the XBee PRO communication link between the control console and rover for the driving commands. The XBee has a much lower bandwidth, but a much higher range (theoretical 28 miles, line-of-sight).

In order to meet our design requirement of 150 meters line-of-sight communication link, we chose to install a Amped Wireless High Power Wireless-N 600mW Gigabit Router (R10000G). While the theoretical range of this device is dependent on the wifi performance of the receiving device, our field tests demonstrated an actual range of over 300 meters. Photos of the XBee and router can be found in Figure 6.1.

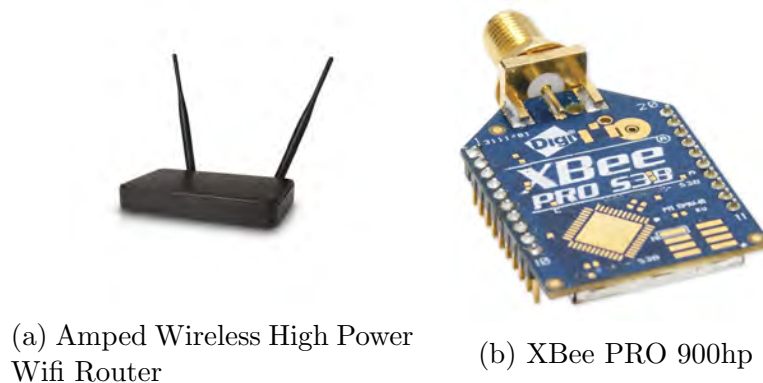


Figure 6.1: Communication Subsystem

We recognize that the shortcomings of a peer-to-peer network are unacceptable and unsafe for our application. However, due to budget constraints and the technology that we had readily available to us, we decided to use the peer-to-peer wifi network anyways. This was enough to develop a functioning prototype of the vehicle and implement all of the other functionality that we need to.

In order to create a market ready solution, future teams should look to update this subsystem and use a technology that has a greater range and reliability.

## 7 Subsystem: Power

Our vehicle houses a great deal of equipment and payloads that run off of electrical power, and it is important that we keep all of these payloads operational and functioning in an efficient manner.

We have decided to utilize three, 12V deep-cycle marine batteries. Two of these are wired in series to supply 24VDC to the actuators, LIDAR units and other electronic equipment, and one is used to provide 12V power to the engine, lights and sensing packages. This option of marine batteries provided us a space and cost efficient way to power our equipment, though different options have been considered.

We have also added a 24V DC/DC regulator to filter out electrical transients. While the actuators are still powered directly from the battery bank, the LIDAR units required a source of "clean" power. The 24V regulator which we chose can both up and down convert the voltage levels in order to provide a stable power source for the more sensitive electronics. We have successfully incorporated this change into the vehicle's power system and have power margins to support additional 5V, 12V and 24V payloads.

We considered attaching two more additional batteries in parallel to the 24 volt battery bank, but decided to postpone adding more battery capacity due to the incremental cost. However should future payloads require more power, additional batteries could be wired in parallel to provide more energy storage capacity. For more information, please reference our power budget: Appendix F.

## 8 Subsystem: Localization/Mapping

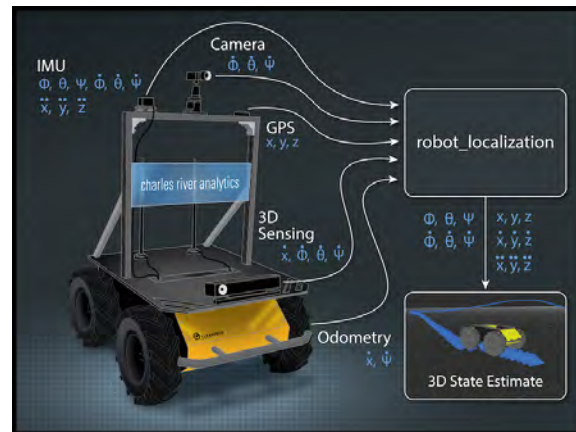


Figure 8.1: Localization Illustration [?]

Localization is an important step for any autonomous or partially autonomous system hoping to navigate in the physical world. While autonomous driving does not fall within the scope of our project, localization is a necessary step for creating an accurate map of the environment.

ROS provides a framework for developing localization systems and provides several cutting-edge algorithms for fusing sensor data into an accurate position estimate. For our vehicle we are combining the LIDAR, accelerometer, gyroscope, GPS, and wheel sensor data using an Extended Kalman filter, using ROS's robot\_localization package (Figure 8.1), to generate an accurate position estimate. Using this estimate, we then build both 2D and 3D maps of the environment using our LIDAR sensor data.

### 8.1 Sensors

The first sensor that was incorporated into our localization scheme was the vehicle tachometer (Figure 8.2). The tachometer is an aftermarket sensor addition which senses the vehicle speed, which we consider to be a "body-frame forward" speed. The tachometer is actually two adjacent hall-effect sensors which sense the movement of



Figure 8.2: Hall Effect Tachometer

the teeth on a gear fixed to the rear drive shaft. Through the use of dual hall-effect sensors, the sensor is able to output a quadrature signal from which both speed and direction can be extrapolated. The output of this sensor is fed to the 'Tachometer Arduino', which interprets the quadrature signal and feeds the resulting position and speed to the Vehicle Mega.



Figure 8.3: CH Robotics UM7 IMU

The second source of data comes the Inertial Measurement Unit (IMU), which

includes 3 sensors; a 3-axis accelerometer, a 3-axis gyro, and a 3 axis-magnetometer. We chose the CH Robotics UM7 IMU (shown in Figure 8.3) to use on the vehicle because of it's relatively low price point, existing integration with ROS and it's active user base. Most modern IMU's have filtering on the device and in the case of the UM7, it outputs an orientation estimate (yaw,pitch,roll) as well as the raw sensor data.



(a) GPS Receiver



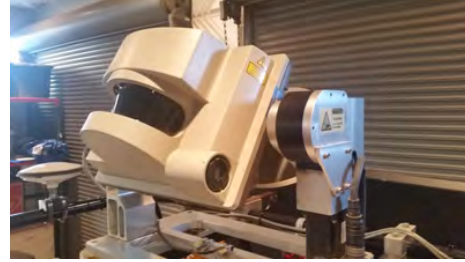
(b) GPS Antenna

Figure 8.4: Novatel ProPak-LB GPS System

Next, the GPS unit, a Novatel Propak-LB, consists of a GPS antenna (Figure 8.4b) and a separate GPS receiver (Figure 8.4a). The system outputs position and velocity estimates in the form of standard NMEA strings. Preexisting ROS nodes (`nmea_serial_driver`) enable us to read in these NMEA strings over a usb-to-serial adapter and incorporate the GPS data into our state estimate. The GPS unit has the added benefit of performing it's own estimate of the covariance (a statistical measure of the estimate's certainty). The Kalman Filter utilizes the covariance to weigh the GPS more heavily when the GPS has more satellites in view. Additionally, the GPS unit is capable of taking WAAS (Wide Area Augmentation System) GPS corrections into account. The WAAS system calculates and transmits corrections to increase the prevision of GPS receivers.



(a) Front Mounted Static LIDAR



(b) Roll Cage Mounted Sweeping LIDAR

Figure 8.5: RSL Rover LIDAR Sensors

Finally, the rover incorporates 2 LIDAR sensors; a statically mounted unit on the front grill (Figure 8.5a) and a tilting unit on a gimbal mounted on top of the vehicle (Figure 8.5b). For localization, we are using the unit mounted on the front of the vehicle, along with a ROS package called `hector_slam` to produce an estimate for the robot's state. The SLAM (Simultaneous Localization and Mapping) process compares the current laser scan to a map or past data to estimate the movement of the vehicle. The SLAM process also provides us with a 2D map of the environment which can be saved and used for navigation at a later time.

## 8.2 Coordinate Frames

When developing a complex robotic system, keeping track of the reference frames for all of the different components of the robot is critical for determining the configuration of the robot in space. ROS has convenient functionality for not only defining coordinate frames, but also for calculating the transformations between them. The coordinate frame locations are illustrated in Figure ??.

Our first frame of interest is the `base_link` frame. By convention this is the highest level frame on the robot. For our purposes we chose to define the `base_link` frame to be located at the centroid of the four rear tires with the X axis forward, Y axis left and Z axis up. The direction conventions are ROS standards, but the translational



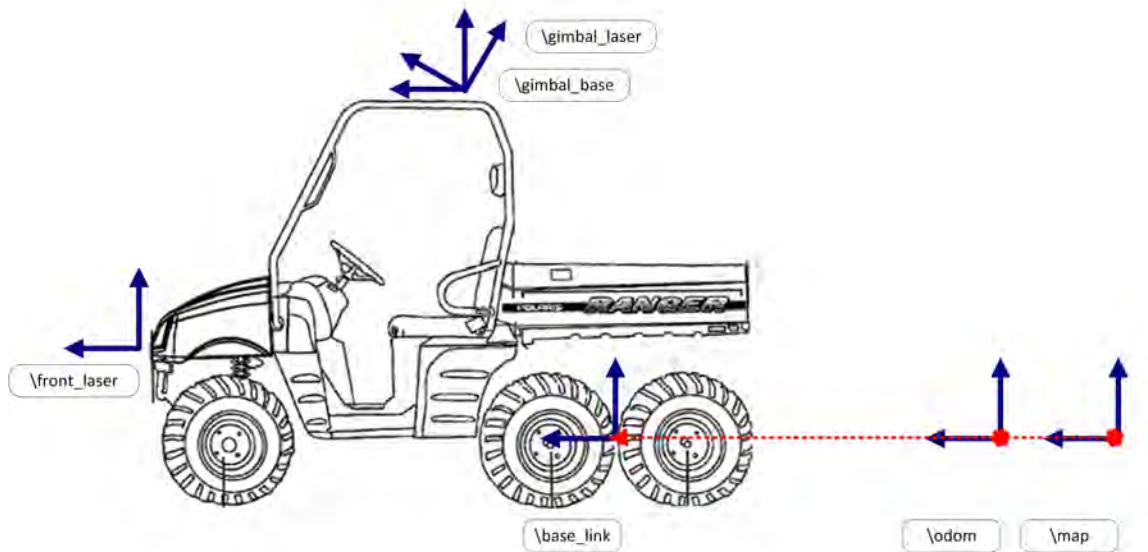


Figure 8.6: ROS Coordinate Frame Illustration

location of this frame was left to us. We chose this particular location based on the kinematics of the vehicle as the center of rotation of the vehicle should be around the centroid of the four rear wheels.

When considering the location of the rover in its environment or the "world frame" as it is often called in ROS documentation, we consider two types of estimates and therefore two frames. The first frame is the odometry frame (`\odom`) and the second is the map frame (`\map`). The odometry frame serves as an intermediate step in the estimation process for determining where the rover is located in the map frame.

The localization of the vehicle is performed in two stages, first taking into account only sensors which produce a continuous estimate of the robot state (i.e., Tachometer, IMU) for use in the future autonomous driving and local path planning. The output of the localization defines a coordinate transform between `\base_link` and the `\odom` frame.

The second step takes into account discontinuous sensor data (i.e., GPS, SLAM) to produce a more accurate, but sometimes 'jumpy' position estimate. The advantage

of using this frame is that when planning long-term navigation, this frame is not subject to the same drift that the odom frame is subject to. The second step in the localization process defines a transformation between the `\base_link` and the `\map` frames, often using the `\odom` frame as a starting point.

Several other frames exist for generating the pointcloud data. The `\front_laser` frame is located in the center of the front bumper. The `\gimbal_base` frame is located on the top of the roll-cage, at the center of rotation of the gimbal and parallel to the `\base_link` frame. Finally the `\gimbal_laser` frame is located on top of the `\gimbal_base` frame but rotates in pitch with the motion of the LIDAR.

## 8.3 Kalman Filter

We are using an implementation of an extended Kalman filter for state estimation through the ROS `robot_localization` package. This package takes input from an arbitrary number of sensors and produces an estimate of the robot's 15 dimensional state, `[x,y,z,roll,pitch,yaw,vx,vy,vz,vroll,vpitch,vyaw,ax,ay,az][?]`.

As mentioned previously, the localization takes place in two steps. The first step defines the continuous estimate in the `\odom` frame. For this step we only fuse the continuous estimates of the robot position, specifically the IMU and the tachometer. For each of these sensors, we only define a subset of their measured quantities to use. For example, with the IMU, we chose only to fuse the roll, pitch and yaw into the position estimate, excluding the raw angular rates, and accelerations, because they were already incorporated in the IMU's internal filters. Next, we chose to fuse the tachometer speed as a body-frame forward velocity for the rover. Using these two pieces of information, the IMU for the direction and the tachometer for the speed, we are able to get a rough position estimate for the rover.

However, as time increases, small bias errors or noise will accumulate to create a position estimate which is unusably incorrect. To compensate for this, we perform

a second localization step which incorporates the GPS and SLAM functionality to provide absolute positioning relative to the environment. We fuse the same variables from the IMU and tachometers but then add the X,Y position from the GPS and X,Y velocities from the SLAM node. The `\map` frame defined by this node has small discontinuous jumps, however its absolute position error does not grow continuously like the `\odom` frame.

One of the features which was most frustrating to debug with the Kalman filter was the effects of the covariance matrix for each sensor. The covariance matrix represents the statistical confidence in the sensor data being reported. Some nodes, such as the GPS node and the SLAM node report an actual, dynamic covariance based on the performance of their algorithms. However, other sensors such as the IMU and Tachometer don't report covariance. In order for the Kalman filter to output a valid estimate, we had to estimate these values. In order to do this, we viewed the steady state noise floor for each sensor and set the covariance such that it was slightly higher than this value. While not extremely accurate, this seemed to perform well and prevented the Kalman filter from propagating this noise into the position estimate.

## 8.4 Hector SLAM

In order to generate a more accurate absolute position estimate than the GPS could provide, we turned to a SLAM (Simultaneous Localization And Mapping) solution. SLAM works by analyzing LIDAR data, comparing subsequent scans to determine a position and orientation estimate for the robot. In order to decrease the implementation time, we turned to several available ROS implementations of SLAM algorithms. The most common packages for SLAM are `gmapping` and `hector_slam`. Both packages are 2D SLAM algorithms which compare planar laser scans to determine position. We chose to use `hector_slam` due to the fact that an odometry input is not necessary.

While we saw very good performance out of the SLAM system, there are several critical caveats which would make the currently available implementations not ideal for our particular application. First, the SLAM algorithms have a fixed map of a fixed size. This essentially means that the area that the vehicle can navigate is limited by the memory available on the laptop. An ideal and proven solution to this problem would be to have a rolling map, where only the map in the area surrounding the current rover persists. Secondly, through ROS, we only had access to 2D SLAM implementations, which rely on the assumption that the environment is uniform in the vertical direction. While these implementations work very well in indoor environments with many vertical features, the outdoor environment, especially the natural outdoor environment has very few purely vertical features. In order to get better performance in an unstructured 3D environment, a 3D SLAM solution would be required. While the algorithms currently available in ROS do not address these 3-dimensional problems well, future teams or researchers may choose to implement more advanced algorithms which can perform 3D slam.

## 8.5 3D Visualization

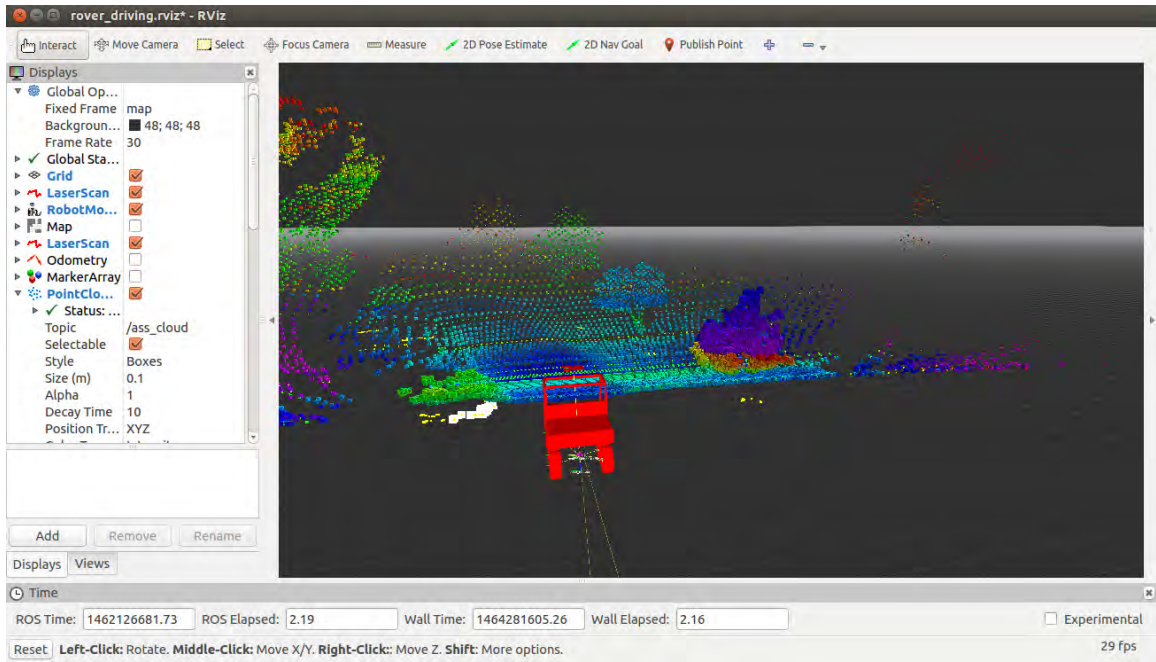


Figure 8.7: RVIZ LIDAR Pointcloud

In addition to the 3D pointcloud which is visualized in the web UI (Ch.5), ROS's rviz utility was heavily utilized to visualize pointclouds and provide debugging information to both operators and developers. Figure 8.6 shows a typical view that an operator or developer might see when using rviz to visualize pointcloud data. We chose to use the RVIZ utility over other visualization tools because of its existing integration with ROS and the ability to view all of our data, including camera streams, transformation information and raw sensor data all in one convenient location. Additionally, rviz handled the projections of the various sensor messages into the appropriate coordinate frames so that we did not have to program that from scratch. Finally, it allowed us to dynamically change what we are visualizing depending on the task at hand. Displaying all of the information being generated by the rover simultaneously would be impracticable so this functionality proved to be critical

in our development.

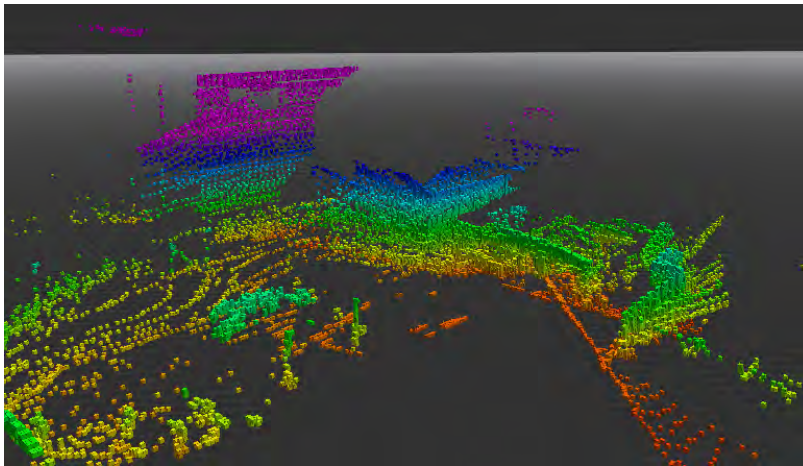


Figure 8.8: Assembled Pointcloud of Santa Clara Service Street

In post-processing we are able to feed the position estimates and LIDAR data into the Octomap ROS package. Figure 8.7 shows an example of an accumulated octomap of the service road behind Santa Clara's School of Engineering. Octomap is a 3D occupancy grid implementation which allows the LIDAR not only to identify obstacles, but also clear obstacles from the map. Octomap uses a probabilistic algorithm which means that it is able to correct mistakes or outliers when additional, contradictory sensor data is received. For example, for a case where the rover is scanning a moving obstacle, such as a person, it is able to clear the space where that the person previously occupied while simultaneously marking the place where the person currently is as 'occupied'. The dynamic abilities of this algorithm is critical for any application where a robot will be operating in a real-world environment.

## 9 Construction Plan

Our team spent the first 11 weeks of the school year planning the scope and finalizing our design and requirements for the RSL Rover: Disaster Response and Reconnaissance Vehicle. As early as December, we began purchasing the bulk of our sensors so that we could begin our next stage of prototyping as early as the first week of winter quarter. Our major milestones, including prototype deadlines and testing schedules, are outlined in the Gantt Chart in Appendix E.

Each team member took ownership of a specific subsystem or task; purchasing components and prototyping functions over winter break. In January, we began to integrate the remaining sensors required for localization, and we manufactured the housing and mountings for our environmental sensing sensors. By the end of winter quarter, we completed a first iteration of all hardware, including electronics and mounting, in order to iterate on that design in the spring. We spent spring quarter integrating our subsystems onto the vehicle and preparing for testing.

# 10 System Integration Testing and Results

In order to validate our final product, we designed a series of tests to determine to what extent we met the hard engineering requirements outlined in our engineering requirement flowdown (Appendix A).

## 10.1 Range Requirement Testing

Being able to operate the vehicle at a safe distance was a very important requirement for our customers. We wanted to verify that the remote operator console could maintain good contact with the vehicle at range and also that the router on the back of the rover gave us enough range to be able to operate the User Interface at a range as well.

We knew that the router on the back of the vehicle that powers our WiFi network that allows us to run the User Interface was going to have a shorter range than the system powering the communication with the remote operator console. We set our requirement threshold to a range that we considered feasible based on the specifications of the router. Our requirement was that the WiFi network and remote operator console have a range of at least 150 meters.

To test this, we left the vehicle and one member of our team in a stationary location and loaded the remote operator console and a laptop into another vehicle and drove as far away as possible before we lost connection. We would stop along the way, while remaining in line of sight with the vehicle, and would send commands to the vehicle via the remote operator console. If our team member with the vehicle saw the event happen, we knew the remote operator console still had contact. For the WiFi network, the web-based user interface will display a message when it loses connection, so we monitored the user interface and marked when that message appeared.

For the WiFi network, we lost connection at about 250 meters away from the vehicle. The remote operator console still had good connection, so we kept driving. We reached the edge of the property we were testing at about one kilometer away



from the vehicle and were still able to send commands. We know that the remote operator console has a range in excess of one kilometer, which greatly surpasses our 150 meter requirement.

Verification	Requirement (meters)	Result (meters)
Remote Operator Console	150	1000
WiFi Network	150	250

Table 10.1: Range requirements and verification results

Table 10.1 shows the requirements we set for our the range of our operator console and WiFi network.

## 10.2 Latency Requirement Verification

Latency is important to be able to operate the vehicle effectively. We do not want operators to be making decisions based on stale data, and if the video feeds are not coming through fast enough, it will make it very hard to drive the vehicle. Figure 10.1 shows an example of the view from the front of the vehicle, provided by our front facing camera.



Figure 10.1: Screenshot from the front facing camera on the vehicle taken during our testing

There were several areas of latency that we wanted to test. First, we wanted to test the latency from the cameras to the onboard laptop. Second, we wanted to

test the latency from the cameras to the web-based user interface and see what the difference is between the two. Finally, we wanted to check what the latency was for vehicle state data, such as wheel speed and the readings from environmental sensing packages, from the vehicle to the user interface. In all of these cases, we set the requirement that this latency be less than one second. Due to practical constraints, we conducted our latency tests near (¡10m away) the vehicle, but did not experience any significant increase in latency during our range testing. Table 10.2 shows the specific requirements we set for the latency of our cameras and vehicle state data.

Verification	Requirement (seconds)	Result (seconds)
Cameras to onboard laptop	1	.75
Cameras to user interface	1	.8
Vehicle state data to user interface	1	2

Table 10.2: Latency requirements and verification results

As shown in table 10.2, the camera feeds in both cases had a latency under our one second requirement. The vehicle state data however, had a latency of two seconds. We later discovered that this was due to lack of optimization. There were points in the data pipeline that were causing data to take longer to send and be processed than we originally thought. Displaying the data in graph form, such as the gauges for the different gas readings, added extra time to processing the data.

Future optimizations could be done to increase the speed of this processing.

### 10.3 GPS Testing

In order for our vehicle to be effective in it’s goal of providing meaningful information to fire responders, the vehicle needs to know it’s geographic location, not only for driving purposes, but also to correlate the payload (environmental sensor) data as well. In order to do this we used a high end Novatel GPS receiver and with the incorporation of WAAS (Wide Area Augmentation System) corrections we were able to get a reliable GPS fix higher than 2 meter accuracy. We attained this number by



Figure 10.2: Logged GPS tracks of various field testing activities

looking at the GPS receiver’s self-reported HDOP (Horizontal Dilution of Precision). For determining long term trends in environmental data, we consider this accuracy to be more than sufficient for our application. Figure 10.2 shows GPS driving data we collected during our testing.

## 10.4 Localization and Mapping Testing

As part of our testing process, we ran our localization and 3D visualization nodes while driving in order to confirm the functionality of our system. As shown in Figure 10.3, the resolution and accuracy of our 3D visualizations is more than sufficient for detecting driving obstacles and offers a viable alternative to visual driving (via cameras) in situations where smoke occludes the camera view.

While there were no quantitative requirements associated with the 3d visualization, we can definitively say that the 3D views, exemplified in Figure 10.3, allow fire responders not only to avoid obstacles but to inspect objects of interest in their environment, therefore validating our results as providing a solution to the fire responder’s

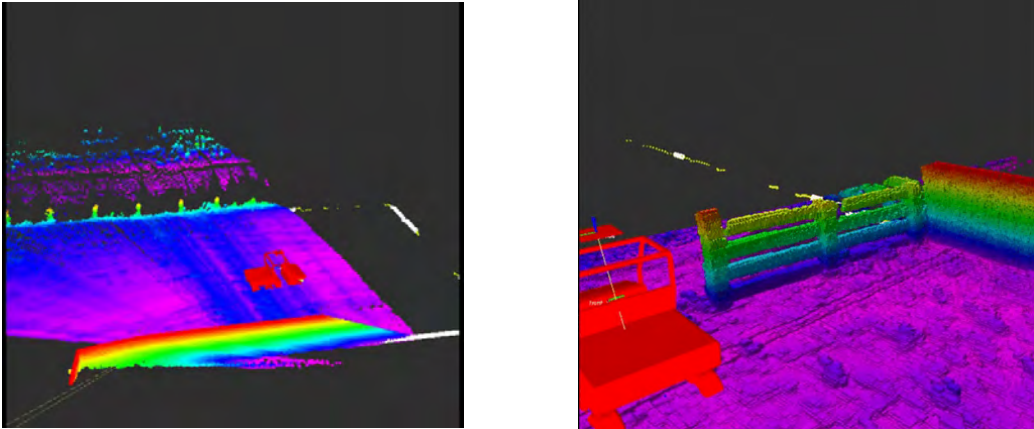


Figure 10.3: 3D visualization of vehicle during mapping activity

practical needs.

## 10.5 Environmental Sensor Package Testing

Our environmental sensing packages are a critical component of our vehicle. To test the effectiveness of these packages, we set a small, controlled burn and circled the fire with our vehicle. With each pass around the fire we increased the radius of our lap. The point of this was to test our ability to pinpoint the location of a fire using the mapping and localization system in conjunction with the environmental sensing packages.

As shown in figure 10.4 our environmental sensing units are able to detect the presence of environmental hazards. The x and y-axes are position, and the z-axis is the concentration of air particulates, which in this case is the presence of smoke. The concentration peaks at the center of the graph and decreases as position away from the center increases. These are the expected results, since the concentration of smoke should decrease as we move away from the fire. Additionally, we measured higher concentrations of smoke downwind of the fire which was expected due to the visibly higher concentration of smoke in that direction.

We can also see how the different sensor packages detect the gases differently.

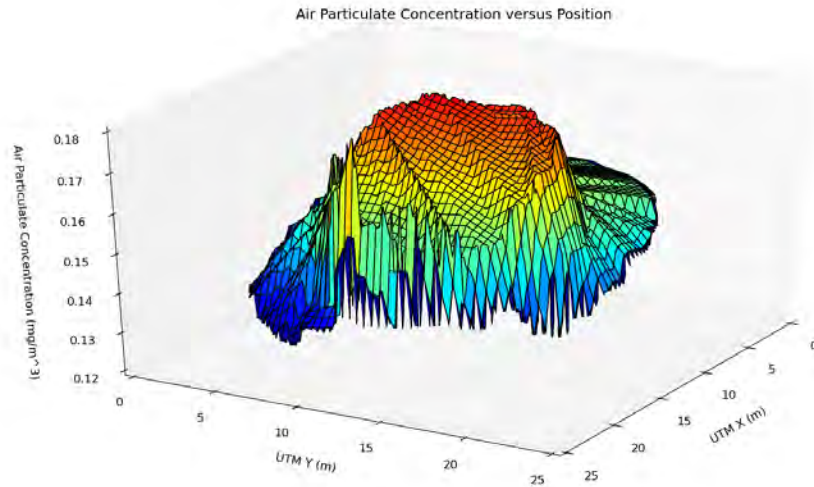


Figure 10.4: The results of our controlled burn test.

While not an actual requirement, we were curious to see how the different sensor packages functioned. The point to having redundant sensors was to be able to continue detecting environmental hazards even if one package dies. We wanted to know if the packages behave comparably to each other.

Figure 10.5 shows the measurement of LPG versus time for each of the environmental sensing units. While all of the sensing units follow a similar trend, each one displays different levels of LPG. When the test started, we were closest to the fire, which is why the LPG concentration is the higher at the beginning of the plot. Each sensor is also positioned in a different place on the vehicle, which explains some of the differences. However, all of the sensors show an increase in LPG closest to the fire, and less LPG as the vehicle moved away from it, which was what we expected.

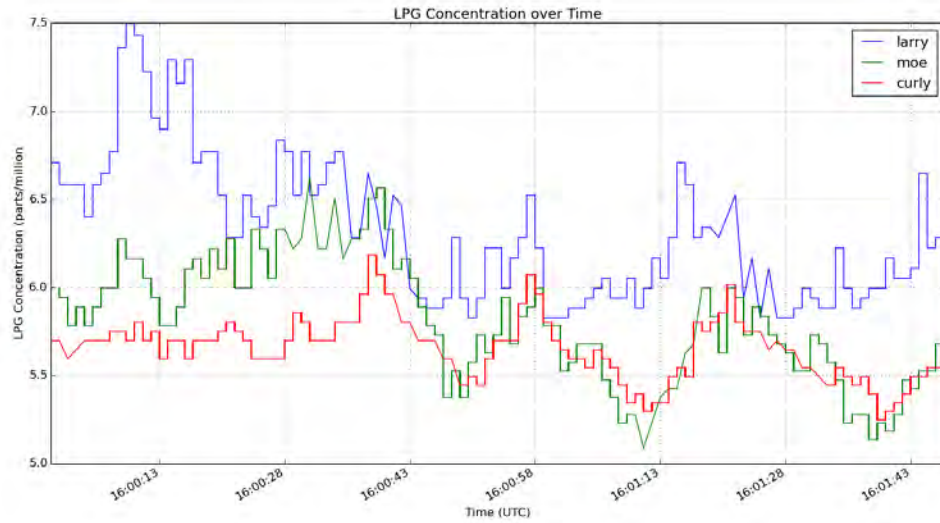


Figure 10.5: LPG versus time of three sensor packages.

## 10.6 Blind-spot Testing

To test the blind-spots on the vehicle, we attached string to the cameras and placed posts in the ground when they had just left the field of view of the cameras. The strings were all of close to equal length such that we could measure the distance between the posts in a straight line. You can see how we performed this test in Figure 10.6. We can then convert those distances into angles that define the field of view of the vehicle, and the size of the blind-spots of the vehicle.



Figure 10.6: The resulting web from attaching string to each camera and then tying those strings to posts and placing them in the ground when the post left the field of view of the respective camera.

Our requirement was that the cameras would offer 360 degrees of view. In other words, our requirement was to have no blind-spots. Unfortunately, based on the resulting calculations from the measurements we took, each camera only provided 65 degrees of coverage, resulting in a total 260 degrees of view. Each blind-spot between the cameras was only 25 degrees, but this still fell short of our requirement.

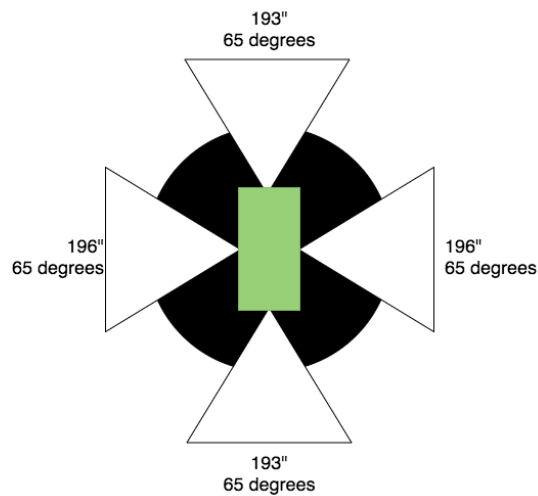


Figure 10.7: The distance between the posts and the angles we derived between them.

A diagram of the field of view is shown in Figure 10.7. We could further increase

our field of view by adding two more cameras to the vehicle, placing them on the side of the roll cage angled backwards. We would then take the two cameras already on the roll cage and angle them more forwards. These six cameras together would then cover the current blind-spots on the vehicle.



# 11 Costing Analysis

As part of the design process, a detailed, itemized budget was assembled for the purchase of the components necessary to fulfill our engineering requirements (Appendix D). The most expensive items that we purchased are the sensors, for environmental sensing and for localization. We were fortunate enough to be provided with 2 SICK laser range finders by our advisor (Dr. Christopher Kitts). These items alone would have exceeded our budget of \$2500. In addition to the sensors provided by Dr. Kitts, we calculated that the environmental sensing packages cost roughly \$200. The remainder of our funds went towards communication and interface equipment as well as towards repairing the vehicle. As the vehicle is indeed an experimental research platform and is highly modified from a stock state, we allotted a generous amount of our funding to go towards repairing, tuning and upgrading the vehicle.

The RSL Rover Vehicle is a prototype and a technology demonstrator. Our primary goal is to showcase the capabilities of a vehicle with environmental sensing capabilities, remote drive by wire functionality and data visualization packages. Both cost and time are constraints to our design. While we currently do not offer a market-ready solution, we have considered features beyond what we will be providing, which would have great value in a commercial product. For example, for a market vehicle, we would want to have the vehicle adhere to as many specifications of MIL-STD-810G, the US Military's environmental resistance testing specifications, as possible. Additionally, features such as heat shielding, LTE connectivity, advanced and high powered RF transmitters and autonomy would all be desired capabilities for a market-ready vehicle, but are beyond our budget and time line.

# 12 Commercialization Plan

## 12.1 Introduction

The RSL Rover is a robotic vehicle prototype designed to aid in the process of post fire investigation. The RSL Rover design is built around integrating individual subsystems under the industry standard Robotic Operating System (ROS) in order to create a vehicle capable of driving fire fighters to areas affected by forest fires, proceeding unmanned into said area, visually scanning the area, creating 3D visual maps using LIDAR technology and GPS location, gathering data on various dangerous gasses, the temperature, air particulates, and smoke, storing the data for future dissemination, and presenting important data live to users on an interface that is accessible and easy to understand. While this vehicle is a prototype built around the technology available to the RSL at Santa Clara University, our final design would integrate our modular environmental sensing and mapping subsystems to a modified drive-by-wire vehicle such as a Ford F-150 so that our product can be useful in a variety of disaster situations. We would seek to market our design to both government funded and privately funded forest fire fighting organizations. Though forest fire fighting services already have a process they go through during the stages of post-fire investigation, these processes are dangerous for the fire fighters who enter in to scan the area before any other kinds of ground-level assessments have been made. Though currently many fire fighting agencies are purchasing UAV's for robotic assistance in post fire investigation, these platforms have many disadvantages in comparison to the RSL Rover.

## 12.2 Goals and Objectives

Our company's main objective is to keep disaster responders out of harm's way. Responders will use our unmanned, modular vehicle to assess the environment that

has been affected by a disaster. Our vehicles are driven remotely and collect a variety of meaningful data and our user interface elegantly displays this data to the operators, who are located far from the disaster zone at a safe distance. With this data, operators can make educated decisions regarding the subsequent steps that must be taken in order to recover the affected area. Our goal is to ensure that responders are not subjecting themselves to any environmental hazards during this disaster response process.

Our vision for the commercialization of our product is to provide unmanned and modular disaster response vehicles to a variety of public and private disaster response and recovery departments. To do this, we will work closely with these departments to offer useful vehicles that can be used to assess environments that have been affected by various disasters. Our company will develop these vehicles with the proper environmental sensors and user interface for the specific needs of the customer in question. For example, our prototype was developed with fire response in mind (see Chapter 3 for a more detailed description of our prototype).

## 12.3 Description

The RSL Rover is unlike any product currently on the market. It is a modular drive-by-wire vehicle designed to serve emergency responders in all types of natural disasters and emergency situations. The drive-by-wire system allows emergency personnel to operate the vehicle at a safe distance while still being able to collect relevant ground level information about an area. The vehicle is not limited to drive-by-wire, though; it is also capable of being controlled manually so that it can be driven to the command center rather than being towed there. This also means that the vehicle will not only be limited to remote operation missions, it could also be utilized in scenarios where remote driving is not necessary.



Figure 12.1: The RSL Rover monitoring air quality around a controlled burn in California

Current modular packages that could be outfitted onto the vehicle include a smoke and fire-gas detection package useful in forest fire investigations and a flammable and toxic gas detection package for urban and suburban natural disaster response. Figure 12.1 shows one capability of the vehicle where it is monitoring air quality during a controlled burn in the central valley of California. Additionally, the vehicle is capable of generating 3-dimensional maps of an environment so that emergency personnel can have the most information to make informed decisions on response plans. The platform for a versatile, multi-function vehicle is there with the RSL Rover; it is up to fire departments and emergency services to discover the many functions the RSL Rover has to offer.

## 12.4 Potential Markets

The RSL Rover is meant to be marketed towards forest fire fighting agencies. These agencies range from the United States Forest Service, which currently has 750 locations, hires over 30,000 employees, of which a third are fire fighters, and has an

annual budget of over \$5.5 billion, to private forestry services, such as the National Wildfire Suppression Association who represent over 150 private wildfire services that contract out work on an as-needed basis to fight fires. Both of these styles of forest fire fighting rely on the bravery of fire fighters to analyze areas affected by forest fires, and thus our product could provide the capabilities to remove people from harms way during the period of post fire investigation.

We were lucky enough to talk to Lieutenant Firefighter Perez from the Santa Clara Fire Department, who told us that his department is given an annual budget of around one-million dollars for new equipment. While budgets vary from area to area, it can be assumed that fire fighting agencies are given high amounts of money for the purchasing of equipment that could be spent on our vehicle. As many fire fighting agencies are considering drone technologies for fire reconnaissance, we believe that this market is one that we can tap into, as our vehicle not only provides reconnaissance, but also many levels of useful data to fire fighters.

## 12.5 Competition

Many all-terrain, unmanned vehicles on the market are for military contracts exclusively. However, Northrop Grumman and Argo Robotics both have products that fit the category of an unmanned, all-terrain vehicle that can be equipped with sensor packages for non-military purposes. Some of the specs for these products can be found in Table 2.1. These are the Argo J5 Mobility Platform at thirty-nine-thousand dollars, the Northrop Grumman Andros F6, and the Northrop Grumman Wheelbarrow Mk9. Prices for the two later vehicles are not available without a direct quote.

Product	Height(m)	Width(m)	Length(m)	Sensor Packages	Features
J5 Mobility Platform	0.83	1.38	1.52	1 Camera Mount	Amphibious
Andros F6	1.486	0.445	1.32	5 Cameras	Extendable Arm
Wheelbarrow Mk9	1.24	0.63	1.24	Up to 10 Cameras	Extendable Arm

Table 12.1: Current Product Comparison



Figure 12.2: Argo J5 Mobility Platform



Figure 12.3: Northrop Grumman's Andros F6



Figure 12.4: Northrop Grumman's Remotec Wheelbarrow Mk9

While these products that are on the market are the most similar to the product we wish to produce, the issues we wish to address are not solved by these rovers. Both of Northrop Grumman's rovers do not have the housing needed for sensor packages necessary to respond to forest fires. The Argo rover also does not house the necessary data packages, and while it is amphibious, we have concluded through our research and customer interactions that the sensor housing is more important.

Other Aerial Drones are also in the market for unmanned vehicles capable of relaying data such as thermal imaging and video back to the user. Two such products are the Elimco UAV-E300 and the Sensefly's EBee. Both of these products have the capabilities necessary to map and relay usable data, and are marketed as disaster response vehicles. However, there are some inherent weaknesses to their aerial nature. Images provided by Aerial vehicles can oftentimes be unusable as smoke can sit on top of the canopy of a forest, especially in the mornings. They also can only provide overhead data, and may not be able to map certain elements disaster responders could need, such as air quality at ground level or mapping paths that are safe. Our vehicle seeks to get under this layer of smoke to provide data from the ground level.



Figure 12.5: Elimco UAV-E300



Figure 12.6: Sensefly's UAV EBee

## 12.6 Sales and Marketing Strategy

In order to market and sell our product, we would need to target national and international disaster response departments and make our product visible within the department communities.

To make our product visible and generate interest, we must get communities involved in the effort to keep disaster responders out of harm's way. We can create a series of disaster response community training efforts. For example, we could offer members of a community courses where they can learn what to do in the event of an earthquake or fire where we would also educate them on the sort of dangers disaster responders face in their daily lives. We can also make efforts to establish a relationship with local media outlets. For example, local news channels can highlight



our product during a segment on innovation in robotics or on disaster response education. Through strategies such as these, we will be able to market our product by establishing a presence within the communities our product can help in the event of a disaster. This presence will encourage disaster departments to adopt our technology.

When advertising our product, our company's main goal is to emphasize how these vehicles can save the lives of the brave disaster responders who serve to protect our communities in the event of an earthquake, fire, etc. Our vehicles allow the responders to assess the environment that has been affected by a disaster from a safe distance. With our product, these responders will not have to subject themselves to dangers such as unstable structures or toxic gases in the air. Our company should convey this message clearly through our advertising efforts so that potential customers can immediately recognize the great value of our product.

In order to sell our vehicles, we would have to employ company representatives in multiple states and countries. These representatives will meet with the department leads to determine what their specific department needs are and whether our company can meet those needs. If an agreement is made and the department purchases one or more vehicle(s), the vehicle will be hauled from one of our factories to the department location. A minimal amount of training for these disaster responders would be included. The regional/statewide/countrywide company representative will also be responsible for holding a training workshop for the responders.

## **12.7 Manufacturing Plan**

Manufacturing facilities have been minimized by using the existing platform of Ford F-150 pickup trucks as the base for the RSL Rover. The trucks will be outfitted in our warehouse by trained technicians. Manufacturing the RSL Rover at the same location where it is designed will allow the engineers to have the oversight necessary to ensure manufacturing goes according to plan. It also makes it possible to quickly

implement design changes as needed. This will require the lease of a warehouse facility with welding equipment, 3-phase power, hydraulic hoists, and a large footprint. The initial process of training technicians and outfitting the warehouse will be the most time consuming part of the process. Once the initial setup is complete, the vehicles will take approximately 200 hours to manufacture and test. This initial phase will be costly and likely require one million dollars, including the upfront research and development costs.

To offset these large costs, the production goal will be 100 to 200 vehicles per year. According to the cost estimates outlined in the following section, this will net five million to ten million dollars in revenue annually. As demand increases, more warehouses will be opened throughout the nation and potentially globally if the global market is receptive to the vehicle.

Manufacturing will take place in four phases. The first phase of the process will be outfitting the vehicle with the drive-by-wire system that is the heart of the vehicle's functionality. The importance of this system is reinforced by the next phase; phase two of manufacturing is the rigorous testing of the drive-by-wire system. After the drive-by-wire system is installed and tested, the Lidar and cameras will be outfitted to the vehicle to provide vision for the operators and analysts. The final stage of production is outfitting the vehicle with the selected modular sensing packages. There are multiple packages to choose from that allow functionality for different types of disasters.

## **12.8 Product Cost and Price**

The vehicle will be built upon the existing platform of a Ford F-150 truck in an effort to minimize construction costs and maximize utility for fire fighters. The breakdowns for manufacturing costs are listed in tables 12.2 and 12.3 and the cost of producing a single vehicle at any volume is listed in table 12.4.

<b>Component</b>	<b>Price</b>
Drive-By-Wire Outfitting	\$5,000
F-150 vehicle platform	\$31,000
Modular Sensing Package(s)	\$1,000 ea
Velodyne Puck Lidar	\$7,999
<b>Total Component Cost</b>	<b>\$44,999+</b>

Table 12.2: Component cost breakdown

Labor Cost/hr	\$20
Labor Hours/vehicle	250
<b>Labor Cost per Vehicle</b>	<b>\$5,000</b>

Table 12.3: Labor breakdown

Component Cost per Vehicle	\$44,999
Labor Cost per Vehicle	\$5,000
<b>Cost per Vehicle</b>	<b>\$49,999</b>

Table 12.4: Cost per Vehicle

The vehicle will be priced at \$100,000 base with additional modular packages available for purchase and integration. The profits associated with the sale will fund the development costs of the vehicle as well as future research and development of modular accessories.

There are no direct competitors to the RSL Rover but the available budgets of departments should be considered. A Typical, mid-sized department has a million dollar annual budget for equipment. This means that it would be relatively easy for fire departments to add RSL Rovers to their fleets. Even small departments will be able to afford them and large departments will be able to purchase several per year. According to the United States Fire Administration, there are over 25,000 registered fire departments in the U.S. Therefore, it is reasonable to predict 100 vehicle sales annually. This would allow for significant reasearch efforts and cover the initial

development costs in just one year.

## **12.9 Service and Warranties**

Since we plan on using existing vehicles (such as a Ford F-150) as a platform to apply our technology to, warranties must be handled solely by our company. We would be modifying the existing vehicle to a degree that would potentially void the car warranty. This potential loss of warranty is caused by the chance that any modifications we make to the vehicle can be the reason why the vehicle fails. The vehicle mechanical components should last roughly as long as the vehicle manufacturer claims they should last. However, we cannot predict the type of wear and tear the vehicle will experience with the subsystems we introduce.

If our product were to fail, it would be the company's responsibility to diagnose the issue and replace the necessary parts. Since our product includes complicated electrical subsystems and modified driving components we cannot expect on-site or local mechanics to know how to fix them. In some cases, like in highly populated cities, disaster response departments have their own mechanic. Or, in smaller cities and towns, these departments have local shops that they turn to when their equipment requires maintenance. Our company could introduce a special contract with automotive parts retailers as well as automotive repair shops in order to quickly and locally rectify the vehicle's small mechanical issues for the departments. For the more complicated subsystems and driving components that we introduce, we would need to have a series of specially trained technicians that can travel to the locations of disaster departments and repair the vehicles. We could require a yearly maintenance fee from the disaster departments in order to cover these costs.

## 12.10 Financial Plan and ROI

As was previously mentioned in our Cost and Price section, our vehicles when manufactured should have a cost of around \$49,999, and we feel that due to the capability of our vehicle to protect human lives, the research and development of robotic solutions to the problem, as well as the amount of skilled labor being put into combining the multiple subsystems into a usable and applicable device our price is set at \$100,000. In order to estimate overhead, our team looked for warehouse space to establish as a workshop through LoopNet, a website dedicated to renting commercial real estate. We established a relative average of around \$15 per square foot per year. Thinking that we would need at least 500 square feet for production of 20 or less vehicles in a year, while moving up 1000 square feet for production of 50 or more vehicles. Our plan would be to sell 5 vehicles in our first year to initial buyers, then expanding our market every year based on our profits we acquire. We also had an extra \$2000 in overhead for the first year in case of extra expenses. In order to cover the costs of the first year, we would need an investment of around \$30,000 (or perhaps a loan if we are desperate), followed by continuing to find investors for the future two years. Our goal is then to work up to selling 100 units per year. The Return on investment and Financial plan is shown in the figure below.

RSL Rover Business Plan - Business Plan A	FY1	FY2	FY3	FY4	FY5	FY6
Vehicles Sold	5	12	20	50	100	100
Cost per Vehicles	\$49,999.00	\$49,999.00	\$49,999.00	\$49,999.00	\$49,999.00	\$49,999.00
Overhead	\$10,000	\$8,000	\$8,000	\$15,000	\$15,000	\$15,000
Total Cost	\$259,995.00	\$607,988.00	\$1,007,980.00	\$2,514,950.00	\$5,014,900.00	\$5,014,900.00
Price per Vehicle	\$100,000	\$100,000	\$100,000	\$100,000	\$100,000	\$100,000
Revenue	\$500,000	\$1,200,000	\$2,000,000	\$5,000,000	\$10,000,000	\$10,000,000
Grant Money	\$2,500	\$0	\$0	\$0	\$0	\$0
Investment/Loan	\$300,000	\$500,000	\$500,000	\$0	\$0	\$0
Annual Profit	\$240,005.00	\$592,012.00	\$992,020.00	\$2,485,050.00	\$4,985,100.00	\$4,985,100.00
Total Net Cash	\$242,505.00	\$1,334,517.00	\$2,826,537.00	\$5,311,587.00	\$10,296,687.00	\$15,281,787.00

Figure 12.7: Excel table showing Team RSL Rover's estimated financial plan and return of investment

# 13 Engineering Standards and Realistic Constraints

## 13.1 Ethics

Any ethical concerns are very important to our design of this vehicle. We aim to potentially save the lives of fire responders. This can only be achieved if our system displays true sensor data to the best of our abilities. If our vehicle provides false values, that could mean the environment is inaccurately assessed and people could potentially perish. There might be some legal repercussions to this. We had to ensure that our readings of environmental factors like air quality and temperature are accurate. It may not completely be our responsibility since the sensors are to be purchased from other vendors (we are not making the sensors ourselves), but we must be sure to handle these sensors properly and provide the most accurate data to our ability.

To ensure that our readings are correct, we equipped the vehicle with three redundant sensor packages. This was done to ensure that any anomalous readings can be verified or disregarded, according to what the other sensor packages show.

The vehicle itself, as a tool, is not unethical. Since its application is so specific and it is clear that the vehicle is to be used in a post-fire situation, we do not foresee any scenario in which our vehicle can be used unethically. We must be “good” engineers in the sense that we had to be honest about our testing, results, and errors so that any future developers may not assume that all systems are working properly if they, in reality, are not.

## 13.2 Health and Safety

Our vehicle is housed in a small garage, where it sits on top of a handful of jack stands. We have a formal safety document that has been reviewed by certain Santa

Clara University employees that deal with safety concerns. The document contains certain rules for us engineers to follow. For example:

- make sure that the garage is properly ventilated
- have a "kill-switch" functionality for the vehicle at all times
- take special precaution when working underneath the vehicle or in any internal electrical wiring
- there should always be at least two people present in the garage when working on the vehicle.

We have explicit guidelines and our own intuition and common sense to follow. If we work on our vehicle in accordance with the safety documentation, it should be safe to operate by other users. All design decisions were made with the existing safety documentation in mind. For example, the vehicle will contain two kill-switches (one in the front and one in the back) and the remote controls will also contain a kill-switch.

The actual operators of this vehicle will also have a document that outlines all safety concerns and rules associated with operation and maintenance of this vehicle.

### **13.3 Manufacturability**

The previous electrical work done to the vehicle is not easy to reproduce. This electrical work supports the "unmanned" feature of this vehicle. It was the 2014 RSL Rover team that created the "drive-by-wire" system. [?]

This year, with ease of production in mind, we have decided to make all sensor packages as modular as possible. We wanted the environmental sensors to be easily incorporated into any vehicle. We have kept the environmental sensor processors and other components completely separate from the subsystem that drives the vehicle. So, ideally, the disaster response aspect of this vehicle's design can be simple to reproduce onto already manufactured unmanned (or even autonomous) vehicles.

We have also decided to integrate a robotic operating system (ROS) that is considered to be the industry standard at this time. That way, any perfective or corrective maintenance on the system software can be done with ease by a team of competent programmers.

## 13.4 Environment

Environmental impact was not much of a relevant concern for our design. Since the vehicle was inherited, we must work with what is available to us. The Polaris 6x6 Ranger that we are using can undergo inspection by a professional mechanic who can determine whether the vehicle emits the appropriate amount of exhaust into the environment. However, our fire response vehicle design was not affected by environmental concerns, since none of the features we will be incorporating will have a positive or adverse affect on the environment.

## 13.5 Society

The motivation behind our project as a whole is to benefit society by keeping fire responders out of harm's way. We recognized an issue in society (that fire responders are exposed to deadly conditions while investigating a fire) and we aimed to fix this issue. We considered societal repercussions while making our design decisions, especially the designs that pertain to the "unmanned" feature of our vehicle. We recognize that unmanned and autonomous vehicles are slowly making their way into daily life. [?] We also recognize that this transition is met with great opposition. With any new technological advancements that transform activities that are heavily integrated into daily life (such as driving), many members of society react with fear. This is why we have decided to keep the driving functionality as transparent as possible. This means that we did not abstract away any driving mechanisms, such



as the gear stick. When the operator changes gears, the actual gear stick moves accordingly. Keeping the driving functionality as familiar as possible will increase our vehicle's chances of being embraced by society.

## 14 Summary and Conclusions

Our senior design team sought to design and implement an unmanned vehicle that gathers and relays information about a post-fire environment back to its operator so that humans do not have to subject themselves to those hazardous environments. Our team identified five key subsystem that, when combined, create a fully functioning vehicle fit to meet our customer needs. These subsystems are environment sensing, user interface, communications, power, and localization.

Overall, our design solves the problem by providing the necessary tools to give operators the ability to investigate a post-fire environment without placing themselves in immediate danger. The sensor packages, including the air particulate sensors, gas sensors and cameras, provide data on environmental hazards. Our user-interface makes it easy for operators to both drive the vehicle and view the incoming data from these sensor packages so that operators can quickly identify potential hazards.

We tested the various subsystems in a series of field tests to verify the success of the project. The results showed that the vehicle is capable of being driven from a kilometer away which exceeded the 100 meter goal. The GPS mapping capabilities were tested throughout the days and then analyzed showing successful tracking when overlaid on a map. Air quality assessment verification was done with a controlled burn and concentrated propane, confirming the proper operation of the air quality assessment subsystem.

One area in which our design is lacking is in the communications subsystem. Due to budget constraints, we are relying on a short range peer-to-peer wifi connection to communicate with the vehicle. This works for prototyping the functions of the vehicle, but future teams should consider upgrading the communication subsystem to something that has a longer ranger and greater reliability.

Furthermore, while not included in the scope of our project, future teams could make use of the localization subsystem and the Robotic Operating System (ROS) to develop autonomous functionality to run on the vehicle. All of our design decisions

tried to keep this ultimate goal of full vehicle autonomy in mind.

We are now the third team to have worked on this vehicle and we hope that future teams will be able to reuse and improve upon our design, similarly to how we built upon the work of previous teams. We envision the next phase of work including developing semi-autonomous functionality, identifying obstacles and points of interest automatically and alerting the operators. Over the next few years, future senior design teams could feasibly implement autonomous functionality, incorporating path planning and navigation stacks to allow the vehicle to drive simple trajectories on it's own. Finally, we hope that the vehicle will be developed to the point where it can independently serve exploratory functions, navigating complex environments autonomously, even in communication-denied environments.

# Appendix A: Design Requirement Flow-down

Mission Statement: To create a teleoperable all terrain vehicle capable of transmitting air quality, temperature, and LIDAR data to a remote operator to monitor environmental conditions after a wildland fire.		T/DR	Evaluation
<b>MO-1</b>	<b>Driver can safely operate the vehicle out of harms way</b>		
MO-1-1	Teleoperable at a distance of 300ft	T	Conduct driving maneuvers from a distance > 300 feet
MO-1-2	Provide 360 degree camera view of the vehicle for driving	T	Conduct blind spot analysis of the lidar and cameras
MO-1-3	Video latency of <0.5s	T	Measure time for an event to be viewed on camera
MO-1-4	Real-time web interface for vehicle health information	DR	Subjective
MO-1-5	Control response <1s	T	Measure time between command and vehicle response
MO-1-6	Emergency stop systems	DR	Test all emergency stop buttons on the vehicle and software emergency stops
<b>MO-2</b>	<b>Dual Purpose Vehicle (Locally and Remotely Drivable)</b>		
MO-2-1	Carry 2 passengers and 200 lbs. of gear	DR	Test driving with multiple people and gear
MO-2-2	Manually drivable (without remote controller)	T	Test driving with manual override features
MO-2-3	Off Road Capable	DR	Evaluate off-road driving performance during field testing
<b>MO-3</b>	<b>Map the environment in real-time</b>		
MO-3-1	Statically scan the environment in 3 dimensions	T	Generate static point cloud images during field testing
MO-3-2	Provide 360 degree mapping capability	T	Conduct blind spot analysis of the lidar and cameras
MO-3-3	Ability to save maps for later analysis	DR	Save maps generated during field testing
MO-3-4	Generate 2d map in real-time	T	Field testing
<b>MO-4</b>	<b>Evaluate the hazards to humans in an area affected by forest fires</b>		
MO-4-1	Detect smoke concentrations	DR	Subject vehicle to smoke (at a safe distance) and evaluate expected response
MO-4-2	Detect levels of Natural Gas, LPG, CO, CO2, H2	DR	Same as above, testable for all gases but equipment not available/ too expensive
MO-4-3	Compare concentrations to LEL (lower explosive limit)	T	Compare to measured quantities to safety thresholds during field testing
MO-4-4	Sense temperature and humidity	DR	Compare measurements to known weather measurements
MO-4-5	Redundant sensing packages	DR	Compare output of 3 units
MO-4-6	Stream data in realtime to remote operators	DR	Design Req
MO-4-7	Latency < 2s	T	Measure time between stimulation and UI response
MO-4-8	Enclosure designed to accurately sample the environment	DR	Evaluate subjectively during field testing
<b>MO-5</b>	<b>Provide an effective user interface</b>		
MO-5-1	Display all information relevant to driving the vehicle on a single page	DR	Design Req
MO-5-2	Display environmental data concisely	DR	Design Req
MO-5-3	"Drill down" style to obtain more in-depth information	DR	Design Req
MO-5-4	Latency < 2s on non-critical data	T	Measure time between stimulation and UI response

Figure A.1: Requirements Flowdown

Eval ID	Evaluation Technique	Location/Time	Equipment	Accuracy	Trials	Expected Outcome	Formula / Assumptions	Man Hrs
MO-1-1	Conduct driving maneuvers from a distance > 300 feet	Ranch / Apr-30	Measuring Tape	3 ft	15 min	500 ft	Line of Sight	2.5
MO-1-2	Conduct blind spot analysis of the LIDAR and cameras	SCU / Apr-20	Measuring Tape, Poles, String	10 deg	3	300 deg	-	5
MO-1-3	Measure time for an event to be viewed on camera	SCU / Apr-20	Stopwatch	0.2 s	10	0.5 s	Within 300 ft	3
MO-1-4	Subjective	SCU / May-2	Computer	N/A	Continuous	-	-	-
MO-1-5	Measure time between command and vehicle response	Ranch / Apr-30	Stopwatch	0.2 s	10	0.5 s	Within 1000ft	5
MO-1-6	Test all emergency stop buttons on the vehicle and software emergency stops	Completed	N/A	N/A	3	Full Stop	-	5
<b>MO-2</b>								
MO-2-1	Test driving with multiple people and gear	Ranch / Apr-30	Hay Bale	20 lbs	1	No failure	Distributed Load	4
MO-2-2	Test driving with manual override features	Completed	N/A	N/A	15 min	No issues	Slow Speeds	2.5
MO-2-3	Evaluate off-road driving performance during field testing	Ranch / Apr-30	N/A	N/A	30 min	No issues	Slow Speeds	5
<b>MO-3</b>								
MO-3-1	Generate static point cloud images during field testing	Completed	Onboard Systems	1 ft	5	Decipherable graphic	environment	3
MO-3-2	Conduct blind spot analysis of the LIDAR and cameras	SCU / Apr-20	Onboard Systems	10 deg	3	350 deg	-	4
MO-3-3	Save maps generated during field testing	Completed	Onboard Systems	N/A	3	Decipherable graphic	environment	3
MO-3-4	Field testing, measure latency	Completed	Onboard Systems, stopwatch	1 ft, 0.2 s	15 min	Obstacle identification	Static environment	3
<b>MO-4</b>								
MO-4-1	Subject vehicle to smoke (at a safe distance) and evaluate expected response	Ranch / Apr-30	Controlled Fire / Test Gas	0.1 ppm	5	0.1 ppm	Homogenous smoke field	5
MO-4-2	Same as above, testable for all gasses but equipment not available/Too expensive	Ranch / Apr-30	Controlled Fire / Test Gas	50 ppm	5	50 ppm	Homogenous environment	5
MO-4-3	Compare to measured quantities to safety thresholds during field testing	Ranch / Apr-30	Controlled Fire / Test Gas	50 ppm	5	50 ppm	Homogenous environment	5
MO-4-4	Compare measurements to known weather measurements	SCU / Apr-15	Handheld thermometer/ Hygrometer	3 C, 5%	5	1 C, 5%	Accurate comparators	5
MO-4-5	Compare output of 3 units	Ranch / Apr-30	N/A	N/A	5	Consistent info within	Homogenous environment	5
MO-4-6	Design Req	SCU / May-2	Computer	N/A	Continuous	10%	-	-
MO-4-7	Measure time between stimulation and UI response	SCU / May-2	Stopwatch	0.2 s	10	2 s	Within 300 ft	3
MO-4-8	Evaluate subjectively during field testing	Ranch / Apr-30	N/A	N/A	5	No issues	Homogenous environment	3
<b>MO-5</b>								
MO-5-1	Design Req	SCU / May-2	Computer	N/A	Continuous	Requirement Met	Subjective, large display	3
MO-5-2	Design Req	SCU / May-2	Computer	N/A	Continuous	Requirement Met	Subjective, large display	3
MO-5-3	Design Req	SCU / May-2	Computer	N/A	Continuous	Requirement Met	Subjective, large display	3
MO-5-4	Measure time between stimulation and UI response	SCU / May-2	Computer / Stopwatch	0.2 s	10	2 s	Within 300 ft	3

Figure A.2: Test Plan

# Appendix B: Market Survey

Need Category	Need	Description	Priority	Justification
Communication	Strong Communication Link	Vehicle must have a strong communication link both to the operators as well as to mobile command station	LOW	Communication links are expensive COTS solutions, our senior design is a proof of concept, not a marketable product.
	Clear Effective Data Presentation	Sensor data must be presented in such a way that the first responder can quickly and easily extract useful information	HIGH	Information presentation is key functionality which is necessary in a proof of concept
Sensors	Imaging Sensors	An array of thermal imaging cameras, LIDAR, and video feeds that work in many visibility conditions	HIGH	The vehicle awareness from the cameras is key to the proof of concept vehicle.
	Combustible Gas Indicator (CGI)	First responders need to be aware of any potentially explosive gasses present at a disaster site	MEDIUM	We can include inexpensive sensors to search for some gasses but more comprehensive sensors are outside of our price range
	Air Particulate (Smoke)	Accurate smoke levels provide first responders information about the survivability of an area for unprotected individuals	HIGH	Air particulate sensors are within our price range and provide valuable information
	Temperature	Temperature sensors provide information about danger levels especially in forest fire fighting operations	HIGH	Sensor is readily available and provides useful information to the first responder
	Geiger Counter	The geiger counter would provide useful information but only in specific disaster scenarios	MEDIUM	While the data would be interesting, the sensor is expensive it is not necessary for a proof of concept.
Vehicle	Off-Road Capable	The vehicle must be able to reasonably traverse obstacles while remaining nimble	HIGH	The RSL Rover platform is already tailored for off-road applications
	Weather/Temperature Resistant	This is especially important for fire fighting operations. Heat shielding would be invaluable for these instances	LOW	Heat shielding would be very expensive and can be added for a post-prototype vehicle
	High Angle of Attack	The vehicle needs to be able to climb hills and over debris	MEDIUM	The RSL Rover vehicle platform already has this capability
	Long Range	The range of the vehicle directly impacts the type of operations that it can participate in.	HIGH	The RSL Rover vehicle was designed with an extended gas tank (greater than 20 gal)
	Power Supplies	The vehicle must be able to incorporate application specific sensors	HIGH	We can provide excess power and communication facilities for extra sensors to be integrated

Table B.1: Customer Needs

# Appendix C: Tradeoff Analysis

Prioritizing Matrix

Criteria	1	2	3	4	5	6	7	8	9	10	11	12	SUM	FACTOR
1. Implementation Time	1	0	0.5	1	0	1	0.5	1	0.5	1	1	1	4.5	4.5
2. Forward FOV	0.5	1	1	1	0.5	1	1	0.5	1	1	1	1	7	7
3. Reverse FOV	0	0	0	0	0	0	0	0	0	0	0	0	0	4.5
4. Side FOV	1	0.5	1	1	1	0.5	0	1	1	1	1	1	1.5	1.5
5. Blind Spots	0	0	0	0	0	0	0	0	0	0	0	0	0	7.5
6. Weight	0	0	0	0.5	0	0	0	0	0	0	0	0	0.5	0.5
7. Material Cost	0.5	0	0	0	0	0	0	0	0	0	0	0	0	4.5
8. Suspense Height	0.5	0.5	0	0	0	0	0	0	0.5	0.5	0.5	0.5	4.5	4.5
9. Vertical Clearance	1	1	1	1	1	1	1	1	1	1	1	1	1.5	1.5
10.	1	1	1	1	1	1	1	1	1	1	1	1	9	9
11.	1	1	1	1	1	1	1	1	1	1	1	1	10	10
12.	1	1	1	1	1	1	1	1	1	1	1	1	11	11

Project: RSL Rover  
 System: Lidar Configuration  
 Date: 11/10/2015

11/14/2015

Figure C.1: LIDAR Configuration Tradeoff Analysis Priority Matrix

Design Project = RSL Rover		System = Lidar Configuration		DESIGN IDEAS											
CRITERIA	TARGET or FACTOR	1 - Baseline	Top & Rear	Hood & Read	0	0	0	0	0	0	0	0	0	0	0
Time - Design		1	1	1	1	1	1	1	1	1	1	1	1	1	1
Time - Build		1	1	1	1	1	1	1	1	1	1	1	1	1	1
Time - Test		1	1	1	1	1	1	1	1	1	1	1	1	1	1
Time Score	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Cost - Prototype	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00
Cost - Production	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00
Cost Score	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Implementation Time	4.5	3	13.5	2	9	0	0	0	0	0	0	0	0	0	0
Forward FOV	7	3	2.1	2	1.4	3	2.1	1	1	1	1	1	1	1	1
Reverse FOV	4.5	3	13.5	4	18	3	13.5	1	1	1	1	1	1	1	1
Side FOV	1.5	3	4.5	3	4.5	3	4.5	1	1	1	1	1	1	1	1
Blind Spots	7.5	3	22.5	1	7.5	3	22.5	1	1	1	1	1	1	1	1
Weight	0.5	3	1.5	3	1.5	2	1.5	1	1	1	1	1	1	1	1
Material Cost	4.5	3	13.5	2	9	0	0	0	0	0	0	0	0	0	0
Perceptive Height	4.5	3	13.5	5	13.5	1	4.5	1	1	1	1	1	1	1	1
Vertical Clearance	1.5	3	4.5	3	4.5	5	7.5	1	1	1	1	1	1	1	1
	0	9	2.7	0	0	0	0	0	0	0	0	0	0	0	0
	0	10	3	3	3	3	3	0	0	0	0	0	0	0	0
	0	11	3	3	3	3	3	0	0	0	0	0	0	0	0
TOTAL		198.0	81.5	74.5	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
RANK		100.0%	41.2%	37.6%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%
MAX		198.0	100.0%	41.2%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%	10.1%

NOTE: User fills in purple areas, gold areas are calculated or fixed  
Light blue areas filled from printing mark

BASELINE = Top & Front

Design Idea Descriptions

2	Top & Rear
3	Hood & Read
4	
5	
6	
7	
8	
9	
10	

Timescore(j) = Timescore(B)\*(TD(j)/TD(B) + TB(j)/TB(B) + TT(j)/TT(B))/3  
 Costscore(j) = Costscore(B)\*(Cprod(j)/Cprod(B) + Cprod(j)/Cprod(B))/2  
 Total(j) = SUM(Factor(i)\*Comparison(i,j)) + (Timescore(B)\*Timescore(j)) + (Costscore(B) - Costscore(j))  
 Comparison(j) = 5 if idea "i" is much better than baseline for criteria "j"  
 Comparison(j) = 4 if idea "i" is better than baseline for criteria "j"  
 Comparison(j) = 3 if idea "i" is same as baseline for criteria "j"  
 Comparison(j) = 2 if idea "i" is worse than baseline for criteria "j"  
 Comparison(j) = 1 if idea "i" is much worse than baseline for criteria "j"

Figure C.2: Lidar Configuration Tradeoff Analysis Summary



Project: RSI Rover  
 System: Sensor Configuration  
 Date: 11/02/2015

Criterion	1	2	3	4	5	6	7	8	9	10	11	12	SUM	FACTOR
1) Implementation Time	1	0	0	0	0.5	1	1	0.5	1	1	0	0.5	3.5	9
2) Forward FOV	0	1	0	0	0	1	1	1	1	1	0.5	0.5	9	3.5
3) Reverse FOV	0	0	1	0	0	1	1	0.5	0	1	0	0	3.5	3.5
4) Side FOV	0	0	0	1	0.5	1	1	0.5	1	1	1	0.5	7.5	7.5
5) Blind Spots	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6) Weight	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7) Material Cost	0.5	0	0.5	0	0	1	0	1	1	1	0	0.5	4.5	4.5
8) Perspective Height	1	0	1	0.5	0	1	0	1	1	1	0.5	0	5	5
9) Vertical Clearance	0	0	0	0	0	1	0	0	0	0	0	0	1	1
10) Electrical Power	1	0.5	1	0	0.5	1	1	0.5	1	1	0	1	7.5	7.5
11) Robustness	0.5	0.5	1	0.5	0.5	1	0.5	1	1	1	0	1	6.5	6.5
12)	1	1	1	1	1	1	1	1	1	1	1	1	11	11

Fill in Purple squares above

Fill in upper triangle of the matrix

Working across each row, determine if the criterion in that row is more important (1), same importance (0.5) or less important (0) than the criterion in that column

Assign weighting factors for each criterion in the Yellow squares

Figure C.3: Sensor Configuration Tradeoff Analysis Priority Matrix

Design Project = <b>ISL Rover</b>		System = <b>Lidar Configuration</b>		DESIGN IDEAS										
CRITERIA	TARGET or FACTOR	1 - Baseline	Case B: 6 Cameras	Case C: 2 Cameras	0	0	0	0	0	0	0	0	0	0
Time - Design	1	1	1	1										
Time - Build	1	1	1	1										
Time - Test	1	1	1	1										
Time Score	10	10	10	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cost - Prototype	1 \$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00										
Cost - Production	1 \$ 1.00	\$ 1.00	\$ 1.00	\$ 1.00										
Cost Score	10	10	10	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Implementation Time	3.5	3	10.5	4	14	4	14	5						
Forward FOV	9	3	2.7	3	27	3	27	3						
Reverse FOV	3.5	3	10.5	3	10.5	3	10.5	3						
Side FOV	7.5	3	22.5	4	30	3	22.5	4						
Blind Spots	7	3	21	2	14	3	21	2						
Weight	0	3	0	3	0	3	0	3						
Material Cost	4.5	3	13.5	4	18	3	13.5	4						
Perceptive Height	3	3	15	4	20	3	15	4						
Vertical Clearance	3	3	15	3	15	3	15	3						
Electrical Power	3.5	3	22.5	3	22.5	3	22.5	3						
Robustness	6.5	3	19.5	4	26	3	19.5	4						
TOTAL	11	3	198.0	0	185.0	0	168.5	20.0	20.0	10.1%	10.1%	10.1%	10.1%	20.0
RANK														
% MAX			100.0%		93.4%		85.1%							

NOTE: User fills in purple areas, gold areas are calculated or fixed  
Light blue areas filled from printing mark

BASELINE = Case A: 4 Cameras mounted on roll cage

Design Idea Descriptions  
 2 Case B: 6 Cameras mounted on roll cage  
 3 Case C: 2 Cameras on roll cage and 2 on hood  
 4  
 5  
 6  
 7  
 8  
 9  
 10

Timescore(j) = Timescore(B)\*(TD(j)/TD(B) + TB(j)/TB(B) + TT(j)/TT(B))/3  
 Costscore(j) = Costscore(B)\*(Cprot(j)/Cprot(B) + Cprod(j)/Cprod(B))/2  
 Total(j) = SUM(Factor(i)\*Compscore(i,j)) + (Timescore(B)\*Timescore(j)) + (Costscore(B) - Costscore(j))  
 Compscore(j) = 5 if idea "i" is much better than baseline for criteria "j"  
 Compscore(j) = 4 if idea "i" is better than baseline for criteria "j"  
 Compscore(j) = 3 if idea "i" is same as baseline for criteria "j"  
 Compscore(j) = 2 if idea "i" is worse than baseline for criteria "j"  
 Compscore(j) = 1 if idea "i" is much worse than baseline for criteria "j"

Figure C.4: Sensor Configuration Tradeoff Analysis Summary

# Appendix D: Budget

Part Name	Part Description	Notes	Qty	Price	Actual or Est Price	Item Total
<b>Computing/Communication</b>						
RS422 to USB adapter	GearMo® Pro 5ft. USB to RS485 / RS422 FTDI Chip		1	\$29.95	Actual	\$29.95
RS232 to USB adapter			2	\$10.86	Actual	\$21.72
RAM Upgrade for Laptop	16GB Ram (max supported by laptop)		1	\$100.00	Estimated	\$100.00
SSD for Laptop	SSD Hard Drive (prevent vibration issues with platter hard drive)		1	\$150.00	Estimated	\$150.00
<b>Power Systems</b>						
120VAC to 24VDC power supply	SUPERNIGHT(TM) 24V 15A 360W DC Regulated Switching Power Supply		1	\$24.99	Actual	\$24.99
24V/DC/DC converter			1	\$150.98	Actual	\$150.98
Replacement Car Battery			3	\$130.00	Estimated	\$390.00
<b>Electronic Components</b>						
Sensing Packages	General Budget for Sensing Packages		3	\$200	Estimated	\$600
IP Cameras			4	\$49.95	Actual	\$199.80
USB 3.0 Hub	7 port powered USB hub		1	\$19.99	Actual	\$19.99
Cable Clam Wire Pass Through	Side entry cable pass through. Connector up to 1 inch, cable up to 0.28 inch (Option 2)	Cable: 0.22in Diam	1	\$5.99	Actual	\$5.99
Replacement Switches			2	\$12.00	Estimated	\$24.00
Replacement Tachometer	85-S-NG-QFLA-000		1	\$50.21	Actual	\$50.21
UM7 AHRS (IMU)			1	\$175.00	Actual	\$175.00
Cable Clam Wire Pass Through	Cable pass through for connectors up to 0.83 inches (Option 1)		1	\$16.22	Actual	\$16.22
Ethernet Panel Pass Through			1	\$10.33	Actual	\$10.33
USB Panel Pass Through			1	\$7.60	Actual	\$7.60
<b>Shop Supplies</b>						
Exhaust Vent Tube	Rubber Duct Hose	Safely vent exhaust out	1	\$100.00	Estimated	\$100.00
Floor Jack		Replace Broken jack	1	\$80.00	Estimated	\$80.00
Soldering Iron			1	\$45.00	Estimated	\$45.00
<b>Miscellaneous</b>						
Gasoline	5 Gallons		1	\$4.00	Estimated	\$4.00
Additional Fuel			1	\$60.00	Estimated	\$60.00
<b>GRAND TOTAL</b>						<b>\$2,265.78</b>

Figure D.1: Budget

# Appendix E: Gantt Chart

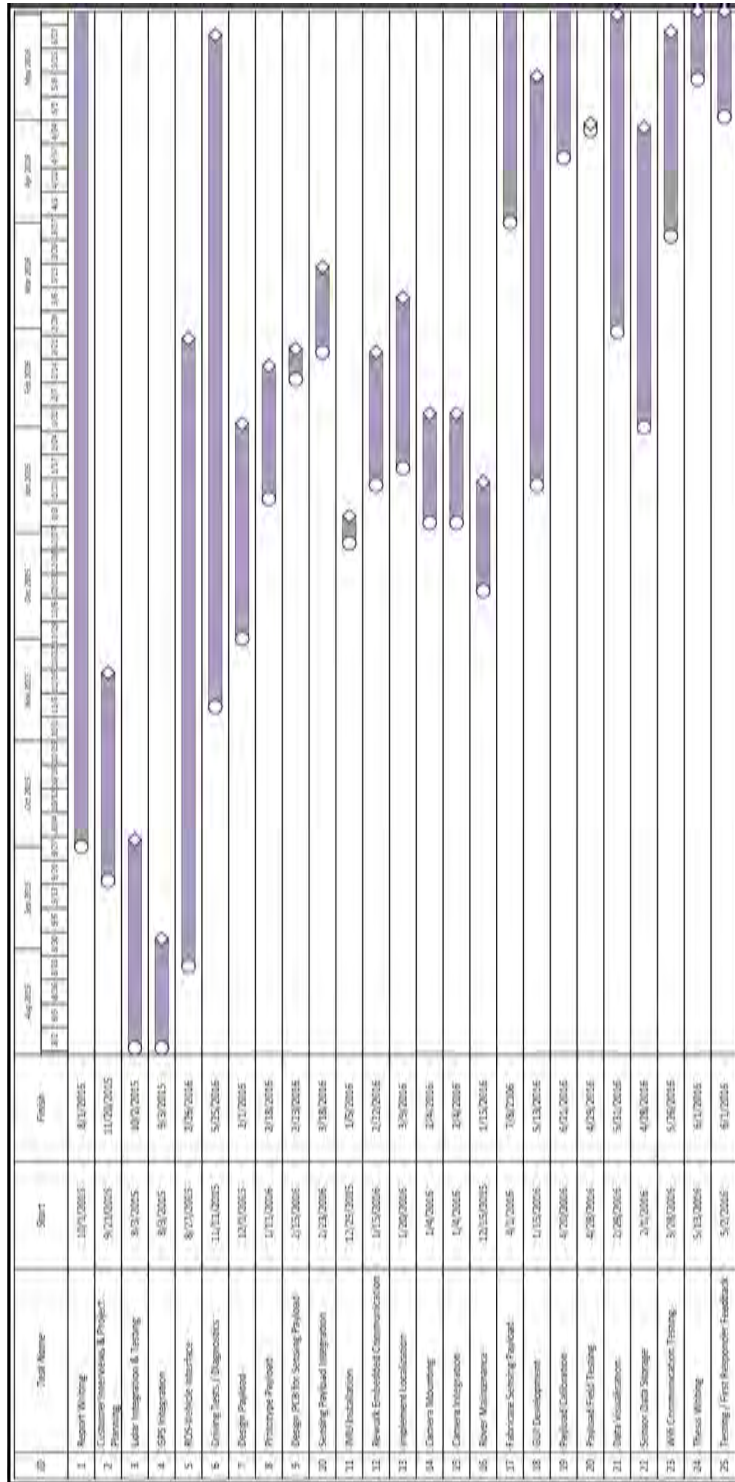


Figure E.1: Gantt Chart

# Appendix F: Power Budget

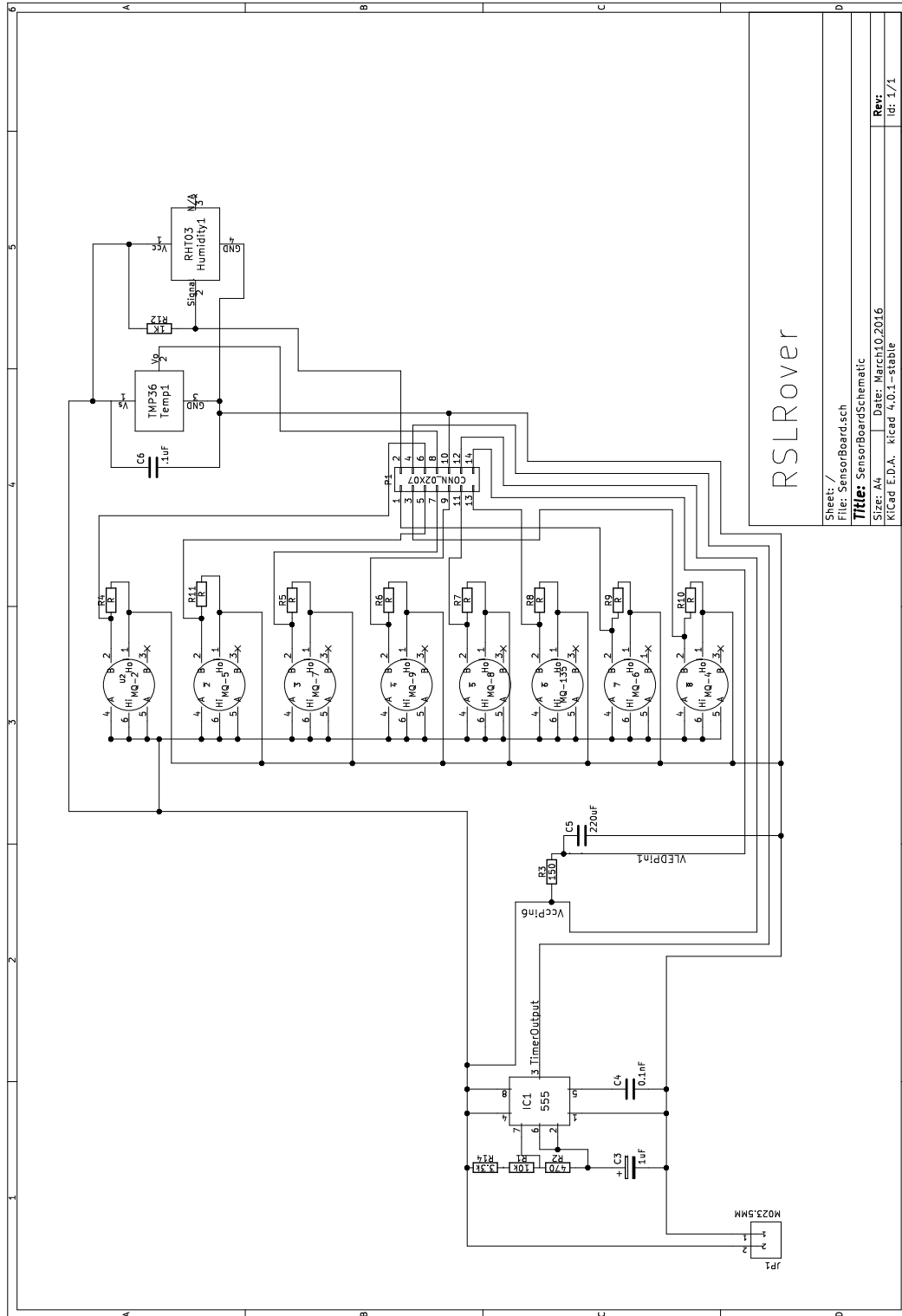
The following power budget was adapted from the 2014 senior design team, updated with the new hardware that we plan to add to the rover [?]. We hope to attain a more accurate understanding of how much power each component draws at the quarter progresses as we can measure the current draw of each component and update the spreadsheet.

Component (VEHICLE)	Voltage (V)	Max Current (A)	Qty	Max Total Power (W)	Duty Cycle	Avg Total Power (W)	V notes	A notes
Arduino Mega	5	0.1	2	1.0	100.00%	1.0		est
Arduino Uno	5	0.1	5	2.5	100.00%	2.5		est
Motor Controller 1	24	1	1	24.0	100.00%	24.0		est
Motor Controller 2	12	1	1	12.0	100.00%	12.0		est
Steering Actuator	24	18	1	432.0	50.00%	216.0		
Brake Actuator	24	7.9	1	189.6	20.00%	37.9		
Transmission Actuator	24	1.8	1	43.2	10.00%	4.3		
E-brake Actuator	12	7	1	84.0	5.00%	4.2		
Throttle Actuator	24	0.79	1	19.0	80.00%	15.2		
Warning Beacon	12	1.3	1	15.6	100.00%	15.6	10-30V	10A fuse
LED sign	7.5	3.25	1	24.4	100.00%	24.4		
xBee	3.3	0.245	1	0.8	100.00%	0.8	2.1-3.6V	
Tach	12	0.25	1	3.0	100.00%	3.0	8-30V	est
Brake Pressure Transducer	12	0.25	1	3.0	100.00%	3.0	8-28V	est
GPS	5	0.1	1	0.5	100.00%	0.5		
Horn	12	5	1	60.0	1.00%	0.6		est
LMS 221	24	1	1	24.0	100.00%	24.0		typ
LMS 111	12	0.66	1	7.9	100.00%	7.9		typ
Lidar Gimbal	24	6	1	144.0	50.00%	72.0		rough est.
Sensor Packages	12	1	3	36	100.00%	36.0		rough est.
<b>STOCK ITEMS:</b>								
Tail lights				5	100.00%	5.0		
Brake lights				5	20.00%	1.0		
Headlights				35	100.00%	35.0		
Indicator				1	0.00%	0.0		
<b>Total</b>				<b>1,172.5</b>		<b>545.9</b>		
<b>Component (CONSOLE)</b>								
Arduino Mega	5	0.1	1	0.5	100.00%	0.5		est
Switches/buttons	5	0.1	1	0.5	100.00%	0.5		
Joystick	5	0.05	1	0.25	100.00%	0.25		
LCD screen	5	0.2	1	1	100.00%	1		
LEDs	5	0.013	20	1.3	70.00%	0.91		13 mA: need resistor?
xBee	3.3	0.245	1	0.8	100.00%	0.8		
<b>Total</b>				<b>4.4</b>		<b>4.0</b>		
<b>ALTERNATOR OUTPUT</b>				<b>250</b>				
<b>Battery Capacity</b>	<b>Individual Battery Capacity (Ah)</b>	<b>Number of Batteries</b>	<b>Power Capacity (VAh)</b>	<b>Avg Power</b>	<b>Avg Time</b>	<b>Peak Power</b>	<b>Min Time</b>	
	85	3	3060	545.9	5.60530416	1,172.5	2.60988935	

Figure F.1: Power Budget: Adapted from [?]



# Appendix G: Drawings



RSLRover

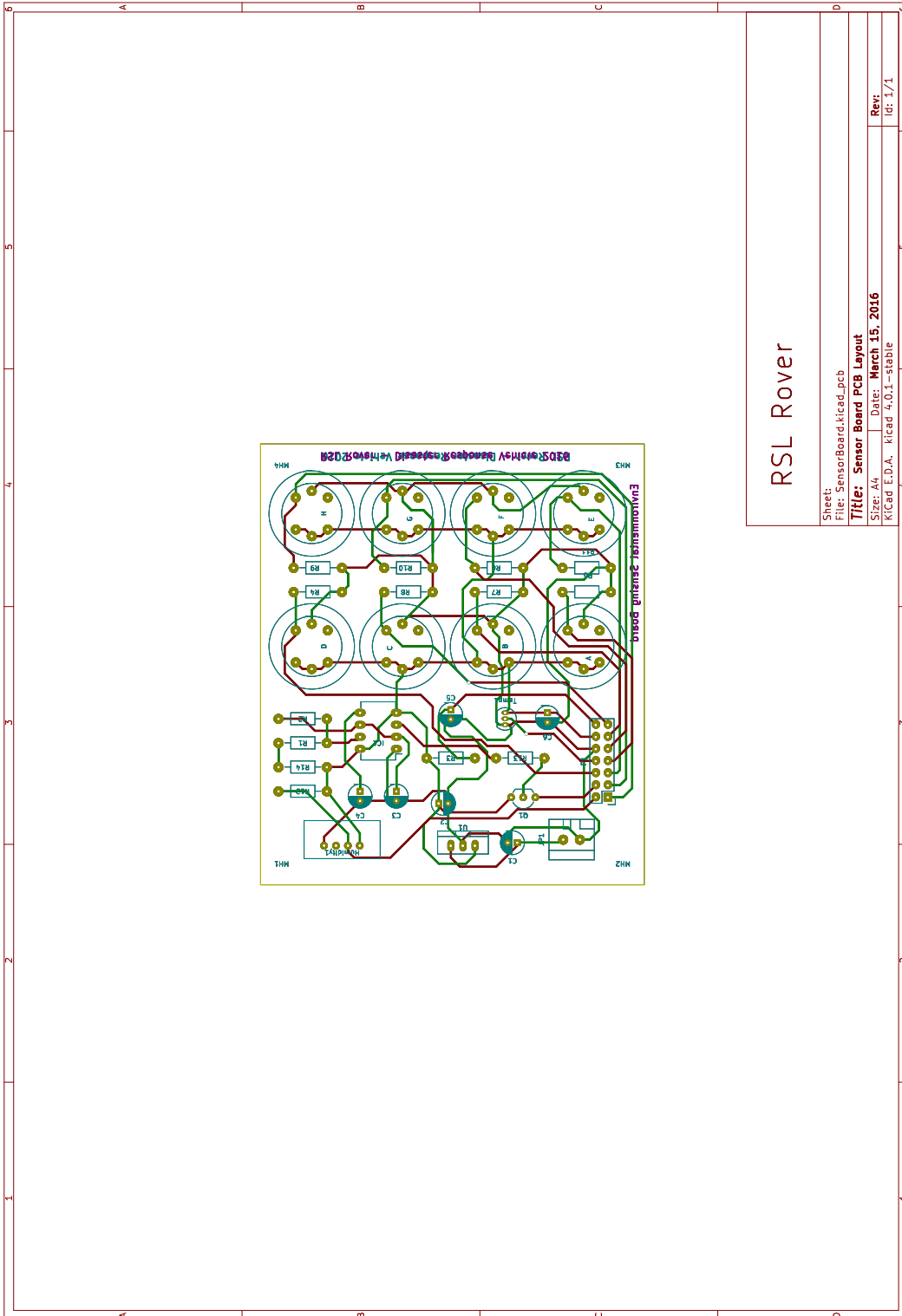
Sheet: /  
File: SensorBoard.sch

Title: SensorBoardSchematic

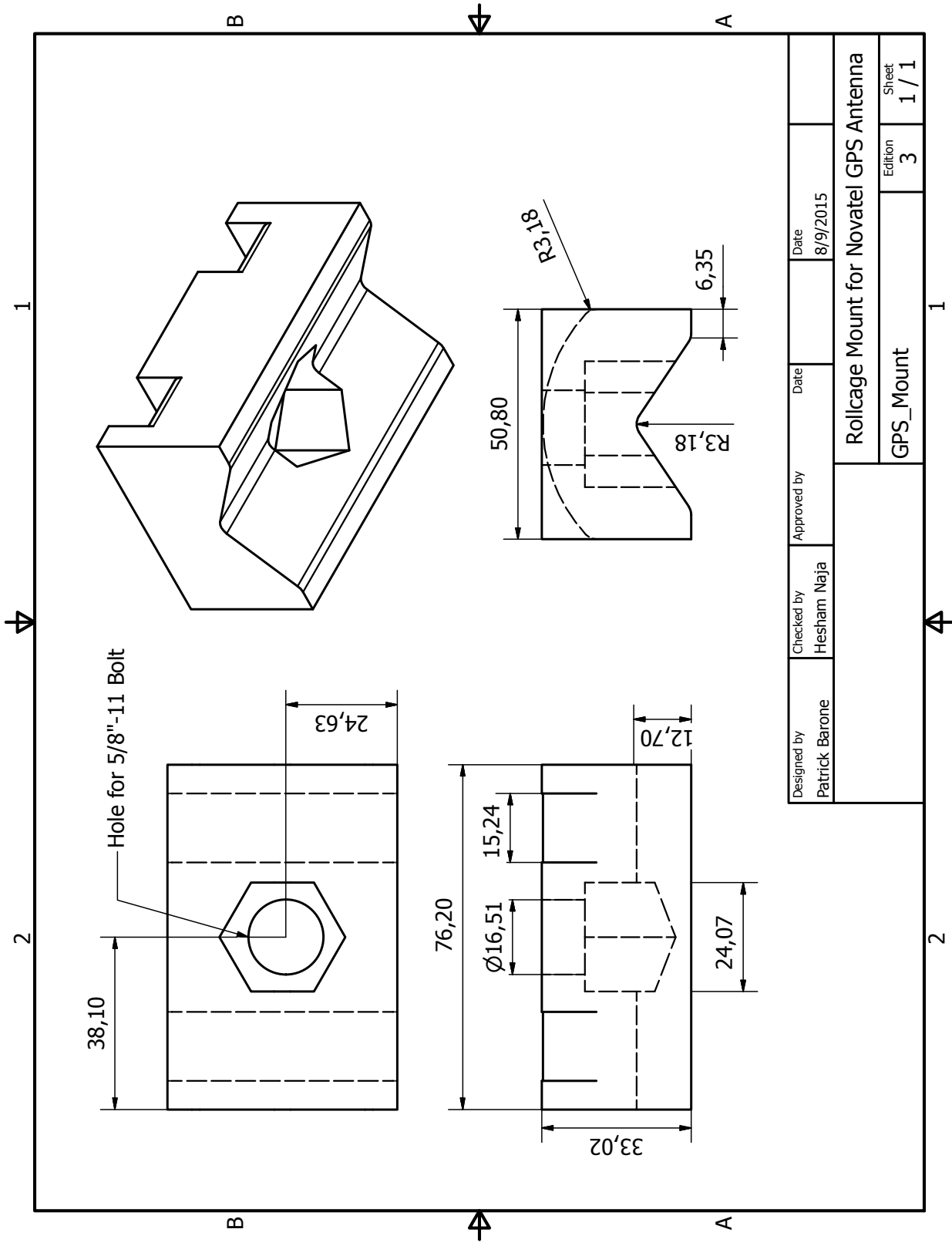
Size: A4 | Date: March10,2016

KiCad E.D.A. - KiCad 4.0.1 - stable

Rev:  
Id: 1/1







1

2

B

A

B

A

Hole for 5/8"-11 Bolt

38,10

24,63

76,20

Ø16,51

15,24

33,02

24,07

12,70

50,80

R3,18

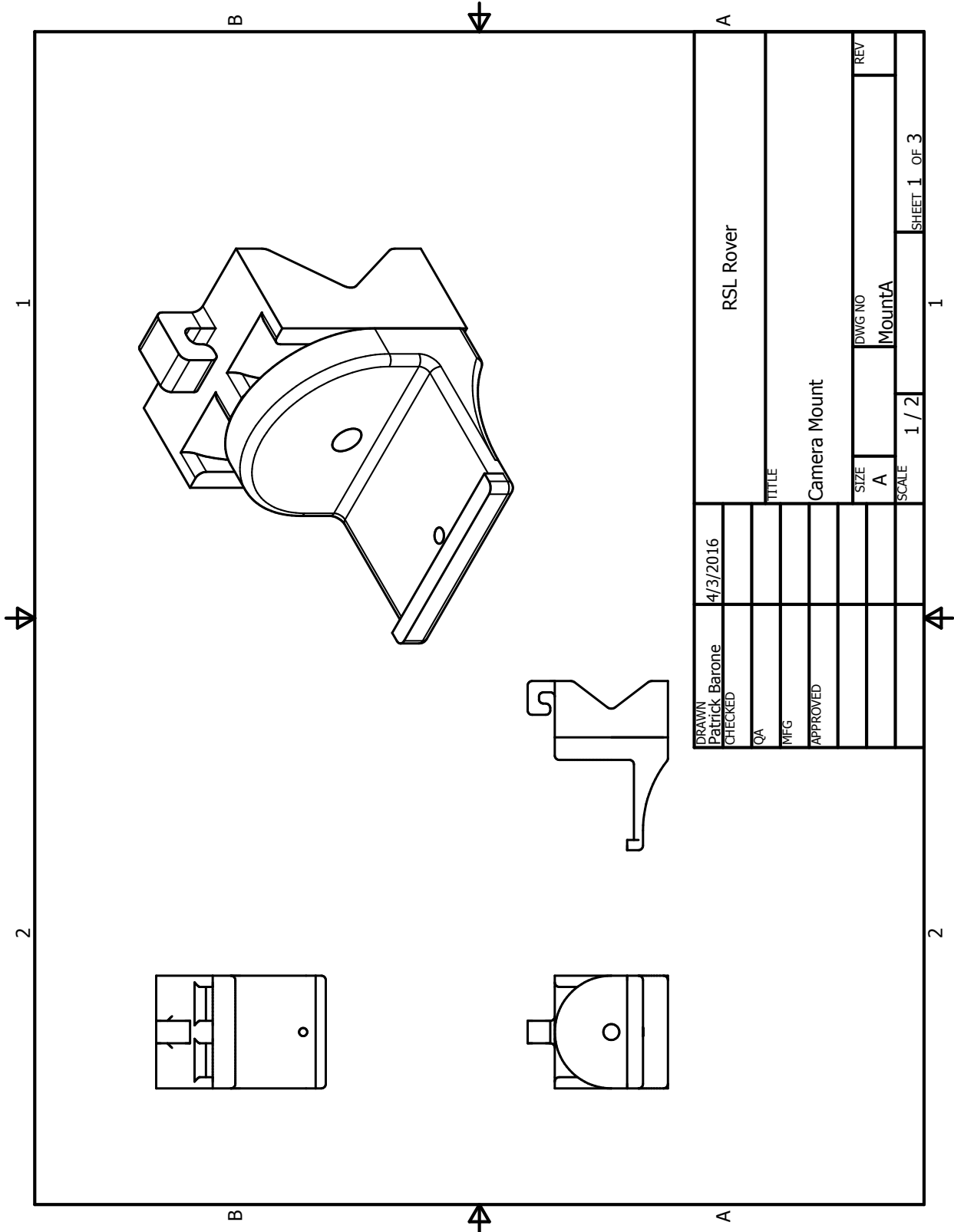
6,35

R3,18

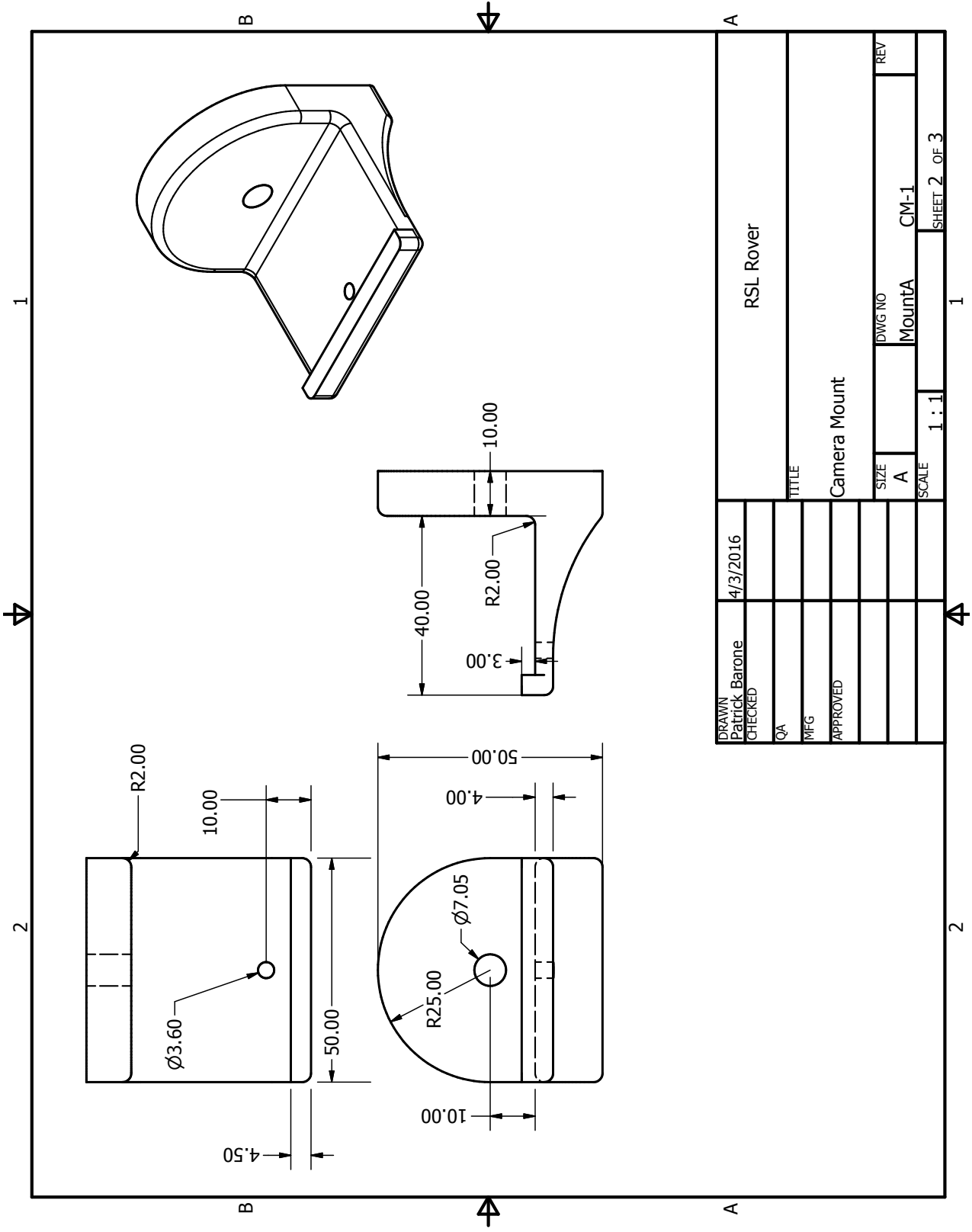
Designed by Patrick Barone	Checked by Hesham Naja	Approved by	Date 8/9/2015
Rollcage Mount for Novatel GPS Antenna			
GPS_Mount			Sheet 1 / 1
Edition 3			

1

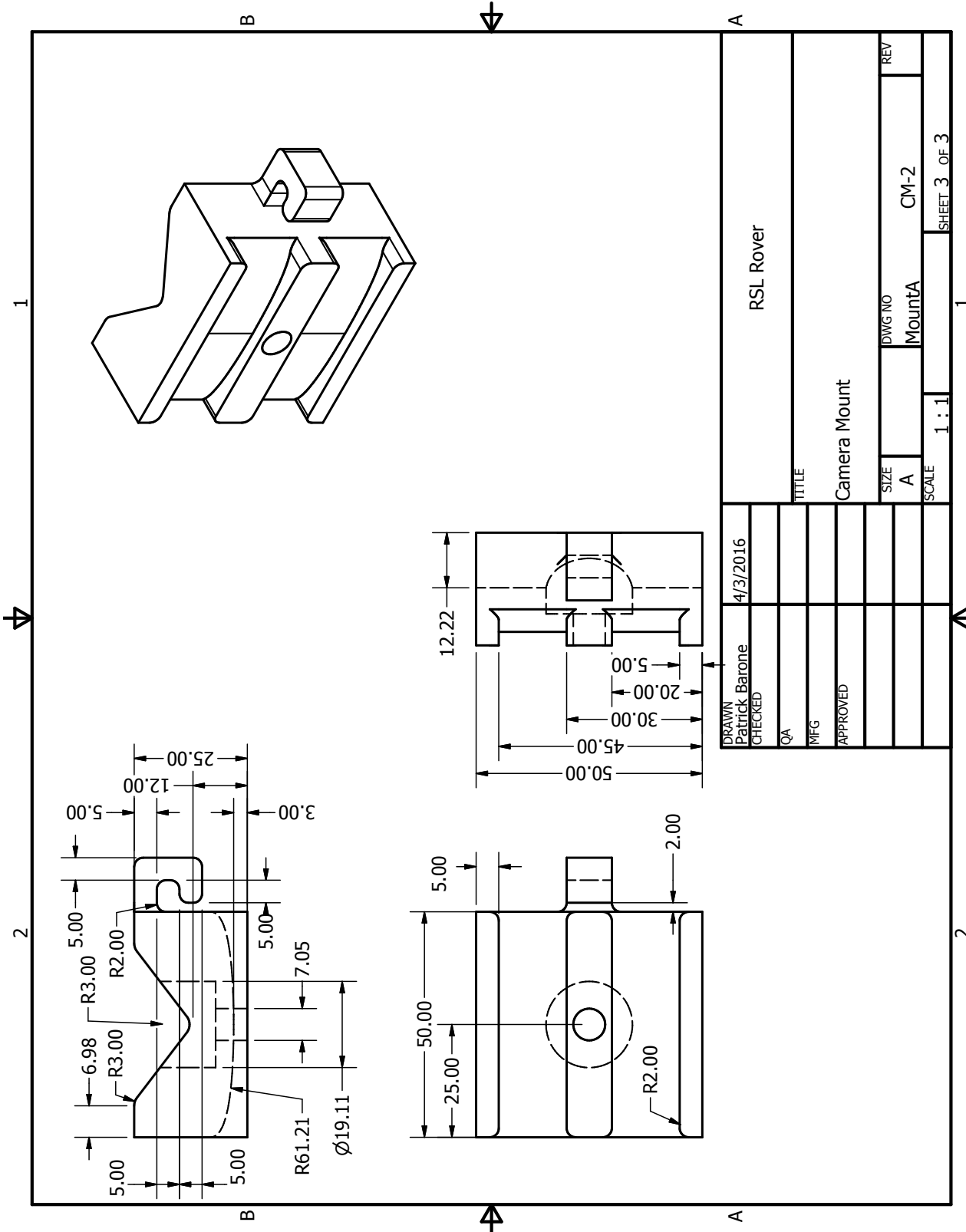
2



DRAWN Patrick Barone		4/3/2016		RSL Rover	
CHECKED					
QA				TITLE	
MFG				Camera Mount	
APPROVED				SIZE	
				A	
				DWG NO	
				MountA	
				REV	
				SCALE	
				1 / 2	
				SHEET 1 OF 3	



DRAWN	4/3/2016	RSL Rover	
CHECKED	Patrick Barone	MountA	
QA		SIZE	DWG NO
MFG		A	CM-1
APPROVED		SCALE	1 : 1
		TITLE	Camera Mount
		REV	
		SHEET 2 OF 3	



1

2

2

B

↓

A

DRAWN  
Patrick Barone  
CHECKED  
QA  
MFG  
APPROVED

4/3/2016

TITLE

Camera Mount

SIZE  
A

DWG NO  
MountA

REV  
CM-2

SCALE  
1 : 1

SHEET 3 OF 3

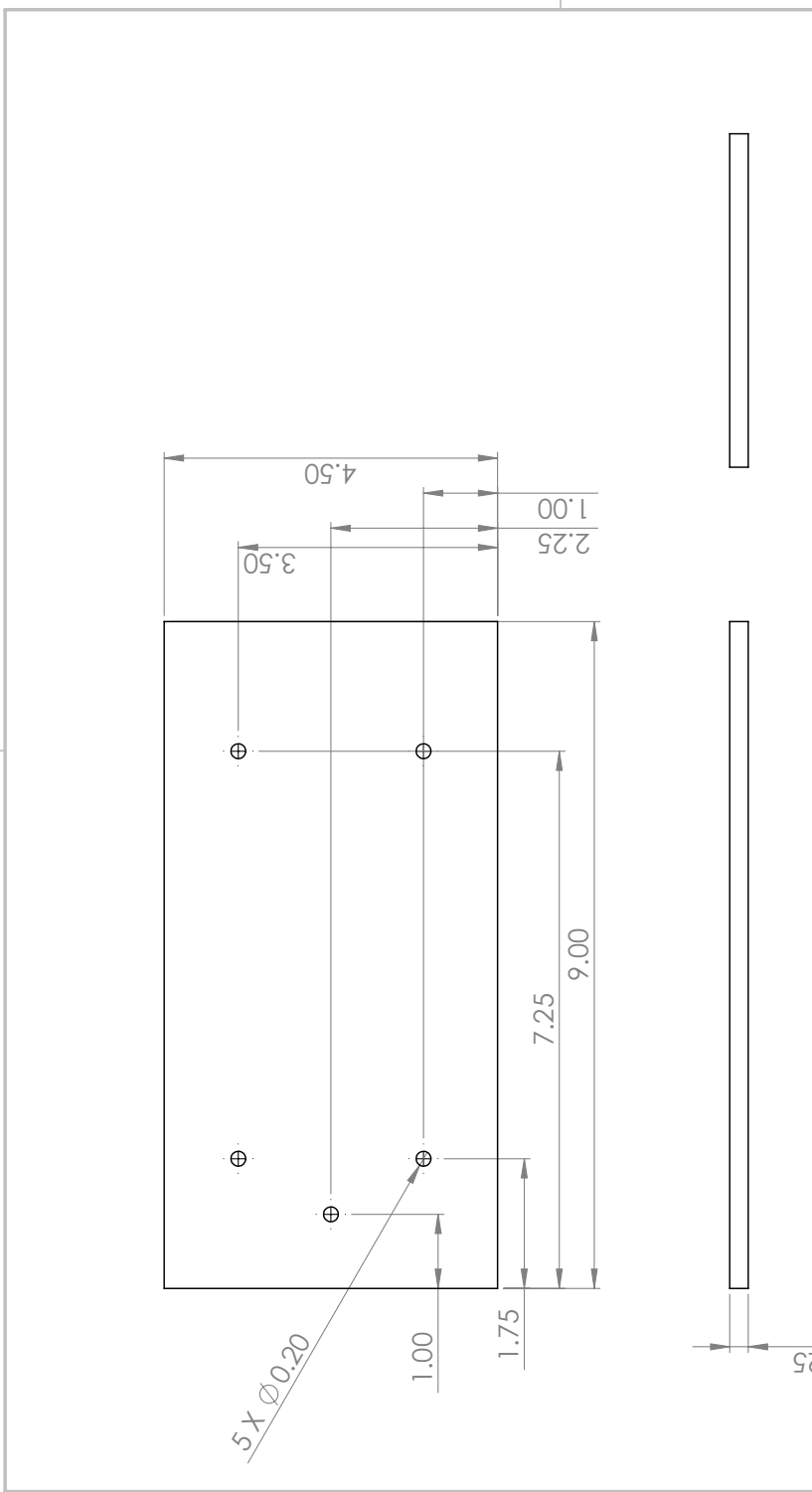
RSL Rover

1

G-7

1

2

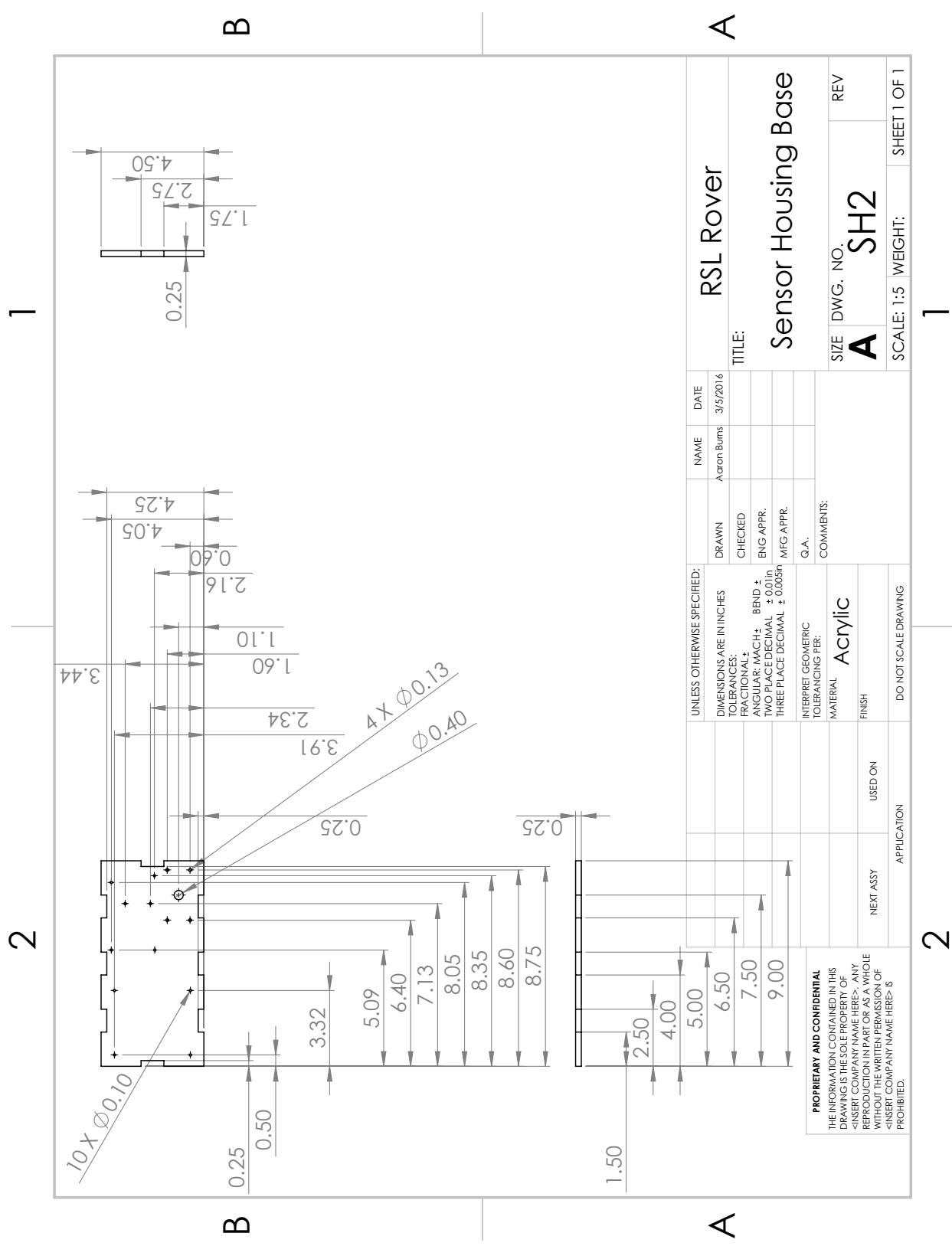


A

A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01in THREE PLACE DECIMAL ± 0.003in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	G.A.	CHECKED	Aaron Burns	3/5/2016	TITLE: Sensor Housing Front	
MATERIAL: Acrylic		ENG APPR.			SIZE: A	DWG. NO.: SH1
FINISH: USED ON		MFG APPR.			REV	
NEXT ASSY	APPLICATION	COMMENTS:			SCALE: 1:5	WEIGHT: SHEET 1 OF 1
DO NOT SCALE DRAWING		1				

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.



1

2

B

B

A

A

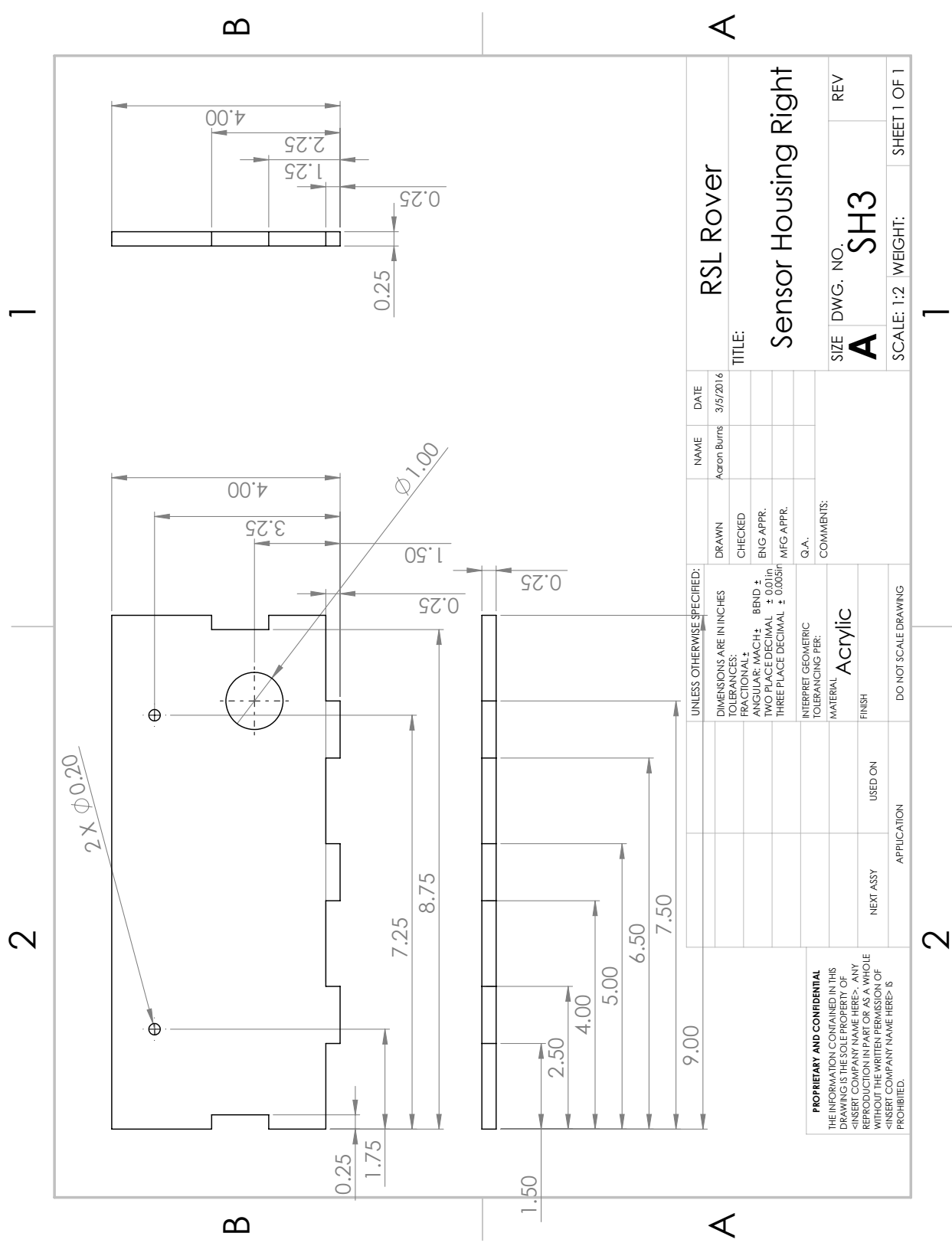
RSL Rover

Sensor Housing Base

DRAWN		NAME	DATE	RSL Rover	
CHECKED		Aaron Burns	3/5/2016	Sensor Housing Base	
ENG APPR.				TITLE:	
MFG APPR.				SIZE DWG. NO. SH2	
G.A.				SCALE: 1:5 WEIGHT:	
COMMENTS:				SHEET 1 OF 1	
UNLESS OTHERWISE SPECIFIED:					
DIMENSIONS ARE IN INCHES					
TOLERANCES:					
FRACTIONAL: ±					
ANGULAR: MACH ± BEND ±					
TWO PLACE DECIMAL ± 0.01in					
THREE PLACE DECIMAL ± 0.005in					
INTERPRET GEOMETRIC TOLERANCING PER:					
MATERIAL Acrylic					
FINISH					
DO NOT SCALE DRAWING					
NEXT ASSY		USED ON		APPLICATION	
<p><b>PROPRIETARY AND CONFIDENTIAL</b>          THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF &lt;INSERT COMPANY NAME HERE&gt;. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF &lt;INSERT COMPANY NAME HERE&gt; IS PROHIBITED.</p>					

1

2



1

2

B

B

A

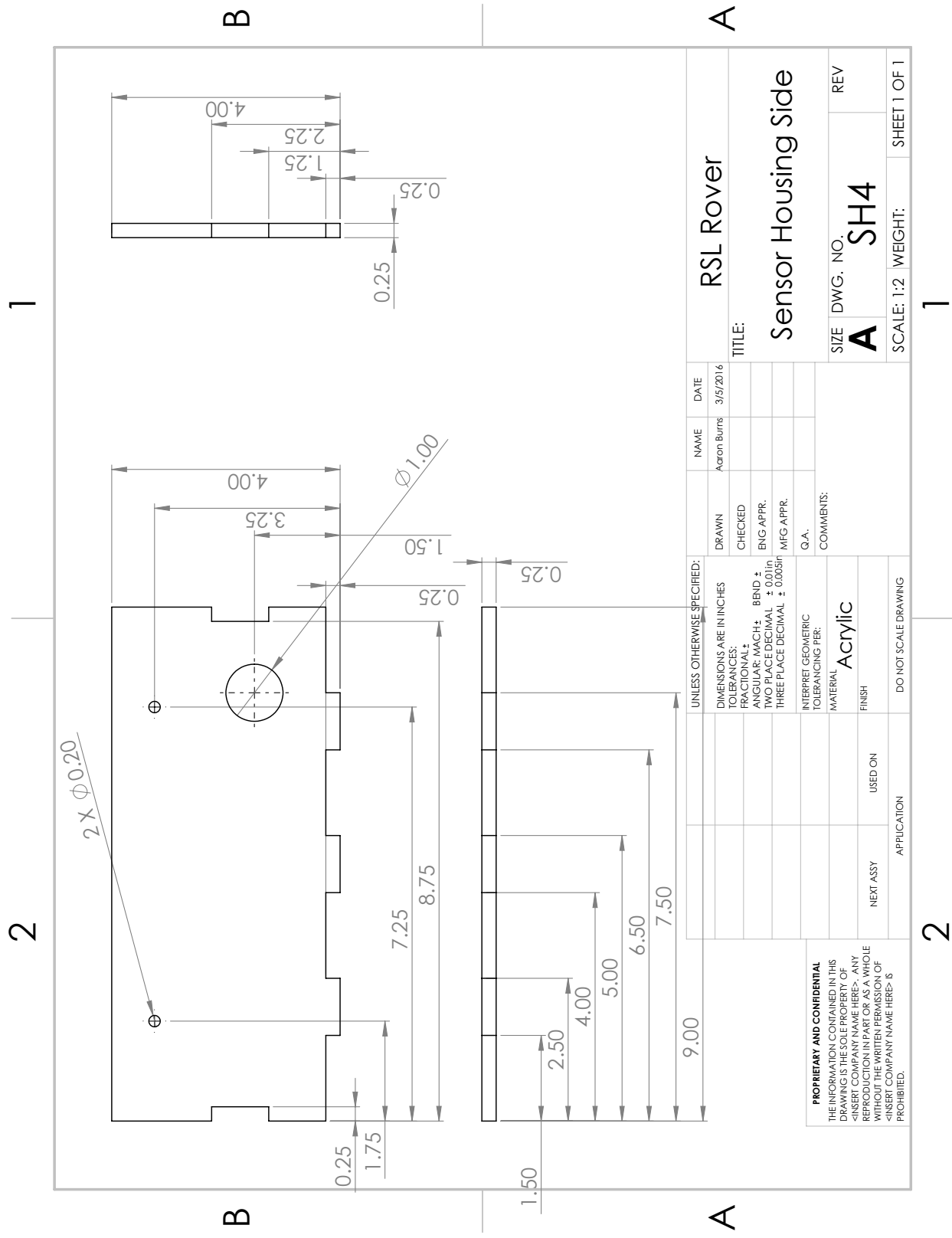
A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01 in THREE PLACE DECIMAL ± 0.005 in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	G.A.	CHECKED	Aaron Burns	3/5/2016	TITLE:	
MATERIAL	Acrylic	ENG APPR.			Sensor Housing Right	
FINISH		MFG APPR.			SIZE	DWG. NO.
NEXT ASSY		COMMENTS:			A	SH3
APPLICATION		DO NOT SCALE DRAWING			SCALE: 1:2	WEIGHT:
						SHEET 1 OF 1

**PROPRIETARY AND CONFIDENTIAL**  
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2



1

2

B

B

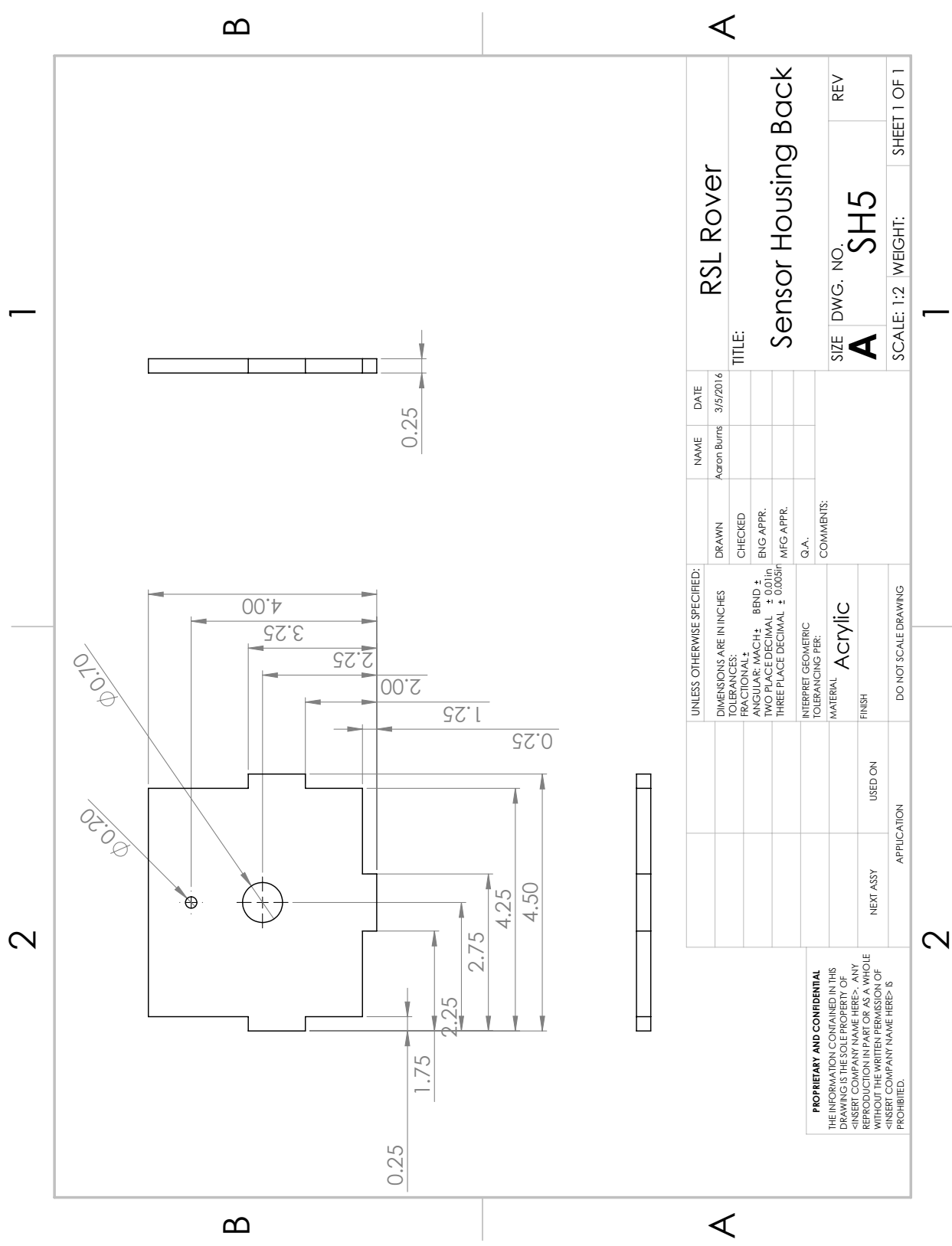
A

A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01 in THREE PLACE DECIMAL ± 0.005 in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	G.A.	CHECKED	Aaron Burns	3/5/2016	TITLE: Sensor Housing Side	
MATERIAL	Acrylic	ENG APPR.			SIZE	DWG. NO.
FINISH		MFG APPR.			A	SH4
NEXT ASSY		COMMENTS:			REV	
APPLICATION		DO NOT SCALE DRAWING			SCALE: 1:2	WEIGHT:
						SHEET 1 OF 1

**PROPRIETARY AND CONFIDENTIAL**  
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.



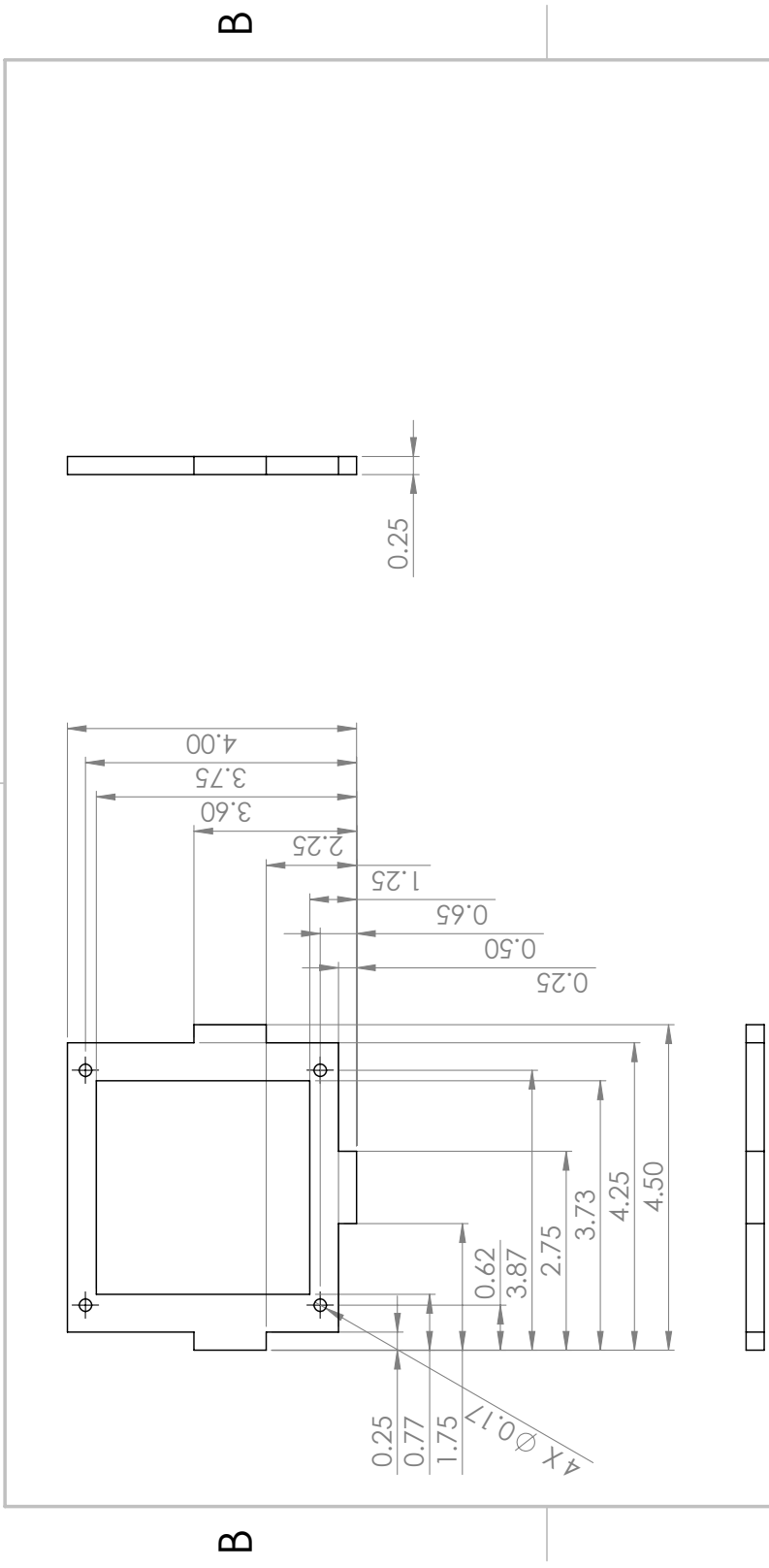


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01in THREE PLACE DECIMAL ± 0.005in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	MATERIAL	CHECKED	Aaron Burns	3/5/2016	TITLE:	
ACRYLIC	Acrylic	ENG APPR.			Sensor Housing Back	
FINISH	USED ON	MFG APPR.			SIZE	DWG. NO.
NEXT ASSY	APPLICATION	G.A.			A	SH5
DO NOT SCALE DRAWING		COMMENTS:			SCALE: 1:2	WEIGHT:
						SHEET 1 OF 1

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2

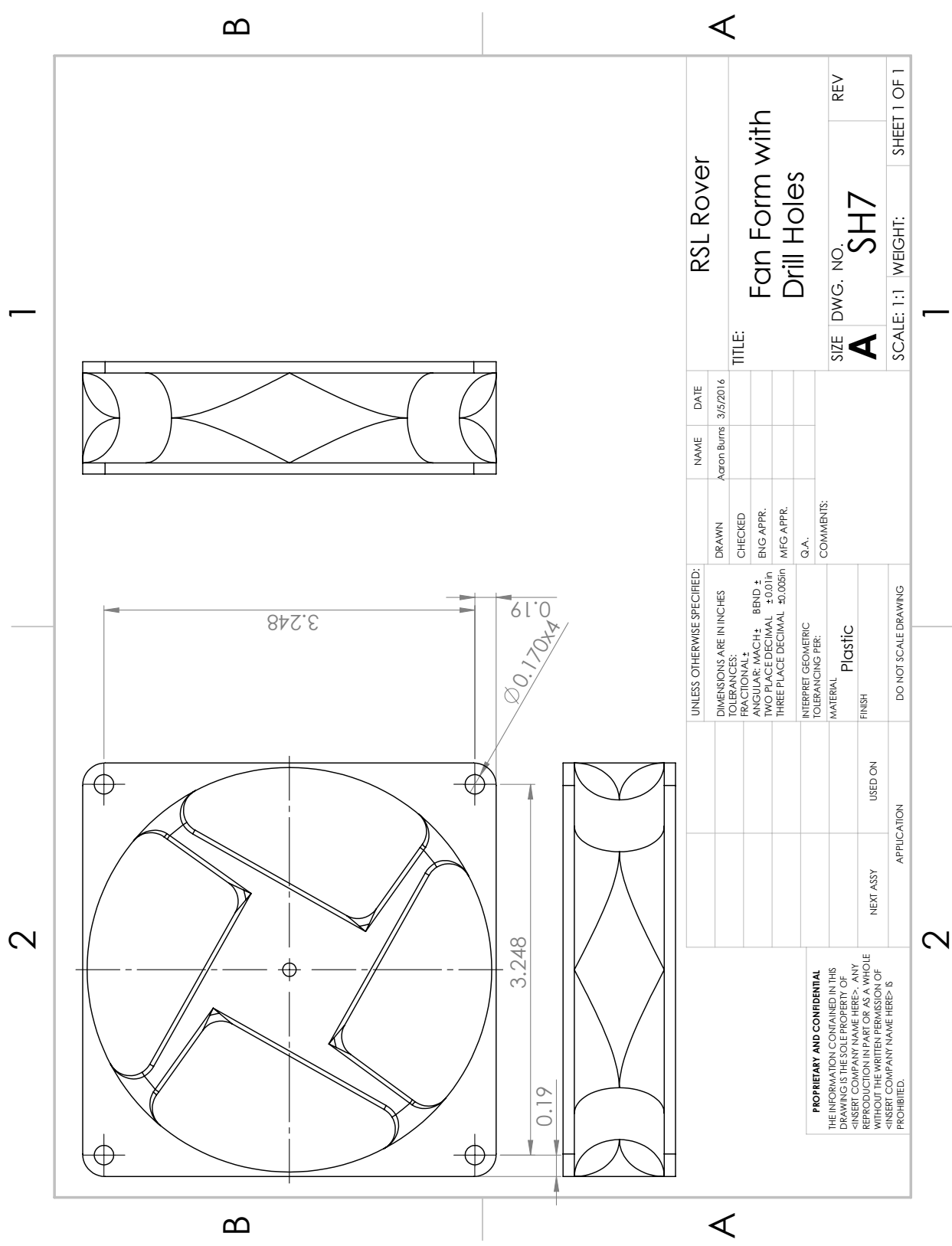


A

A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01in THREE PLACE DECIMAL ± 0.003in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:		CHECKED	Aaron Burns	3/5/2016	TITLE: Sensor Housing Front	
MATERIAL: Acrylic		ENG APPR.			SIZE	DWG. NO.
FINISH		MFG APPR.			A	SH6
NEXT ASSY		G.A.			REV	
APPLICATION		COMMENTS:			SCALE: 1:2	WEIGHT:
USED ON					SHEET 1 OF 1	
DO NOT SCALE DRAWING					1	

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.



1

2

B

B

A

A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01 In. THREE PLACE DECIMAL ± 0.005 In.		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	MATERIAL	CHECKED	Aaron Burns	3/5/2016	Fan Form with Drill Holes	
FINISH	Plastic	ENG APPR.			SIZE	DWG. NO.
NEXT ASSY	USED ON	MFG APPR.			A	SH7
APPLICATION		G.A.			SCALE: 1:1	WEIGHT:
		COMMENTS:				SHEET 1 OF 1
		DO NOT SCALE DRAWING				

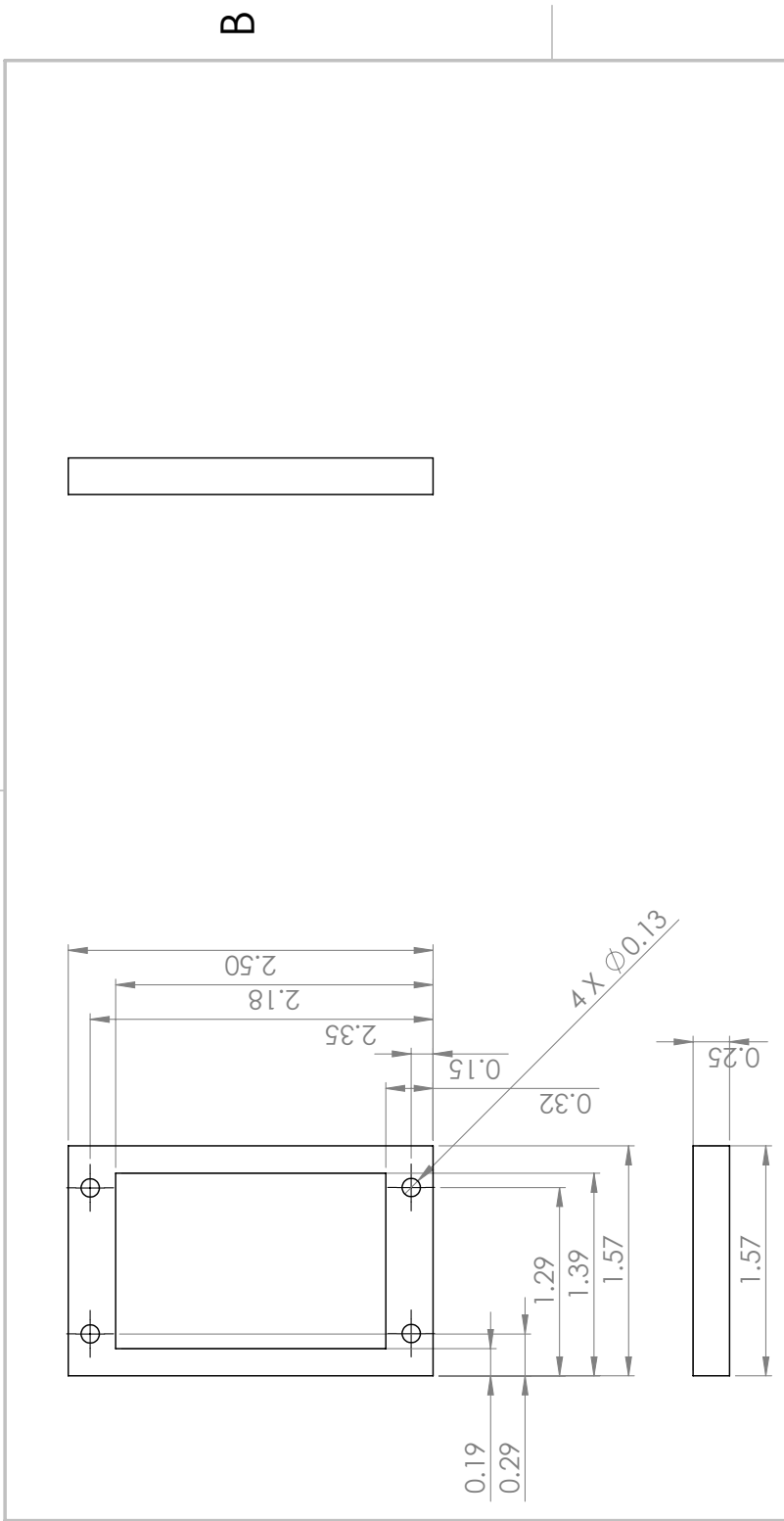
**PROPRIETARY AND CONFIDENTIAL**  
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2

1

2

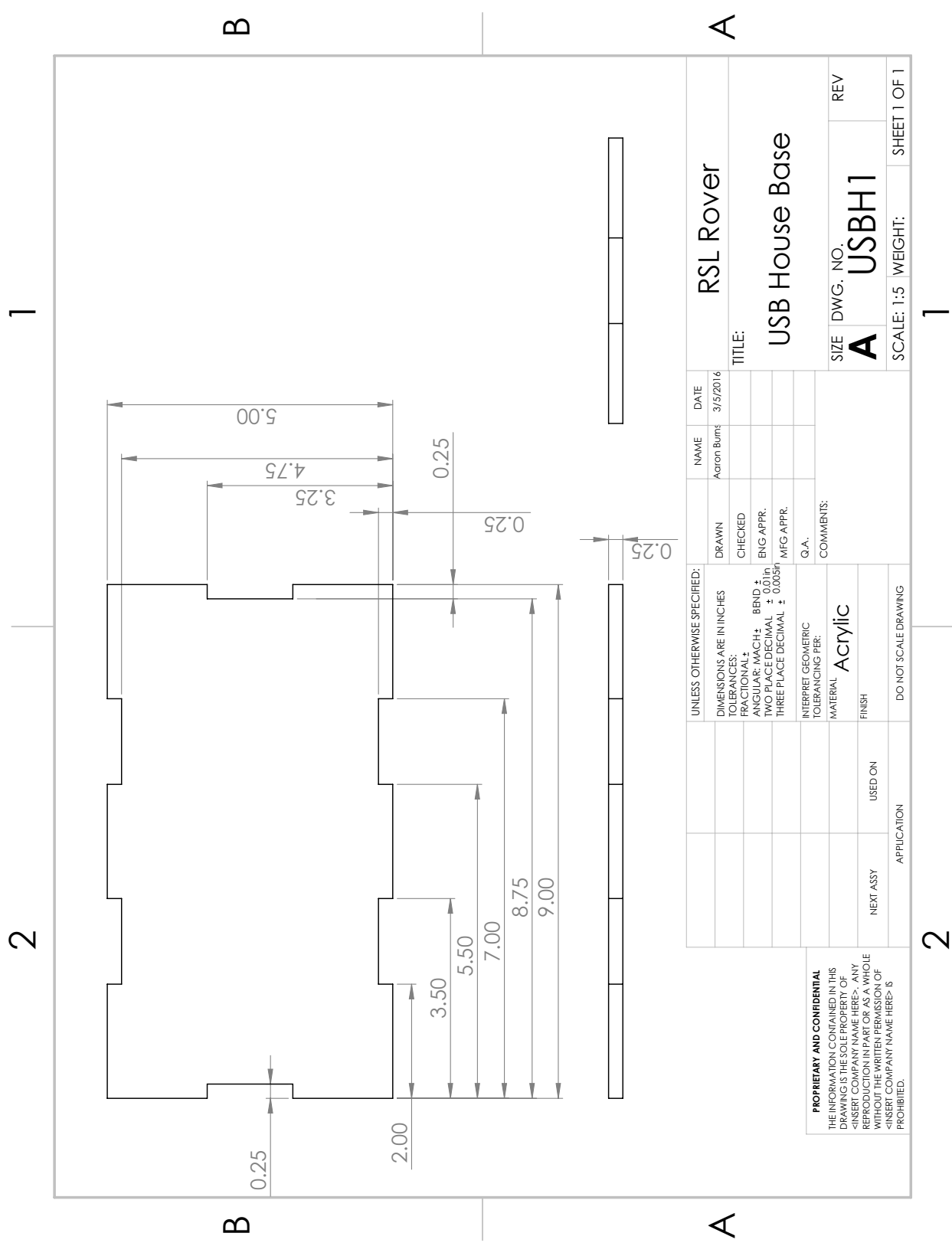


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01in THREE PLACE DECIMAL ± 0.003in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:		CHECKED	Aaron Burns	3/5/2016	TITLE:	
MATERIAL: Acrylic		ENG APPR.			Air Particulate Mount	
FINISH: USED ON		MFG APPR.			SIZE	DWG. NO.
NEXT ASSY		G.A.			A	SH8
APPLICATION		COMMENTS:			SCALE: 1:1	WEIGHT:
		DO NOT SCALE DRAWING				SHEET 1 OF 1

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2



1

2

B

B

A

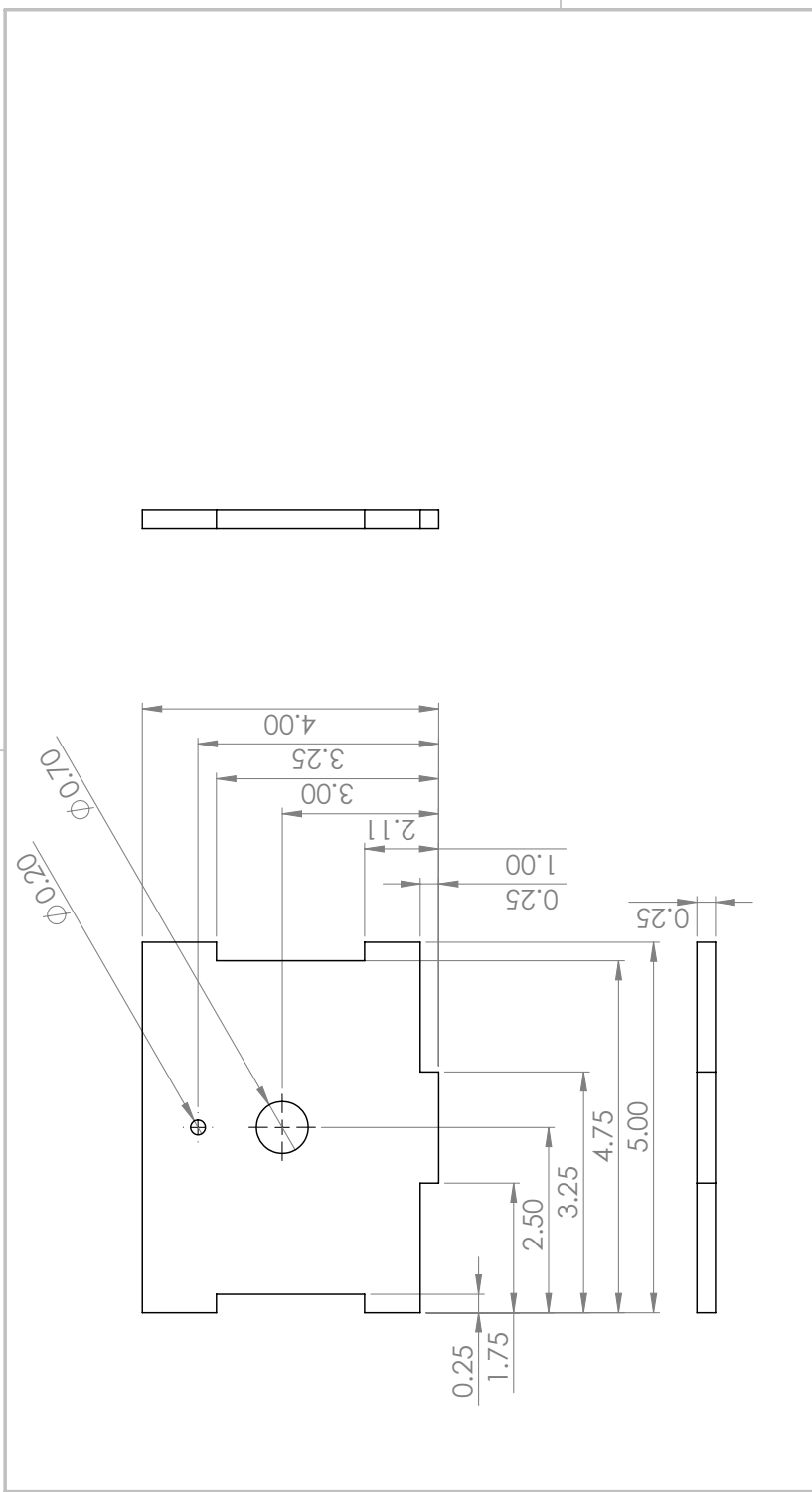
A

UNLESS OTHERWISE SPECIFIED:		NAME	DATE	RSL Rover	
DIMENSIONS ARE IN INCHES		Aaron Burt	3/5/2016	USB House Base	
TOLERANCES:		DRAWN	CHECKED	TITLE:	
FRACTIONAL: ±				SIZE DWG. NO. <b>A</b> USBH1	
ANGULAR: MACH ± BEND ±				SCALE: 1:5 WEIGHT: SHEET 1 OF 1	
TWO PLACE DECIMAL ± 0.01in					
THREE PLACE DECIMAL ± 0.000in					
INTERPRET GEOMETRIC TOLERANCING PER:					
MATERIAL: <b>Acrylic</b>					
FINISH: USED ON					
APPLICATION: NEXT ASSY					
DO NOT SCALE DRAWING					

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2



B

B

A

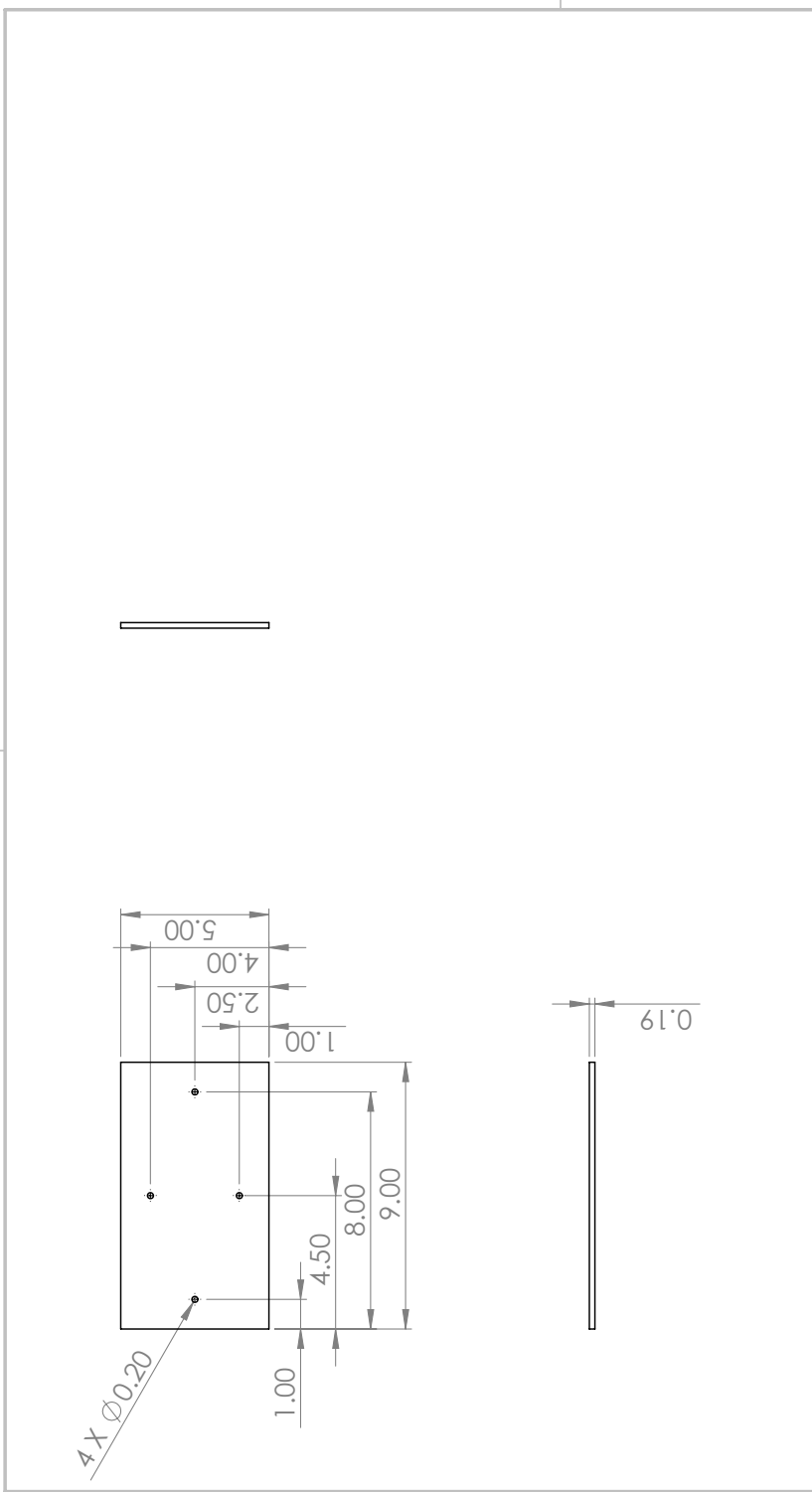
A

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± 0.01in THREE PLACE DECIMAL ± 0.005in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:		CHECKED	Aaron Burns	3/5/2016	TITLE: USB House Front	
MATERIAL: Acrylic		ENG APPR.			SIZE: A	DWG. NO.: USBH2
FINISH: USED ON		MFG APPR.			REV	
APPLICATION: NEXT ASSY		G.A.			SCALE: 1:2	WEIGHT: SHEET 1 OF 1
DO NOT SCALE DRAWING		COMMENTS:			1	

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

1

2



B

B

A

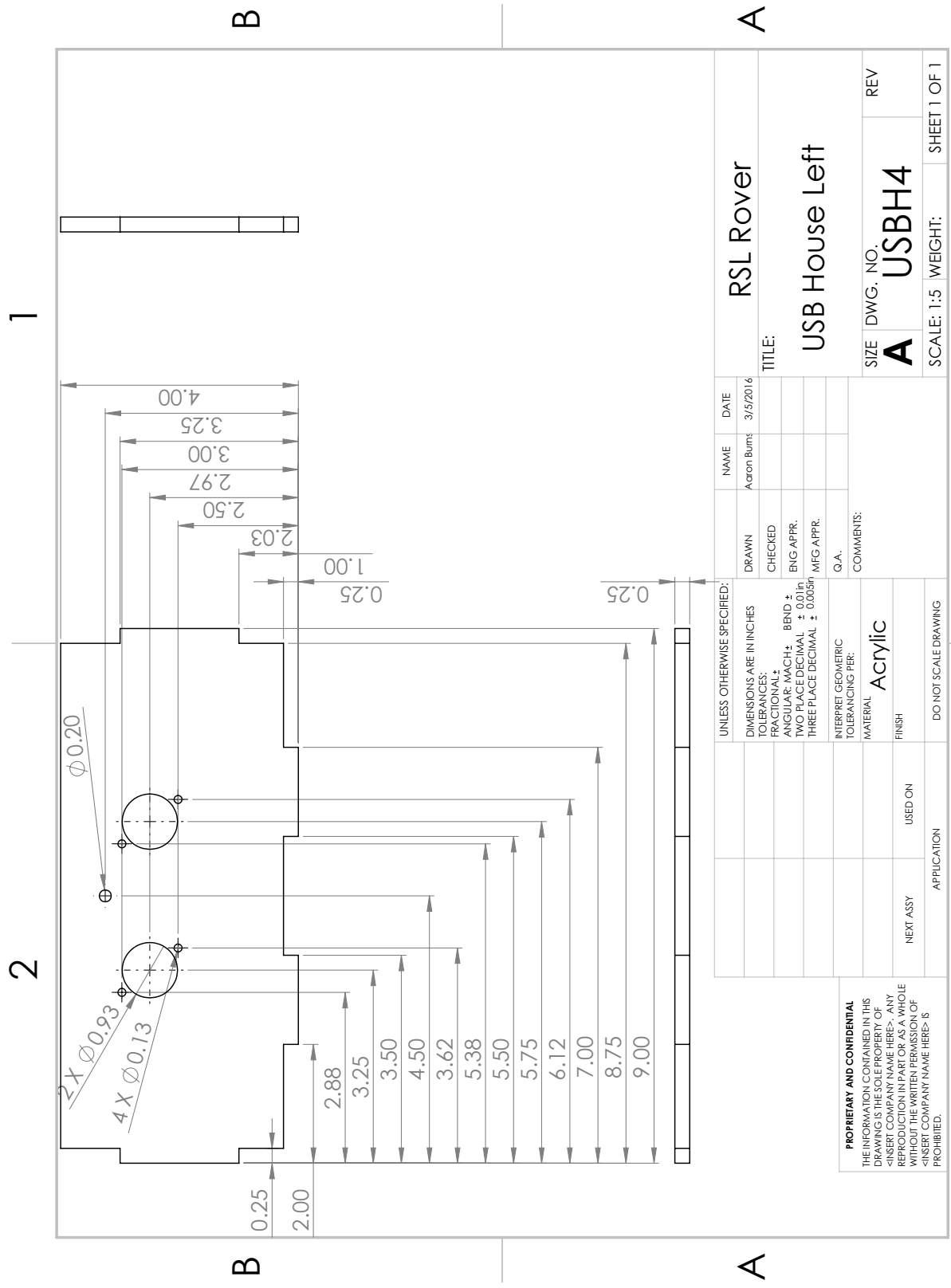
A

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: BEND ± ANGULAR: MACH ± 0.01in. TWO PLACE DECIMAL ± 0.005in. THREE PLACE DECIMAL ± 0.0005in.		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:		CHECKED	Aaron Burns	3/5/2016	TITLE: USB House Lid	
MATERIAL: Acrylic		ENG APPR.			SIZE: A	DWG. NO.: USBH3
FINISH: USED ON		MFG APPR.			REV	
NEXT ASSY		G.A.			SCALE: 1:5	WEIGHT:
APPLICATION		COMMENTS:			SHEET 1 OF 1	
DO NOT SCALE DRAWING					1	

2

1



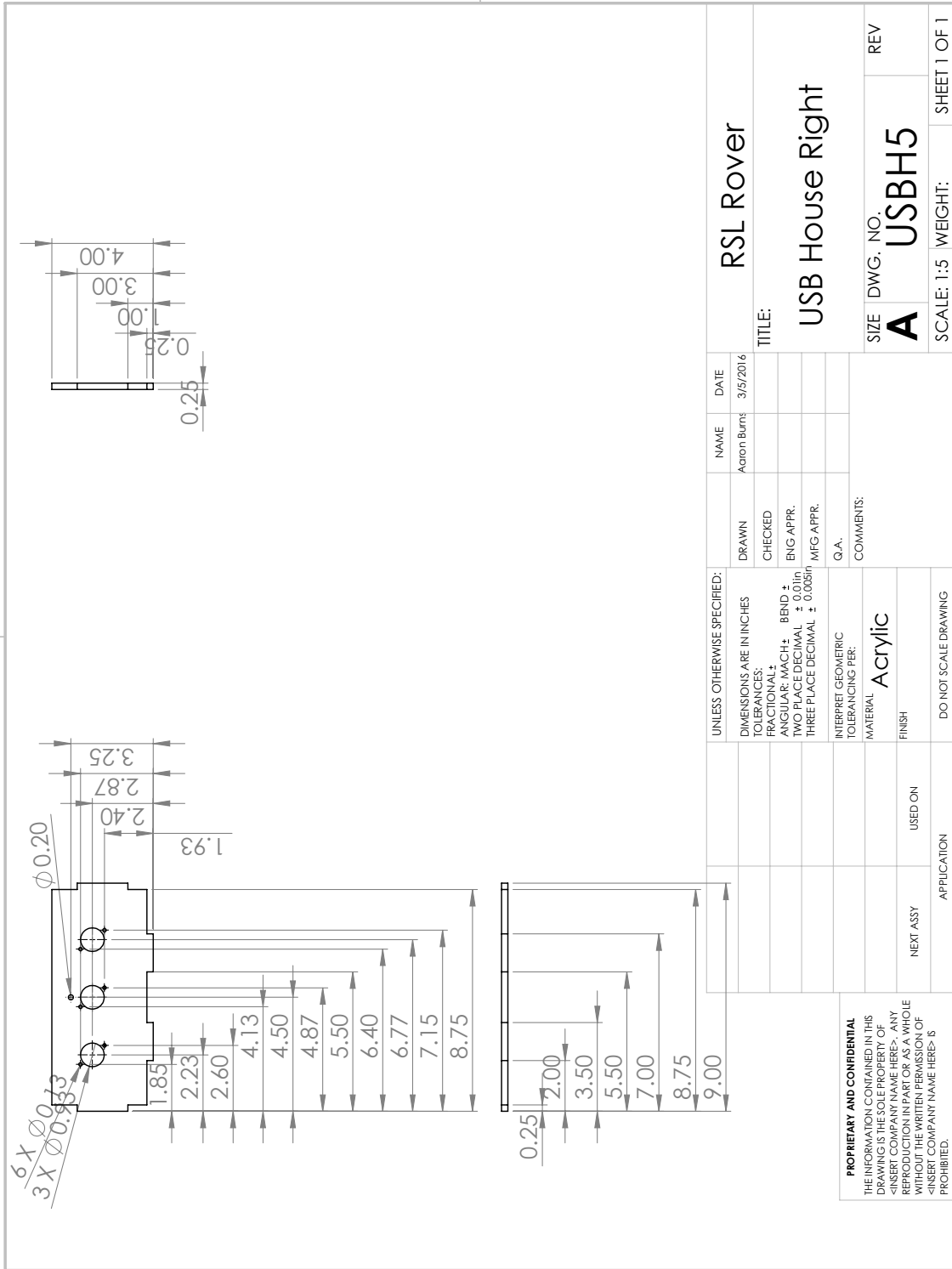
**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

UNLESS OTHERWISE SPECIFIED:		DIMENSIONS ARE IN INCHES	
TOLERANCES:		FRACTIONAL: ±	
ANGULAR: MACH ±		BEND ±	
TWO PLACE DECIMAL ±		0.01 in	
THREE PLACE DECIMAL ±		0.005 in	
INTERPRET GEOMETRIC TOLERANCING PER:		G.A.	
MATERIAL:		Acrylic	
FINISH:		USED ON	
NEXT ASSY:		APPLICATION	
DO NOT SCALE DRAWING			



2

1



B

A

B

A

2

1

**PROPRIETARY AND CONFIDENTIAL**  
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL: $\pm$ ANGULAR: MACH $\pm$ BEND $\pm$ TWO PLACE DECIMAL $\pm$ 0.01in THREE PLACE DECIMAL $\pm$ 0.005in		DRAWN	NAME	DATE	RSL Rover	
INTERPRET GEOMETRIC TOLERANCING PER:	G.A.	CHECKED	Aaron Burns	3/5/2016	TITLE:	
MATERIAL	Acrylic	ENG APPR.			USB House Right	
FINISH	USED ON	MFG APPR.			SIZE DWG. NO. <b>A</b> USBH5	
NEXT ASSY	APPLICATION	COMMENTS:			REV	
		DO NOT SCALE DRAWING			SCALE: 1:5 WEIGHT: SHEET 1 OF 1	

# Appendix H: Code

## Vehicle Visualization URDF Model

```
<robot name="rsl_roverzoe">
<link name="base_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://rsl_rover/urdf/RoughBody_corrected.stl"/>
    </geometry>
  </visual>
  <collision><!-- Test values for now -->
    <geometry>
      <box size="300 200 200" />
    </geometry>
  </collision>
</link>
<!-- Front Right Wheel -->
<joint name="body_to_fr_tire" type="fixed" >
  <parent link="base_link" />
  <child link="fr_tire" />
  <origin xyz="1.9431 -0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="fr_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>
<!-- Front Left Wheel -->
<joint name="body_to_fl_tire" type="fixed" >
  <parent link="base_link" />
  <child link="fl_tire" />
  <origin xyz="1.9431 0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="fl_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>
<!-- Center Right Wheel -->
<joint name="body_to_cr_tire" type="fixed" >
  <parent link="base_link" />
  <child link="cr_tire" />
  <origin xyz="0.3429 -0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="cr_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>
<!-- Center Left Wheel -->
<joint name="body_to_cl_tire" type="fixed" >
  <parent link="base_link" />
  <child link="cl_tire" />
  <origin xyz="0.3429 0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="cl_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>
```

```

<!-- Back Right Wheel -->
</link><joint name="body_to_br_tire" type="fixed" >
  <parent link="base_link" />
  <child link="br_tire" />
  <origin xyz="-0.3429 -0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="br_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>

<!-- Back Left Wheel -->
<joint name="body_to_bl_tire" type="fixed" >
  <parent link="base_link" />
  <child link="bl_tire" />
  <origin xyz="-0.3429 0.584 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="bl_tire">
  <visual>
    <geometry>
      <mesh filename="package://rsl_rover/urdf/Tire_corrected.stl"/>
    </geometry>
  </visual>
</link>

<!-- Front Laser -->
<joint name="body_to_front_laser" type="fixed" >
  <parent link="base_link" />
  <child link="front_laser" />
  <origin xyz="2.4 0 0.24" rpy="3.1416 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="front_laser">
  <visual>
    <geometry>
      <box size=".5 .5 .01" />
    </geometry>
  </visual>
</link>

<!-- Gimbal Base -->
<joint name="body_to_gimbal_base" type="fixed" >
  <parent link="base_link" />
  <child link="gimbal_base" />
  <!--<origin xyz=".978 0 1.892" rpy="0 0 0" />-->
  <origin xyz="1.1 0 1.892" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="gimbal_base">
  <visual>
    <geometry>
      <box size=".5 .5 .01" />
    </geometry>
  </visual>
</link>

<!-- Imu Link -->
<joint name="body_to_imu_link" type="fixed" >
  <parent link="base_link" />
  <child link="imu_link" />
  <origin xyz="0 0 0.5" rpy="0 0 0" />
  <axis xyz="0 1 0" />
</joint>
<link name="imu_link">
  <visual>
    <geometry>
      <box size=".05 .05 .05" />
    </geometry>
  </visual>
</link>

<!-- Base Footprint Link (base link shifted down to under the wheels) -->
<joint name="body_to_base_footprint" type="fixed" >
  <parent link="base_link" />
  <child link="base_footprint" />
  <origin xyz="0 0 -0.3429" rpy="0 0 0" />
  <axis xyz="0 0 0" />
</joint>
<link name="base_footprint">

```

```

</link>
<joint name="body_to_front_camera" type="fixed" >
  <parent link="base_link" />
  <child link="front_camera" />
  <origin xyz="1.0 0 1.0" rpy="3.14 0 0" />
  <axis xyz="0 0 0" />
</joint>
<link name="front_camera">
</link>
</robot>

```

## Cameras Launch File

```

<launch>
  <arg name="fps" default="10" />
  <arg name="width" default="432" />
  <arg name="height" default="240" />

  <node name="camera_web_server" pkg="web_video_server" type="web_video_server" output="
screen">
    <param name="port" value="8080" />
    <param name="address" value="10.0.0.111" />
    <param name="server_threads" value="1" />
    <param name="ros_threads" value="2" />
    <param name="quality" value="90" />
  </node>

  <node pkg="uvc_camera" type="uvc_camera_node" name="camerafront" output="screen">
    <remap from="/image_raw" to="/camerafront_raw" />
    <param name="device" value="/dev/camerafront" />
    <param name="camera_info_url" value="file:///home/ubuntu/cfg/cameracalib/
camerafront.yaml" />
    <param name="fps" value="$(arg fps)" />
    <param name="width" value="$(arg width)" />
    <param name="height" value="$(arg height)" />
    <param name="frame" value="front_camera" />
  </node>

  <node pkg="uvc_camera" type="uvc_camera_node" name="cameraright">
    <remap from="/image_raw" to="/cameraright_raw" />
    <param name="device" value="/dev/cameraright" />
    <param name="camera_info_url" value="file:///home/ubuntu/cfg/cameracalib/
cameraright.yaml" />
    <param name="fps" value="$(arg fps)" />
    <param name="width" value="$(arg width)" />
    <param name="height" value="$(arg height)" />
  </node>

  <node pkg="uvc_camera" type="uvc_camera_node" name="cameraleft">
    <remap from="/image_raw" to="/cameraleft_raw" />
    <param name="device" value="/dev/cameraleft" />
    <param name="camera_info_url" value="file:///home/ubuntu/cfg/cameracalib/
cameraleft.yaml" />
    <param name="fps" value="$(arg fps)" />
    <param name="width" value="$(arg width)" />
    <param name="height" value="$(arg height)" />
  </node>

  <node pkg="uvc_camera" type="uvc_camera_node" name="camerarear">
    <remap from="/image_raw" to="/camerarear_raw" />
    <param name="device" value="/dev/camerarear" />
    <param name="camera_info_url" value="file:///home/ubuntu/cfg/cameracalib/
camerarear.yaml" />
    <param name="fps" value="$(arg fps)" />
    <param name="width" value="$(arg width)" />
    <param name="height" value="$(arg height)" />
  </node>

  <!-- <node name="people_detect" pkg="opencv_apps" type="people_detect" args="image:=/
cameraright_raw"/> -->
</launch>

```

## Camera Calibration: Front

```

image_width: 640
image_height: 480

```

```

camera_name: camera
camera_matrix:
  rows: 3
  cols: 3
  data: [652.560454, 0.000000, 319.752367, 0.000000, 650.694801, 248.296020, 0.000000, 0.000000,
1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.040380, -0.112498, -0.005992, 0.003583, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [640.549927, 0.000000, 321.096783, 0.000000, 0.000000, 644.552368, 246.303601, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000]

```

## Camera Calibration: Left

```

image_width: 640
image_height: 480
camera_name: camera
camera_matrix:
  rows: 3
  cols: 3
  data: [680.338468, 0.000000, 308.172392, 0.000000, 680.430062, 268.088277, 0.000000, 0.000000,
1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.071290, 0.126918, 0.012577, -0.003966, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [677.564697, 0.000000, 305.707383, 0.000000, 0.000000, 674.015991, 271.888351, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000]

```

## Camera Calibration: Rear

```

image_width: 640
image_height: 480
camera_name: camera
camera_matrix:
  rows: 3
  cols: 3
  data: [652.885564, 0.000000, 294.983103, 0.000000, 655.716920, 239.362656, 0.000000, 0.000000,
1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.070639, 0.052269, 0.001334, -0.000681, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [642.366028, 0.000000, 293.619005, 0.000000, 0.000000, 648.596558, 239.218096, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000]

```

## Camera Calibration: Right

```

image_width: 640
image_height: 480
camera_name: camera
camera_matrix:
  rows: 3
  cols: 3
  data: [652.817375, 0.000000, 311.656932, 0.000000, 654.014227, 249.445159, 0.000000, 0.000000,
1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.076846, 0.091739, -0.001813, -0.003684, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [642.887573, 0.000000, 309.222547, 0.000000, 0.000000, 647.120850, 248.549064, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000]

```

## LIDAR Scan to Point Cloud

```

#include "ros/ros.h"
#include "ros/message.h"

#include "tf/transform_listener.h"
#include "sensor_msgs/PointCloud2.h"
#include "sensor_msgs/PointField.h"
#include "tf/message_filter.h"
#include "message_filters/subscriber.h"
#include "laser_geometry/laser_geometry.h"
#include "ros/console.h"

#include <vector>
#include <string>
class LaserScanToPointCloud{

public:

  ros::NodeHandle n_;
  laser_geometry::LaserProjection projector_;
  tf::TransformListener listener_;
  message_filters::Subscriber<sensor_msgs::LaserScan> laser_sub_;
  tf::MessageFilter<sensor_msgs::LaserScan> laser_notifier_;
  ros::Publisher scan_pub_;

  LaserScanToPointCloud(ros::NodeHandle n) :
    n_(n),
    laser_sub_(n_, "scanlms221", 10),
    laser_notifier_(laser_sub_, listener_, "base_link", 10)
  {
    laser_notifier_.registerCallback(
      boost::bind(&LaserScanToPointCloud::scanCallback, this, _1));
    laser_notifier_.setTolerance(ros::Duration(0.01));
    scan_pub_ = n_.advertise<sensor_msgs::PointCloud2>("/ass_cloud",1);
  }

  void scanCallback (const sensor_msgs::LaserScan::ConstPtr& scan_in)
  {
    sensor_msgs::PointCloud2 cloud;
    try
    {
      projector_.transformLaserScanToPointCloud(
        "base_link",*scan_in, cloud, listener_);
    }
    catch (tf::TransformException& e)
    {
      std::cout << e.what();
      return;
    }
    // our scan doesn't come with intensities
    // use the y value as the color value of the point
    int fields_length = cloud.fields.size();
    int y_offset = 0;
    int count = 0;
    for (int i =0; i < fields_length; i++) {
      if (cloud.fields[i].name == "y") {
        y_offset = cloud.fields[i].offset;
        count = cloud.fields[i].count;
        break;
      }
    }
  }

```

```

    }
    sensor_msgs::PointField p;
    p.name = "rgb";
    p.offset = y_offset;
    p.datatype = 6;
    p.count = count;
    cloud.fields.push_back(p);

    // Do something with cloud.
    scan_pub_.publish(cloud);
}
};

int main(int argc, char** argv)
{
    ros::init(argc, argv, "my_scan_to_cloud");
    ros::NodeHandle n;
    LaserScanToPointCloud lstopc(n);

    ros::spin();

    return 0;
}

```

## Vehicle State Information

```

#!/usr/bin/env python
import rospy
import numpy
import binascii
import roslib
import time
import struct
from serial import Serial
from array import array
from collections import namedtuple
from nav_msgs.msg import Odometry
from rsl_rover_msgs.msg import vehicle_state
from math import pi, tan

class RoverInterface:
    #_rx_len = 0
    #_rx_size = 26
    fmt = 'iffhcch????????hf'
    RoverDataKeys = [
        'time',
        'wheel_pos',
        'wheel_speed',
        'desired_throttle',
        'desired_gear',
        'actual_gear',
        'desired_steering',
        'actual_steering',
        'temp_warn',
        'voltage_warn',
        'estop',
        'A', 'B', 'C', 'D', 'E', 'Horn', 'F',
        'estop_code']

    def __init__(self, ser_in):
        self._rx_size = struct.calcsize(self.fmt)
        self.ser = ser_in
        self._rx_len = 0
        self.rx_buffer = bytearray(self._rx_size+1)
        self.rx_array_inx = 0
        self.RoverState = dict()

    def receiveData(self):
        ser = self.ser
        #global rx_len, rx_buffer, rx_array_inx, RoverDataKeys, RoverData
        #Translated from the c++ arduino Easy Transfer Library
        if self._rx_len == 0:
            if ser.inWaiting() >= 3:
                while ser.read() != '\x06':
                    if ser.inWaiting() < 3:
                        return False

            if ser.read() == '\x85':
                self._rx_len = ord(ser.read())
                if self._rx_len != self._rx_size:

```

```

        self._rx_len = 0
        return False

    if self._rx_len != 0:
        while ser.inWaiting() and self.rx_array_inx <= self._rx_len:
            try:
                self.rx_buffer[self.rx_array_inx] = ser.read()
                self.rx_array_inx += 1
            except Exception, err:
                rospy.logwarn('Error while reading after start bits',err)

    if (self._rx_len == (self.rx_array_inx-1)):
        calc_CS = self._rx_len
        for i in range(0,self._rx_len):
            calc_CS ^= self.rx_buffer[i]

        if calc_CS == self.rx_buffer[self.rx_array_inx-1]:
            try:
                self.decodeStruct(self.rx_buffer[0:-1])
            except Exception, err:
                rospy.logwarn(err)
                rospy.logwarn('Failed to Decode Packet')
                rospy.logwarn(binascii.b2a_hex(self.rx_buffer[0:-1]))
            self._rx_len = 0
            self.rx_array_inx = 0
            return True
        else:
            self._rx_len = 0
            self.rx_array_inx = 0
            return False

    def decodeStruct(self,data):
        #print ''.join('{:02x}'.format(x) for x in self.rx_buffer[0:-1])
        values = struct.unpack(self.fmt,data)
        self.RoverState = dict(zip(self.RoverDataKeys,values))

    def getRoverState(self):
        return self.RoverState

if __name__ == '__main__':
    ser = Serial(
        port='/dev/rover',
        baudrate=115200,
        timeout=0.1,
        dsrdtr=False,
    )
    rover = RoverInterface(ser)
    pub = rospy.Publisher('VehicleTach',Odometry,queue_size=2)
    pub_state = rospy.Publisher('VehicleState',vehicle_state,queue_size=2)
    rospy.init_node('RoverInterface')
    rate = rospy.Rate(100)

    while not rospy.is_shutdown():
        rover.receiveData()
        RS = rover.getRoverState()
        try:
            #Build up odometry message
            odom_msg = Odometry()
            odom_msg.header.stamp = rospy.Time.now()
            odom_msg.header.frame_id = "odom"
            odom_msg.child_frame_id = "base_link"

            wheel_speed_rpm = RS['wheel_speed']
            wheel_pos_r = RS['wheel_pos']
            steering = RS['actual_steering']
            #Wheel diameter is 25" or 0.635m
            #Min to sec divide by 30
            #Multiply by pi to get circumference
            wheel_speed = 0.635 * pi / 60 * wheel_speed_rpm #Convert from rpm to m/s
            wheel_pos = 0.635 * pi * wheel_pos_r
            odom_msg.twist.twist.linear.x = wheel_speed
            odom_msg.twist.twist.linear.y = 0
            #odom_msg.twist.twist.angular.z = wheel_speed*tan(pi/6*steering/1000.0)/1.97
            odom_msg.twist.covariance = numpy.diag([1e-2,1e-2,1e-2,1e-2,1e-2,1e-2]).flatten().tolist()

            pub.publish(odom_msg)
        except Exception, err:
            rospy.logwarn('Could Not Publish tach_odom');
            rospy.logwarn(err)

        try:
            #Build up vehicle state message
            vs_msg = vehicle_state()
            vs_msg.header.stamp = rospy.Time.now()
            vs_msg.wheel_pos = wheel_pos # Convert from Revs to M
            vs_msg.wheel_speed = wheel_speed #Convert from RPM to M/S

```



```

vs_msg.desired_throttle = RS['desired_throttle']
vs_msg.desired_gear = RS['desired_gear']
vs_msg.actual_gear = RS['actual_gear']
vs_msg.desired_steering = RS['desired_steering']
vs_msg.actual_steering = RS['actual_steering']
vs_msg.temp_warn = RS['temp_warn']
vs_msg.voltage_warn = RS['voltage_warn']
vs_msg.estop = RS['estop']
vs_msg.aux = [RS[x] for x in ['A', 'B', 'C', 'D', 'E', 'Horn', 'F']]
vs_msg.estop_code = RS['estop_code']
#vs_msg.engine_rpm = RS['engine_rpm']
pub_state.publish(vs_msg)

except Exception, err:
    rospy.logwarn('Could Not Publish rover_state');
    rospy.logwarn(err)

rate.sleep()

```

## GPS Configuration

```

#!/usr/bin/python
import serial
import time
import struct

ser = serial.Serial(
    #port='/dev/ttyUSB0',
    #port='/home/rover-dev/dev/ttyLMS',
    port='/dev/novatel',
    baudrate=115200)

print(ser.isOpen())

#ser.write('\x02\x00\x01\x00\x31\x15\x12')
#cmd = ['COM COM1,115200,N,8,1,N,OFF,ON']
cmd = ['UNLOGALL\r\n', \
'LOG com1 versiona once\r\n', \
'SBASCONTROL ENABLE ANY 0 NONE\r\n', \
'SBASCONTROL DISABLE\r\n', \
'LOG com1 GPGGA ontime 0.1\r\n', \
'LOG com1 GPVTG ontime 1\r\n', \
'LOG com1 GPCSV ontime 1\r\n', \
'LOG com1 GPCSA ontime 1\r\n', \
'LOG com1 GPGST ontime 1\r\n', \
'SAVECONFIG\r\n']

for c in cmd:
    print c
    ser.write(c)
    ser.flush()
    time.sleep(0.5)
    output = ''
    while ser.inWaiting() > 0:
        output += ser.read()
    print output

while 1:
    output = []
    if ser.inWaiting() > 0:
        print ser.readline()
#        print output.encode('hex')

ser.close()

```

## GPS Driver Launch

```

<launch>
  <node pkg="nmea_navsat_driver" type="nmea_serial_driver" name="gps_driver">
    <param name="port" value="/dev/novatel" />
    <param name="baud" value="115200" />
  </node>
</launch>

```

## Gimbal Driver

```

#!/usr/bin/env python
import sys
import socket
import rospy
import roslib
import time
import tf
import re
from math import pi
from rsl_rover_msgs.msg import vehicle_state

TCP_IP = '10.0.0.141'
TCP_PORT = 2000
BUFFER_SIZE = 1024
moving = 0
MOVEPAUSE = 0.01
COMMPAUSE = 0.01
COUNTSTORAD = 45837
OFFSET_RAD = pi/2
GOALMAXRAD.FWD = pi/5
GOALMINRAD.FWD = -pi/5
GOALRAD.FWD.STATIC = -pi/8
CALIBRATION.COUNTS = 1150

GOALMAXRAD.REV = pi/5-pi;
GOALMINRAD.REV = -pi/5-pi;
GOALRAD.REV.STATIC = -pi+pi/8

goal = 0#GOALMAXRAD.FWD

lastCMD_T = 0
reversing = False
sweeping = False

def movetorad(s, setpoint):
    goal = (setpoint+OFFSET_RAD)*COUNTSTORAD
    sendCommandWResp(s, 'GOAL='+str(goal+CALIBRATION.COUNTS))

def moveto(s, pos):
    sendCommandWResp(s, 'GOAL='+str(pos+CALIBRATION.COUNTS))
    moving = 1

def sendPitchTF(br, pitch_rad):
    #br.sendTransform((0,0,0),
    #                tf.transformations.quaternion_from_euler(0,-(pitch_rad-OFFSET_RAD),0),
    #                rospy.Time.now(),
    #                'gimbal_laser',
    #                'gimbal_base')

    br.sendTransform((0,0,0),
                    tf.transformations.quaternion_from_euler(0,-pitch_rad,0),
                    rospy.Time.now(),
                    'gimbal_laser',
                    'gimbal_base')

def getpos(s):
    data = sendCommandWResp(s, 'TP')
    data = float(bufferToInt(data))/float(COUNTSTORAD) - OFFSET_RAD
    return data

def getvel(s):
    data = sendCommandWResp(s, 'TV')
    data = float(bufferToInt(data))/float(COUNTSTORAD)
    return data

def sendCommandWResp(s,cmd):
    s.send(cmd+'\r')
    time.sleep(COMMPAUSE)
    data = s.recv(BUFFER_SIZE) #clear the buffer

def sendCommandRResp(s,cmd):
    s.send(cmd + '\r')
    time.sleep(COMMPAUSE)
    data = s.recv(BUFFER_SIZE)
    return data

def bufferToInt(data):
    data = re.findall(r'-?\d+',data.rstrip())
    data = int(data[0].rstrip())
    return data

def init():
    rospy.init_node('gimbal_driver')
    br = tf.TransformBroadcaster()
    try:
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect((TCP_IP,TCP_PORT))

```

```

except socket.error as msg:
    rospy.logwarn('PROBLEM CONNECTING',msg)
    sys.exit(1)
time.sleep(0.1)
sendCommandWOREsp(s, '')
rospy.loginfo('INIT COMPLETED')
return (s,br)

def vehicle_state_callback(data):
    global reversing, sweeping
    reversing = data.desired_gear == 'R'
    sweeping = data.aux[1]
    pass

if __name__ == '__main__':
    try:
        ret = init()
        s = ret[0]
        br = ret[1]
#        sendCommandWOREsp(s, 'GOAL=10000')
        rate = rospy.Rate(100)

        rover_sub = rospy.Subscriber('VehicleState', vehicle_state, vehicle_state_callback)

        while not rospy.is_shutdown():
            vel = getvel(s)
            pos = getpos(s)
            sendPitchTF(br, pos)
            if vel < 1000.0/COUNTSTORAD:
                moving = False
            else:
                moving = True

            if reversing:
                GOALMINRAD = GOALMINRAD.REV
                GOALMAXRAD = GOALMAXRAD.REV
                GOALRAD = GOALRAD.REV.STATIC
            else:
                GOALMINRAD = GOALMINRAD.FWD
                GOALMAXRAD = GOALMAXRAD.FWD
                GOALRAD = GOALRAD.FWD.STATIC

            if sweeping:
                if not moving and (time.time() - lastCMD.T > 0.2):
                    lastCMD.T = time.time()
                    if pos > (GOALMINRAD+GOALMAXRAD)/2: #GOALMAXRAD:
                        goal = GOALMINRAD
                        movetorad(s, goal)
                    elif pos <=(GOALMINRAD+GOALMAXRAD)/2:# GOALMINRAD:
                        goal = GOALMAXRAD
                        movetorad(s, goal)
                else:
                    movetorad(s, GOALRAD)

            rate.sleep()
        except rospy.ROSInterruptException:
            rospy.logwarn('ROSInterruptException Thrown')
            pass

        sendCommandWOREsp(s, 'GOAL=0')

```

## Environmental Sensor State Information

```

#!/usr/bin/env python
import re
import traceback
import sys
import rospy
import numpy
import binascii
import roslib
import time
import struct
from serial import Serial
from array import array
from collections import namedtuple
from rsl_rover_msgs.msg import env_data
from math import pi, tan
from MQSensor import MQ
import os

class RoverInterface:
    #_rx_len = 0
    #_rx_size = 26

```

```

fmt = 'i' * 11
RoverDataKeys = ['time', 'Temp1', 'Temp2', 'Humidity', 'Particulate']
MQNames = ['MQ-4', 'MQ-135', 'MQ-9', 'MQ-2', 'MQ-5', 'MQ-6', 'MQ-7', 'MQ-8']
RoverDataKeys = RoverDataKeys + MQNames

def __init__(self, ser_in):
    self._rx_size = struct.calcsize(self.fmt)
    self.ser = ser_in
    self._rx_len = 0
    self.rx_buffer = bytearray(self._rx_size+1)
    self.rx_array_inx = 0
    self.RoverState = dict()

def setupMQ(self):
    dbpath = os.path.join(os.path.dirname(MQ.__file__), 'mqdb.sqlite')
    self.MQSEN = dict()
    for name in self.MQNames:
        self.MQSEN[name] = MQ.MQ(dbpath, name)

def receiveData(self):
    ser = self.ser
    #global rx_len, rx_buffer, rx_array_inx, RoverDataKeys, RoverData
    #Translated from the c++ arduino Easy Transfer Library
    if self._rx_len == 0:
        if ser.inWaiting() >= 3:
            while ser.read() != '\x06':
                if ser.inWaiting() < 3:
                    return False

            if ser.read() == '\x85':
                self._rx_len = ord(ser.read())
                if self._rx_len != self._rx_size:
                    self._rx_len = 0
                    return False

    if self._rx_len != 0:
        while ser.inWaiting() and self.rx_array_inx <= self._rx_len:
            try:
                self.rx_buffer[self.rx_array_inx] = ser.read()
                self.rx_array_inx += 1
            except Exception, err:
                rospy.logwarn('Error while reading after start bits', err)

    if (self._rx_len == (self.rx_array_inx - 1)):
        calc_CS = self._rx_len
        for i in range(0, self._rx_len):
            calc_CS ^= self.rx_buffer[i]

        if calc_CS == self.rx_buffer[self.rx_array_inx - 1]:
            try:
                self.decodeStruct(self.rx_buffer[0:-1])
            except Exception, err:
                rospy.logwarn(err)
                rospy.logwarn('Failed to Decode Packet')
                rospy.logwarn(binascii.b2a_hex(self.rx_buffer[0:-1]))
                self._rx_len = 0
                self.rx_array_inx = 0
                return True
        else:
            self._rx_len = 0
            self.rx_array_inx = 0
            return False

    def decodeStruct(self, data):
        #print ''.join('{:02x}'.format(x) for x in self.rx_buffer[0:-1])
        values = struct.unpack(self.fmt, data)
        self.RoverState = dict(zip(self.RoverDataKeys, values))

    def getRoverState(self):
        return self.RoverState

if __name__ == '__main__':
    rospy.init_node('EnviroDataNode')
    serport = rospy.get_param('~port')
    rospy.loginfo(serport)

    ser = Serial(
        port=serport,
        baudrate=115200,
        timeout=0.1,
    )

    rover = RoverInterface(ser)
    rover.setupMQ()
    pub = rospy.Publisher('EnvData', env_data, queue_size=2)
    rate = rospy.Rate(5)

```

```

setupsamples = 0
setup_sample_list = []

while not rospy.is_shutdown():
    rover.receiveData()
    ES = rover.getRoverState()

    if setupsamples < 20:
        if len(ES) > 0:
            setup_sample_list.append(ES)
            setupsamples += 1

    if (setupsamples == 19):
        init_val = {}
        for name in ES.keys():
            if name.startswith('MQ-'):
                init_val[name] = sum(d[name] for d in setup_sample_list) / len(
                    setup_sample_list)
        for name in init_val.keys():
            rover.MQSEN[name].initialCalibration(init_val[name])
        print 'Done Calibrating'
    else:
        try:
            #Build up vehicle state message
            msg = env_data()
            msg.header.stamp = rospy.Time.now()
            msg.MQ4.raw = ES['MQ-4']
            msg.MQ135.raw = ES['MQ-135']
            msg.MQ2.raw = ES['MQ-2']
            msg.MQ5.raw = ES['MQ-5']
            msg.MQ8.raw = ES['MQ-8']
            msg.MQ9.raw = ES['MQ-9']
            msg.MQ7.raw = ES['MQ-7']
            msg.MQ6.raw = ES['MQ-6']

            try:
                for sen_name in rover.MQSEN.keys():
                    sen_name_mod = re.sub('[ -]', '', sen_name)

                    mq = msg._getattribute_(sen_name_mod)
                    mq._setattr_(sen_name, sen_name)

                    Y = rover.MQSEN[sen_name].processValue(ES[sen_name])
                    for sub in Y:
                        mq._setattr_(sub, Y[sub])

                    msg._setattr_(sen_name_mod, mq)

            except Exception, e:
                traceback.print_exc()
                rospy.logwarn('Error Generating Calibrated Values')

            msg.Temperature1 = ES['Temp1']
            msg.Temperature2 = ES['Temp2']
            msg.Humidity = ES['Humidity']
            msg.Particulate = ES['Particulate']
            pub.publish(msg)

        except Exception, err:
            rospy.logwarn('Could Not Publish Env Data');
            rospy.logwarn(err)

    rate.sleep()

```

## Sensor Calibration

```

import sqlite3 as lite
import math
import sys

class MQ:

    con = None

    def __init__(self, calibration_path, name):
        self.calpath = calibration_path
        self.name = name;

    try:
        con = lite.connect(self.calpath)

        cur = con.cursor()

```

```

query = 'SELECT sub.Name, Intercept, C, M, sen.R2, sen.Cair FROM Calibration AS cal INNER JOIN
        Substances AS sub ON
        cal.SubstanceID=sub.id INNER JOIN Sensors AS sen ON cal.SensorID=sen.id WHERE sen.name
        =' + self.name + ''';

result = cur.execute(query)
colname = [ d[0] for d in result.description ]
self.calibrations = [ dict(zip(colname, r)) for r in result.fetchall() ]
con.close()
except lite.Error, e:
    print "Error %s:" % e.args[0]
    sys.exit(0)

finally:
    if con:
        con.close()

def initialCalibration(self, raw_val):
    self.R2 = self.calibrations[0]['R2']
    Cair = self.calibrations[0]['Cair']
    Vm = raw_val * (5.0 / 1023.0)
    self.R0 = self.R2*(5.0 - Vm) / (Cair * Vm)
    return

def processValue(self, raw_val):
    Vm = raw_val * (5.0 / 1023.0)
    try:
        Rs = self.R2*(5.0-Vm)/Vm
    except ZeroDivisionError, e:
        print( 'Zero Division Error Caught' )
        Rs = 0
    Y = dict()
    for x in self.calibrations:
        C = float(x['C'])
        M = float(x['M'])
        name = str(x['Name'])
        try:
            Y[name] = math.pow(C*(Rs/self.R0), M)
        except ValueError, e:
            print 'Math Error Caught'
            Y[name] = 0
    return Y

if __name__ == '__main__':
    MQ1 = MQ('mqdb.sqlite', 'MQ-4')
    MQ1.initialCalibration(128)
    print MQ1.processValue(255)

```

## Laser Startup

```

<launch>
  <!-- Start the robot model which includes visual geometry and
        static transformations which define the robot's
        different frames -->
  <env name="ROSCONSOLEFORMAT" value="{${thread}} [${node}/${function}:${line}]: ${message}" />

  <param name="use_sim_time" value="false" />
  <include file="$(find rsl_rover)/launch/robot_state.launch" />

  <!-- Connect to the Vehicle Mega -->
  <include file="$(find rsl_rover)/launch/rover_interface.launch" />

  <!-- Startup all of the sensors: Lidar, GPS, IMU, Cameras -->
  <include file="$(find rsl_rover)/launch/RSL_LMS221.launch" />
  <include file="$(find rsl_rover)/launch/lms221_filter.launch" />
  <include file="$(find rsl_rover)/launch/RSL_LMS111.launch" />
  <include file="$(find rsl_rover)/launch/novatel.launch" />
  <include file="$(find rsl_rover)/launch/um7.launch" />
  <!--<include file="$(find rsl_rover)/launch/cameras.launch" /> -->

  <!-- Startup all of the mapping & localization nodes -->
  <include file="$(find rsl_rover)/launch/hector.launch" />
  <!--<include file="$(find rsl_rover)/launch/octomap.launch" />-->
  <include file="$(find rsl_rover)/launch/loc.launch" />

  <!-- Startup RVIZ visualization, marked as required so
        all nodes will exit if rviz is closed -->
  <include file="$(find rsl_rover)/launch/rover_rviz.launch" />

  <!-- Startup Web Services -->

```

```

<!-- <include file="$(find rsl_rover)/launch/serv.launch" /> -->
</launch>

```

## LIDAR LMS111 Launch

```

<launch>
  <arg name="host" default="10.0.0.140" />
  <node pkg="lms1xx" name="lms1xx" type="LMS1xx_node" output="screen">
    <param name="host" value="$(arg host)" />
    <param name="frame_id" value="front_laser" />
  </node>
  <include file="$(find rsl_rover)/launch/scan_filter.launch" />
</launch>

```

## LIDAR LMS221 Launch

```

<launch>
  <node pkg="rsl_rover" name="gimbal_driver" type="gimbal_driver.py" output="screen">
  </node>
  <node pkg="sicktoolbox-wrapper" name="lms221" type="sicklms">
    <param name="port" value="/dev/lms221" />
    <param name="baud" value="500000" />
    <param name="frame_id" value="gimbal_laser" />
    <remap from="scan" to="scanlms221" />
  </node>
</launch>

```

## LIDAR LMS221 Filter Launch

```

<launch>
  <node pkg="rsl_rover" type="periodic_snapshotter2" name="rover_periodic_snapshotter2" />

  <!-- Filter the point cloud with a Voxel Fliter Nodelet -->
  <node pkg="nodelet" type="nodelet" name="pcl_manager" args="manager" output="screen" />
  <!-- Run a VoxelGrid filter to clean NaNs and downsample the data -->
  <node pkg="nodelet" type="nodelet" name="voxel_grid" args="load pcl/VoxelGrid
    pcl_manager" output="screen">
    <remap from="~input" to="assembled_cloud" />
    <remap from="~output" to="filtered_cloud" />
    <roscpp command="load" file="$(find rsl_rover)/filter.config/voxel_config.yaml"
  </node>
  </node>
</launch>

```

## All Laser Launch

```

<launch>
  <include file="RSL_LMS221.launch" />
  <include file="RSL_LMS111.launch" />
  <node pkg="laser_assembler" name="laser_scan_assembler" type="laser_scan_assembler" >
    <param name="max_scans" type="int" value="400" />
    <param name="fixed_frame" type="string" value="base_link" />
  </node>
</launch>

```

## Hector Mapping Launch

```

<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="odom"/>
  <arg name="pub_map_odom_transform" default="false"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="4096"/>

```

```

<!-- <node pkg="topic_tools" type="relay" name="scan_relay_1" args="scanlms221 scan_total" />
<node pkg="topic_tools" type="relay" name="scan_relay_2" args="scan scan_total" /> -->

<node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">
  <param name="laser_max_dist" value="140.0" />

  <!-- Frame names -->
  <param name="map_frame" value="map" />
  <param name="base_frame" value="$(arg base_frame)" />
  <param name="odom_frame" value="$(arg odom_frame)" />

  <!-- Tf use -->
  <param name="use_tf_scan_transformation" value="true"/>
  <param name="use_tf_pose_start_estimate" value="false"/>
  <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

  <!-- Map size / start point -->
  <param name="map_resolution" value="0.20"/>
  <param name="map_size" value="$(arg map_size)"/>
  <param name="map_start_x" value="0.5"/>
  <param name="map_start_y" value="0.5" />
  <param name="map_multi_res_levels" value="3" />

  <!-- Map update parameters -->
  <param name="update_factor_free" value="0.4"/>
  <param name="update_factor_occupied" value="0.9" />
  <param name="map_update_distance_thresh" value="0.1"/>
  <param name="map_update_angle_thresh" value="0.02" />
  <param name="laser_z_min_value" value = "-1.0" />
  <param name="laser_z_max_value" value = "1.5" />

  <!-- Advertising config -->
  <param name="advertise_map_service" value="true"/>

  <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
  <param name="scan_topic" value="$(arg scan_topic)"/>

  <!-- Debug parameters -->
  <!--
  <param name="output_timing" value="false"/>
  <param name="pub_drawings" value="true"/>
  <param name="pub_debug_output" value="true"/>
  -->
  <param name="tf_map_scanmatch_transform_frame_name" value="$(arg
    tf_map_scanmatch_transform_frame_name)" />
</node>
</launch>

```

## Localization Launch

```

<!-- Launch file for ekf_localization_node -->

<!-- Layer 1 Localization: Odometry Frame -->
<launch>
  <node pkg="tf" type="static_transform_publisher" name="tach_odom123" args="0 0 0 0 0 1 odom
    tach_odom 20" />
  <node pkg="rsl_rover" type="imu_override_covariance.py" name="IOC" />
  <!-- <node pkg="rsl_rover" type="virt_yaw_sensor.py" name="VirtYaw" output="screen"/> -->

  <node pkg="imu_complementary_filter" type="complementary_filter_node" name="
    complementary_filter_node" >
    <remap from="imu/data_raw" to="imu/data_cov" />
    <remap from="imu/mag" to="imu/mag" />
    <remap from="imu/data" to="imu/data_filtered" />
    <param name="do_bias_estimation" value="true"/>
    <param name="do_adaptive_gain" value="true"/>
    <param name="use_mag" value="false"/>
    <param name="gain_acc" value="0.01"/>
    <param name="gain_mag" value="0.01"/>
  </node>

  <node pkg="robot_localization" type="ekf_localization_node" name="odom_localization"
    clear_params="true" output="screen">

    <param name="frequency" value="30"/>
    <param name="sensor_timeout" value="0.1"/>
    <param name="two_d_mode" value="true"/>
    <param name="map_frame" value="map"/>
    <param name="odom_frame" value="odom"/>
    <param name="base_link_frame" value="base_link"/>

```



```

<param name="world_frame" value="odom"/>
<param name="transform_time_offset" value="0.0"/>

<!-- The filter accepts an arbitrary number of inputs from each input message type (Odometry
, PoseStamped,
TwistStamped, Imu). To add a new one, simply append the next number in the sequence to
its base name,
e.g., odom0, odom1, twist0, twist1, imu0, imu1, imu2, etc. The value should be the
topic name. These
parameters obviously have no default values, and must be specified. -->
<param name="odom0" value="/VehicleTach"/>
<!-- <param name="odom1" value="/VirtYaw"/> -->
<param name="imu0" value="/imu/data_filtered"/>

<!-- Each sensor reading updates some or all of the filter's state. These options give you
greater control over
which values from each measurement /imu/data_filtered are fed to the filter. For example
, if you have an odometry message as input,
but only want to use its Z position value, then set the entire vector to false, except
for the third entry.
The order of the values is x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw,
ax, ay, az. Note that not some
message types lack certain variables. For example, a TwistWithCovarianceStamped message
has no pose information, so
the first six values would be meaningless in that case. Each vector defaults to all
false if unspecified, effectively
making this parameter required for each sensor. -->

<rosparam param="odom0_config">[false, false, false,
false, false, false,
true, true, false,
false, false, true,
false, false, false]</rosparam -->
<!-- <rosparam param="odom1_config">[false, false, false,
false, false, false,
false, false, false,
false, false, true,
false, false, false]</rosparam -->

<rosparam param="imu0_config">[false, false, false,
true, true, true,
false, false, false,
true, true, true,
false, false, false]</rosparam>

<!-- The best practice for including new sensors in robot_localization's state estimation
nodes is to pass in velocity
measurements and let the nodes integrate them. However, this isn't always feasible, and
so the state estimation
nodes support fusion of absolute measurements. If you have more than one sensor
providing absolute measurements,
however, you may run into problems if your covariances are not large enough, as the
sensors will inevitably
diverge from one another, causing the filter to jump back and forth rapidly. To combat
this situation, you can
either increase the covariances for the variables in question, or you can simply set
the sensor's differential
parameter to true. When differential mode is enabled, all absolute pose data is
converted to velocity data by
differentiating the absolute pose measurements. These velocities are then integrated as
usual. NOTE: this only
applies to sensors that provide absolute measurements, so setting differential to true
for twist measurements has
no effect.

Users should take care when setting this to true for orientation variables: if you have
only one source of
absolute orientation data, you should not set the differential parameter to true. This
is due to the fact that
integration of velocities leads to slowly increasing error in the absolute (pose)
variable. For position variables,
this is acceptable. For orientation variables, it can lead to trouble. Users should
make sure that all orientation
variables have at least one source of absolute measurement. -->
<param name="odom0_differential" value="false"/>
<!-- <param name="odom1_differential" value="false"/> -->
<param name="imu0_differential" value="false"/>

<!-- When the node starts, if this parameter is true, then the first measurement is treated
as a "zero point" for all
future measurements. While you can achieve the same effect with the differential
parameter, the key difference is
that the relative parameter doesn't cause the measurement to be converted to a velocity
before integrating it. If
you simply want your measurements to start at 0 for a given sensor, set this to true.
-->
<param name="odom0_relative" value="false"/>

```

```

<!-- <param name="odom1_relative" value="false"/> -->
<param name="imu0_relative" value="false"/>

<!-- If we're including accelerations in our state estimate, then we'll probably want to
remove any acceleration that
is due to gravity for each IMU. If you don't want to, then set this to false. Defaults
to false if unspecified. -->
<param name="imu0_remove_gravitational_acceleration" value="true"/>

<!-- If you're having trouble, try setting this to true, and then echo the /diagnostics_agg
topic to see
if the node is unhappy with any settings or data. -->
<param name="print_diagnostics" value="true"/>

<!-- ===== ADVANCED PARAMETERS ===== -->

<!-- Most users will be able to remove these parameters from the launch file without any
consequences. We recommend
that users do not set values for these parameters without having a thorough
understanding of
the parameters do. -->

<!-- By default, the subscription queue size for each message type is 1. If you wish to
increase that so as not
miss any messages (even if your frequency is set to a relatively small value), increase
these. -->
<param name="odom0_queue_size" value="2"/>
<!--<param name="odom1_queue_size" value="2"/> -->
<param name="imu0_queue_size" value="2"/>

<!-- If your data is subject to outliers, use these threshold settings, expressed as
Mahalanobis distances, to control
how far away from the current vehicle state a sensor measurement is permitted to be.
Each defaults to
numeric_limits<double>::max() if unspecified. -->
<!-- <param name="odom1_pose_rejection_threshold" value="5"/>
<param name="odom1_twist_rejection_threshold" value="1"/> -->
<!--<param name="imu0_pose_rejection_threshold" value="0.3"/>
<param name="imu0_twist_rejection_threshold" value="0.1"/>
<param name="imu0_linear_acceleration_rejection_threshold" value="0.1"/> -->

<!-- Debug settings. Not for the faint of heart. Outputs a ludicrous amount of information
to the file
specified by debug_out_file. I hope you like matrices! Defaults to false if unspecified
. -->
<param name="debug" value="false"/>
<!-- Defaults to "robot_localization_debug.txt" if unspecified. -->
<param name="debug_out_file" value="debug_ekf_localization.txt"/>

<!-- The process noise covariance matrix can be difficult to tune, and can vary for each
application, so it
is exposed as a configuration parameter. The values are ordered as x, y, z, roll, pitch
, yaw, vx, vy, vz,
vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below if unspecified. -->
<roscparam param="process_noise_covariance">[0.05, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0,
0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0,
0.015]</roscparam>

```

```

<!-- This represents the initial value for the state estimate error covariance matrix.
Setting a diagonal value (a
variance) to a large value will result in early measurements for that variable being
accepted quickly. Users should
take care not to use large values for variables that will not be measured directly. The
values are ordered as x, y,
z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below if unspecified. -->
<rosparam param="initial_estimate_covariance">[1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]</rosparam>

<!-- Placeholder for output topic remapping -->
<remap from="odometry/filtered" to="odometry/filtered_cont"/>

</node>

<node pkg="robot_localization" type="navsat_transform_node" name="navsat_transform_node">
<param name="magnetic_declination_radians" value="0.236"/>
<param name="yaw_offset" value="0"/>
<!-- <param name="yaw_offset" value="1.5707963"/>-->
<param name="publish_filtered_gps" value="true" />
<param name="broadcast_utm_transform" value="true" />

<remap from="/imu/data" to="/imu/data_filtered" />
<remap from="/gps/fix" to="/fix" />
<remap from="/odometry/filtered" to="/odometry/filtered_discont" />
</node>

<!-- Layer 2 Localization: Map Frame -->
<node pkg="robot_localization" type="ekf_localization_node" name="map_localization"
clear_params="true" output="screen">

<param name="frequency" value="30"/>
<param name="sensor_timeout" value="0.1"/>
<param name="two_d_mode" value="true"/>

<param name="map_frame" value="map"/>
<param name="odom_frame" value="odom"/>
<param name="base_link_frame" value="base_link"/>
<param name="world_frame" value="map"/>

<param name="transform_time_offset" value="0.0"/>

```

```

<param name="odom0" value="/odometry/filtered_cont"/>
<param name="pose0" value="/poseupdate"/>

<roscparam param="odom0_config">[false , false , false ,
    false , false , false ,
    true , true , false ,
    false , false , true ,
    true , true , false]</roscparam> -->

<roscparam param="pose0_config">[true , true , false ,
    false , false , true ,
    false , false , false ,
    false , false , false ,
    false , false , false]</roscparam> -->

<param name="odom0_differential" value="false"/>
<param name="odom0_relative" value="false"/>
<param name="pose0_differential" value="false"/>
<param name="pose0_relative" value="false"/>

<!-- If you're having trouble, try setting this to true, and then echo the /diagnostics_agg
topic to see
if the node is unhappy with any settings or data. -->
<param name="print_diagnostics" value="true"/>

<!-- ===== ADVANCED PARAMETERS ===== -->

<!-- Most users will be able to remove these parameters from the launch file without any
consequences. We recommend
that users do not set values for these parameters without having a thorough
understanding of
the parameters do. -->

<!-- By default, the subscription queue size for each message type is 1. If you wish to
increase that so as not
miss any messages (even if your frequency is set to a relatively small value), increase
these. -->
<param name="odom0_queue_size" value="1"/>
<param name="pose0_queue_size" value="1"/>

<!-- If your data is subject to outliers, use these threshold settings, expressed as
Mahalanobis distances, to control
how far away from the current vehicle state a sensor measurement is permitted to be.
Each defaults to
numeric_limits<double>::max() if unspecified. -->
<!-- <param name="odom0_pose_rejection_threshold" value="5"/>
<param name="odom0_twist_rejection_threshold" value="1"/> -->

<!-- Debug settings. Not for the faint of heart. Outputs a ludicrous amount of information
to the file
specified by debug_out_file. I hope you like matrices! Defaults to false if unspecified
. -->
<param name="debug" value="false"/>
<!-- Defaults to "robot_localization_debug.txt" if unspecified. -->
<param name="debug_out_file" value="debug_ekf_localization.txt"/>

<!-- The process noise covariance matrix can be difficult to tune, and can vary for each
application, so it
is exposed as a configuration parameter. The values are ordered as x, y, z, roll, pitch
, yaw, vx, vy, vz,
vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below if unspecified. -->
<roscparam param="process_noise_covariance">[0.05, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0.06, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0.025, 0,
    0, 0, 0, 0, 0, 0, 0, 0.025, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0.01, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0.01, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0.02, 0, 0, 0,
    0, 0, 0, 0, 0, 0.02, 0, 0, 0,

```



```

<node pkg="tf" type="static_transform_publisher" name="stationary_robot_transform_1" args
  ="0 0 0 0 0 0 /base_link /odom 100"/>

<!-- Assemble the laser scans into a point cloud, published periodically -->
<node pkg="laser_assembler" type="laser_scan_assembler" name="rover_laser_assembler" >
  <param name="fixed_frame" type="string" value="base_link" />
</node>
<node pkg="rsl_rover" type="periodic_snapshotter" name="rover_periodic_snapshotter" output
  ="screen">
  <param name="pub_duration" value="1.0" />
</node>

<!-- Filter the point cloud with a Voxel Fliter Nodelet -->
<node pkg="nodelet" type="nodelet" name="pcl_manager" args="manager" output="screen" />
<!-- Run a VoxelGrid filter to clean NaNs and downsample the data -->
<node pkg="nodelet" type="nodelet" name="voxel_grid" args="load pcl/VoxelGrid
  pcl_manager" output="screen">
  <remap from="~input" to="assembled_cloud" />
  <remap from="~output" to="filtered_cloud" />
  <roscpp command="load" file="$(find rsl_rover)/filter_config/voxel_config.yaml"
  />
</node>

<!-- Alternative Filter Method -->
<!--
<node pkg="laser_filters" type="scan_to_cloud_filter_chain" name="scan2cloud" >
  <roscpp command="load" file="$(find rsl_rover)/filter_config/laser_config.yaml"
  />
  <roscpp command="load" file="$(find rsl_rover)/filter_config/cloud_config.yaml"
  />
  <param name="high_fidelity" value="false" />
  <param name="target_frame" type="string" value="base_link" />
-->

-->

<node pkg="octomap_server" type="octomap_server_node" name="rover_octomap_server" output="
  screen" >
  <remap from="cloud_in" to="filtered_cloud" />
  <param name="resolution" value="0.3" />
  <param name="filter_ground" value="false" />
  <param name="latch" value="false" />
  <param name="base_frame_id" value="base_footprint" />
  <param name="ground_filter/distance" value="0.04" />
  <param name="ground_filter/angle" value="0.15" />
  <param name="ground_filter/plane_distance" value="0.07" />
</node>

<include file="$(find rsl_rover)/launch/robot_state.launch" />
<include file="$(find rsl_rover)/launch/rover_rviz.launch" />
</launch>

```

## Navigation Stack Move Base Launch

```

<launch>
  <!-- Optionally run something like AMCL -->

  <!-- run Navigation Stack Move Base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <roscpp file="$(find rsl_rover)/cfg/costmap_common_params.yaml" command="load" ns="
      global_costmap" />
    <roscpp file="$(find rsl_rover)/cfg/costmap_common_params.yaml" command="load" ns="
      local_costmap" />
    <roscpp file="$(find rsl_rover)/cfg/local_costmap_params.yaml" command="load" />
    <roscpp file="$(find rsl_rover)/cfg/global_costmap_params.yaml" command="load" />
    <roscpp file="$(find rsl_rover)/cfg/teb_local_planner_params.yaml" command="load" />
    <roscpp file="$(find rsl_rover)/cfg/costmap_converter_params.yaml" command="load" />
    <!-- Enable to activate costmap_conversion plugins -->
    <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS" />
    <!--param name="base_global_planner" value="straight_global_planner/StraightROS" /-->
    <param name="controller_frequency" value="5.0" />

    <remap from="map" to="map" />
    <remap from="move_base_simple/goal" to="/move_base_simple/goal"/> <!-- We have only a
      single robot now, that is controlled via goals -->
    <remap from="odom" to="odometry/filtered_discont" />
  </node>

  <node pkg="rsl_rover" name="cmd_to_ack" type="cmd_vel_to_ackermann_drive.py">
    <param name="wheelbase" value="2.5" />
    <param name="frame_id" value="odom" />
  </node>
</launch>

```

## OctoMap Launch

```
<launch>
  <!-- Assemble the laser scans into a point cloud, published periodically -->
  <!-- <node pkg="laser_assembler" type="laser_scan_assembler" name="rover_laser_assembler"
    >
      <param name="fixed_frame" type="string" value="base_link" />
      <remap from="scan" to="scanlms221" />
    </node> -->

  <!-- <node pkg="rsl_rover" type="periodic_snapshotter" name="rover_periodic_snapshotter"
    -->
      <param name="fixed_frame" type="string" value="/odom" />
      <param name="ignore_laser_skew" type="bool" value="false" />
      <param name="pub_duration" value="1.0" />
    </node>

  -->
  <!--
    <node pkg="rsl_rover" type="periodic_snapshotter2" name="rover_periodic_snapshotter2" />

    <node pkg="nodelet" type="nodelet" name="pcl_manager" args="manager" output="screen" />
    <node pkg="nodelet" type="nodelet" name="voxel_grid" args="load pcl/VoxelGrid
      pcl_manager" output="screen">
      <remap from="~input" to="assembled_cloud" />
      <remap from="~output" to="filtered_cloud" />
      <roscpp command="load" file="$(find rsl_rover)/filter_config/voxel_config.yaml"
        />
    </node>
    -->
  <!-- Alternative Filter Method -->
  <!--
    <node pkg="laser_filters" type="scan_to_cloud_filter_chain" name="scan2cloud" >
      <roscpp command="load" file="$(find rsl_rover)/filter_config/laser_config.yaml"
        />
      <roscpp command="load" file="$(find rsl_rover)/filter_config/cloud_config.yaml"
        />
      <param name="high_fidelity" value="false" />
      <param name="target_frame" type="string" value="base_link" />
    </node>

    -->
    <node pkg="octomap_server" type="octomap_server_node" name="rover_octomap_server" output="
      screen" >
      <remap from="cloud_in" to="filtered_cloud" />
      <param name="resolution" value="0.2" />
      <param name="filter_ground" value="false" />
      <param name="latch" value="false" />
      <param name="base_frame_id" value="base_footprint" />
      <param name="ground_filter/distance" value="0.04" />
      <param name="ground_filter/angle" value="0.15" />
      <param name="ground_filter/plane_distance" value="0.07" />
    </node>

  </launch>
```

## Vehicle State in Map Launch

```
<launch>
  <param name="robot_description" command="cat $(find rsl_rover)/urdf/rsl_roverzoe.urdf" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
</launch>
```

## Vehicle Interface Launch

```
<launch>
  <node pkg="rsl_rover" name="rover_interface" type="rover_decode.py" required="true" output
    ="screen">
  </node>
</launch>
```

## Vehicle RVIZ Launch

```
<launch>
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find rsl_rover)/rviz_cfg/rover_driving
    .rviz" required="true"/>
</launch>
```

## Vehicle Startup Launch

```
<launch>
  <node pkg="myahrs_driver" type="myahrs_driver" name="myahrs_driver">
    <param name="port" value="/dev/ttyACM0" />
    <param name="baud_rate" value="115200" />
  </node>
  <node pkg="nmea_navsat_driver" type="nmea_serial_driver" name="gps_driver">
    <param name="port" value="/dev/ttyS0" />
    <param name="baud_rate" value="9600" />
  </node>
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find myahrs_driver)/rviz_cfg/imu_test.rviz"
  />
</launch>
```

## Laser Scan Filter Launch

```
<launch>
  <node pkg="laser_filters" type="scan_to_scan_filter_chain" name="laser_filter">
    <roscpp command="load" file="$(find rsl_rover)/cfg/my_laser_config.yaml" />
    <remap from="scan" to="scan" />
  </node>
</launch>
```

## UM7 Orientation Sensor Launch

```
<launch>
  <node pkg="um7" type="um7_driver" name="um7_driver">
    <param name="port" value="/dev/um7" />
    <param name="covariance" value=".1 0 0 0 .1 0 0 0 .1" />
    <param name="zero_gyros" value="true" />
  </node>
</launch>
```

## Laser Configuration

```
scan_filter_chain :
- name: shadows
  type: laser_filters/ScanShadowsFilter
  params:
    min_angle: 5
    max_angle: 175
    neighbors: 10
    window: 1
- name: dark_shadows
  type: laser_filters/LaserScanIntensityFilter
  params:
    lower_threshold: 100
    upper_threshold: 10000
    disp_histogram: 0
```

## Web Server Launch



```

<launch>
  <!--<param name="use_sim_time" value="false" />-->
  <!--
  <node name="camera_web_server" pkg="web_video_server" type="web_video_server" output="screen">
    <param name="port" value="8080" />
    <param name="address" value="0.0.0.0" />
    <param name="server_threads" value="1" />
    <param name="ros_threads" value="2" />
  </node>
-->

  <arg name="port" default="9090" />
  <arg name="address" default="" />
  <arg name="ssl" default="false" />
  <arg name="certfile" default="" />
  <arg name="keyfile" default="" />
  <arg name="authenticate" default="false" />

  <group if="$(arg ssl)">
    <node name="rosbridge_websocket" pkg="rosbridge_server" type="rosbridge_websocket" output="
      screen">
      <param name="certfile" value="$(arg certfile)" />
      <param name="keyfile" value="$(arg keyfile)" />
      <param name="authenticate" value="$(arg authenticate)" />
      <param name="port" value="$(arg port)"/>
      <param name="address" value="$(arg address)"/>
    </node>
  </group>
  <group unless="$(arg ssl)">
    <node name="rosbridge_websocket" pkg="rosbridge_server" type="rosbridge_websocket" output="
      screen">
      <param name="authenticate" value="$(arg authenticate)" />
      <param name="port" value="$(arg port)"/>
      <param name="address" value="$(arg address)"/>
    </node>
  </group>

  <node name="rosapi" pkg="rosapi" type="rosapi_node" />
</launch>

```

## User Interface Control Center Main

```

'use strict';

function ROSCCConfig($routeProvider, localStorageServiceProvider) {
  $routeProvider.when('/', {
    templateUrl: 'app/control/control.html',
    controller: 'ControlController',
    controllerAs: 'vm'
  }).when('/settings', {
    templateUrl: 'app/settings/settings.html',
    controller: 'SettingsController',
    controllerAs: 'vm'
  }).otherwise({ redirectTo: '/' });

  localStorageServiceProvider.setPrefix('roscc');
}

angular.module('roscc', ['ngRoute', 'ui.bootstrap', 'LocalStorageModule']).config(ROSCCConfig);
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i <
  props.length; i++) { var descriptor = props[i];
  descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("
  value" in descriptor) descriptor.writable = true;
  Object.defineProperty(target, descriptor.key, descriptor); } } return function (Constructor,
  protoProps, staticProps) { if (protoProps)
  defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(
  Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw
  new TypeError("Cannot call a class as a function"); } }

var ros = undefined;
var isConnected = false;

var ControlController = function () {
  function ControlController($timeout, $interval, Settings, Domains) {
    var _this = this;

    _classCallCheck(this, ControlController);

    _createClass(this, ControlController);
  }
}

```

```

this.$timeout = $timeout;
this.Domains = Domains;

this.isConnected = isConnected;
this.setting = Settings.get();
this.maxConsoleEntries = 200;

// Load ROS connection and keep trying if it fails
this.newRosConnection();
$interval(function () {
  _this.newRosConnection();
}, 1000); // [ms]

this.resetData();
if (isConnected) {
  this.onConnected();
}
}

// The active domain shows further information in the center view

_createClass(ControlController, [{
  key: 'setActiveDomain',
  value: function setActiveDomain(domain) {
    this.activeDomain = domain;
  }
}, {
  key: 'getDomains',
  value: function getDomains() {
    var allData = this.data.topics.concat(this.data.services, this.data.nodes);
    var domains = this.Domains.getDomains(allData);
    if (!this.activeDomain) {
      // if no other domains are found, use Dashboard as the default
      this.setActiveDomain('Dashboard');
    }
    return domains;
  }
}, {
  key: 'getGlobalParameters',
  value: function getGlobalParameters() {
    return this.Domains.getGlobalParameters(this.data.parameters);
  }
}, {
  key: 'resetData',
  value: function resetData() {
    this.data = {
      rosout: [],
      topics: [],
      nodes: [],
      parameters: [],
      services: []
    };
  }
}, {
  key: 'newRosConnection',
  value: function newRosConnection() {
    var _this2 = this;

    if (isConnected || !this.setting) {
      return;
    }

    if (ros) {
      ros.close(); // Close old connection
      ros = false;
      return;
    }

    ros = new ROSLIB.Ros({ url: 'ws://' + this.setting.address + ':' + this.setting.port });

    ros.on('connection', function () {
      _this2.onConnected();
      isConnected = true;
      _this2.isConnected = isConnected;
    });

    ros.on('error', function () {
      isConnected = false;
      _this2.isConnected = isConnected;
    });

    ros.on('close', function () {
      isConnected = false;
      _this2.isConnected = isConnected;
    });
  }
}, {

```

```

key: 'onConnected',
value: function onConnected() {
    var _this3 = this;

    console.log(" Connected!");

    // wait a moment until ROS is loaded and initialized
    this.$timeout(function () {
        _this3.loadData();

        _this3.setConsole();
        if (_this3.setting.battery) {
            _this3.setBattery();
        }
    }, 1000); // [ms]
}

// Setup of console (in the right sidebar)
}, {
key: 'setConsole',
value: function setConsole() {
    var _this4 = this;

    var consoleTopic = new ROSLIB.Topic({
        ros: ros,
        name: this.setting.log,
        messageType: 'rosgraph_msgs/Log'
    });
    consoleTopic.subscribe(function (message) {
        var nameArray = message.name.split('/');
        var d = new Date(message.header.stamp.secs * 1E3 + message.header.stamp.nsecs * 1E-6);

        message.abbr = nameArray.length > 1 ? nameArray[1] : message.name;

        // String formatting of message time and date
        function addZero(i) {
            return i < 10 ? '0' + i : i;
        }
        message.dateString = addZero(d.getHours()) + ':' + addZero(d.getMinutes()) + ':' + addZero(
            d.getSeconds()) + '.' + addZero(d.getMilliseconds());
        _this4.data.rosout.unshift(message);

        if (_this4.data.rosout.length > _this4.maxConsoleEntries) {
            _this4.data.rosout.pop();
        }
    });
}

// Setup battery status
}, {
key: 'setBattery',
value: function setBattery() {
    var _this5 = this;

    var batteryTopic = new ROSLIB.Topic({
        ros: ros,
        name: this.setting.batteryTopic,
        messageType: 'std_msgs/Float32'
    });
    batteryTopic.subscribe(function (message) {
        _this5.batteryStatus = message.data;
    });
}

// Load structure, all data, parameters, topics, services, nodes...
}, {
key: 'loadData',
value: function loadData() {
    var _this6 = this;

    this.resetData();

    ros.getTopics(function (topics) {
        angular.forEach(topics, function (name) {
            _this6.data.topics.push({ name: name });
            console.log(" Getting topic: ", name);
            ros.getTopicType(name, function (type) {
                _this6.data.topics.findWhere(_this6.data.topics, { name: name }).type = type;
            });
        });
    });

    ros.getServices(function (services) {
        angular.forEach(services, function (name) {

```

```

        _this6.data.services.push({ name: name });

        ros.getServiceType(name, function (type) {
            ..findWhere(_this6.data.services, { name: name }).type = type;
        });
    });
});

ros.getParams(function (params) {
    angular.forEach(params, function (name) {
        var param = new ROSLIB.Param({ ros: ros, name: name });
        _this6.data.parameters.push({ name: name });

        param.get(function (value) {
            ..findWhere(_this6.data.parameters, { name: name }).value = value;
        });
    });
});

ros.getNodes(function (nodes) {
    angular.forEach(nodes, function (name) {
        _this6.data.nodes.push({ name: name });
    });
});
}
}));

return ControlController;
})();

angular.module('rosc').controller('ControlController', ControlController);
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i <
    props.length; i++) { var descriptor = props[i];
    descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("
    value" in descriptor)
    descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } } return
    function (Constructor,
    protoProps, staticProps) { if (protoProps)
    defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(
    Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw
    new TypeError("Cannot call a class as a function"); } }

var DomainsService = function () {
    function DomainsService() {
        _classCallCheck(this, DomainsService);
    }

    _createClass(DomainsService, [{
        key: 'filterAdvanced',
        value: function filterAdvanced(entry, advanced) {
            var entryArray = entry.split('/');
            if (advanced) {
                return true;
            }
            if (!entry || ..isEmpty(entryArray)) {
                return false;
            }
            return ..last(entryArray)[0] === ..last(entryArray)[0].toUpperCase();
        }
    }, {
        key: 'getDomains',
        value: function getDomains(array) {
            var result = [];
            angular.forEach(array, function (entry) {
                var nameArray = entry.name.split('/');
                if (nameArray.length > 1) {
                    result.push(nameArray[1]);
                }
            });
            return ..uniq(result).sort();
        }
    }, {
        key: 'getGlobalParameters',
        value: function getGlobalParameters(array) {
            var result = [];
            angular.forEach(array, function (entry) {
                var nameArray = entry.name.split('/');
                if (nameArray.length === 2) {
                    entry.abbr = ..last(nameArray);
                    result.push(entry);
                }
            });
            return result;
        }
    }
    ], {});
};

```

```

    }, {
      key: 'getDataForDomain',
      value: function getDataForDomain(array, domainName, advanced) {
        var _this = this;

        var result = [];

        angular.forEach(array, function (entry) {
          var nameArray = entry.name.split('/');
          if (nameArray.length > 1 && nameArray[1] === domainName && _this.filterAdvanced(entry.name
            , advanced)) {
            entry.abbr = nameArray.slice(2).join(' ');
            result.push(entry);
          }
        });
        return result;
      }
    }
  ]]);

  return DomainsService;
}();

// Filter advanced topics, services, parameters by checking the beginning capital letter
angular.module('roscc').service('Domains', DomainsService);
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i <
  props.length; i++) { var descriptor = props[i];
  descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value
  " in descriptor)
  descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor);
} } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(
  Constructor.prototype, protoProps); if (staticProps)
  defineProperties(Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw
  new TypeError("Cannot call a class as a function"); } }

var QuaternionsService = function () {
  function QuaternionsService() {
    _classCallCheck(this, QuaternionsService);
  }

  _createClass(QuaternionsService, [{
    key: 'getRoll',
    value: function getRoll(q) {
      if (!q) {
        return '';
      }
      var rad = Math.atan2(2 * (q.w * q.x + q.y * q.z), 1 - 2 * (q.x * q.x + q.y * q.y));
      return 180 / Math.PI * rad;
    }
  }, {
    key: 'getPitch',
    value: function getPitch(q) {
      if (!q) {
        return '';
      }
      var rad = Math.asin(2 * (q.w * q.y - q.z * q.x));
      return 180 / Math.PI * rad;
    }
  }, {
    key: 'getYaw',
    value: function getYaw(q) {
      if (!q) {
        return '';
      }
      var rad = Math.atan2(2 * (q.w * q.z + q.x * q.y), 1 - 2 * (q.y * q.y + q.z * q.z));
      return 180 / Math.PI * rad;
    }
  }, {
    key: 'getInit',
    value: function getInit() {
      return { w: 1, x: 0, y: 0, z: 0 };
    }
  }
  ]]);

  return QuaternionsService;
}();

// Quaternions to Euler angles converter
angular.module('roscc').service('Quaternions', QuaternionsService);
'use strict';

```

```

function NavbarDirective($location) {
  return {
    templateUrl: 'app/navbar/navbar.html',
    controllerAs: 'vm',
    controller: function controller() {
      this.isPath = isPath;

      function isPath(path) {
        return $location.path() === path;
      }
    }
  };
}

angular.module('rosc').directive('ccNavbar', NavbarDirective);
'use strict';

function ParamaterDirective() {
  return {
    scope: { parameter: '=' },
    templateUrl: 'app/parameters/parameters.html',
    controllerAs: 'vm',
    controller: function controller($scope) {
      var param = new ROSLIB.Param({ ros: ros, name: $scope.parameter.name });

      this.parameter = $scope.parameter;
      this.setValue = setValue;

      function setValue(value) {
        param.set(value);
      }
    }
  };
}

angular.module('rosc').directive('ccParameter', ParamaterDirective);
'use strict';

function serviceDirective() {
  return {
    scope: { service: '=' },
    template: '<ng-include src="' + vm.fileName + '></ng-include>',
    controllerAs: 'vm',
    controller: function controller($scope, $timeout, $http) {
      var _this = this;

      var path = 'app/services/';

      this.service = $scope.service;
      this.callService = callService;
      this.fileName = path + 'default.html';

      // Check if file exists
      $scope.$watch('service.type', function () {
        if (!$scope.service.type) {
          return;
        }
        var fileName = path + $scope.service.type + '.html';

        _this.service = $scope.service;
        $http.get(fileName).then(function (result) {
          if (result.data) {
            _this.fileName = fileName;
          }
        });
      });
    }
  };

  function callService(input, isJSON) {
    var _this2 = this;

    var data = isJSON ? angular.fromJson(input) : input;
    var service = new ROSLIB.Service({
      ros: ros,
      name: this.service.name,
      serviceType: this.service.type
    });
    var request = new ROSLIB.ServiceRequest(data);
    service.callService(request, function (result) {
      $timeout(function () {
        _this2.result = result;
      });
    });
  }
}
};
}
}

```

```

angular.module('rosc').directive('ccService', serviceDirective);
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i <
  props.length; i++) { var descriptor = props[i];
  descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value"
  in descriptor)
  descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor);
  } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(
  Constructor.prototype, protoProps); if (staticProps)
  defineProperties(Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw
  new TypeError("Cannot call a class as a function"); } }

var SettingsController = function () {
  function SettingsController(localStorageService, Settings) {
    _classCallCheck(this, SettingsController);

    this.Settings = Settings;

    this.settings = Settings.getSettings() || [Settings.getDefaultSetting()];
    this.index = Settings.getIndex();

    if (!this.index || this.index > this.settings.length) {
      this.index = '0';
    }
  }

  _createClass(SettingsController, [{
    key: 'save',
    value: function save() {
      this.Settings.save(this.settings, this.index);
    }
  }, {
    key: 'add',
    value: function add() {
      this.settings.push(this.Settings.getDefaultSetting()); // Clone object
      this.index = String(this.settings.length - 1);
      this.save();
    }
  }, {
    key: 'remove',
    value: function remove() {
      this.settings.splice(this.index, 1);
      this.index = '0';

      if (!this.settings.length) {
        this.add();
      }
      this.save();
    }
  }
  ]
  });

  return SettingsController;
}();

angular.module('rosc').controller('SettingsController', SettingsController);
'use strict';

var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i <
  props.length; i++) { var descriptor = props[i];
  descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("
  value" in descriptor)
  descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor);
  } } return function (Constructor, protoProps, staticProps) { if (protoProps) defineProperties(
  Constructor.prototype, protoProps); if (staticProps)
  defineProperties(Constructor, staticProps); return Constructor; }; }();

function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw
  new TypeError("Cannot call a class as a function"); } }

var SettingsService = function () {
  function SettingsService($location, localStorageService) {
    _classCallCheck(this, SettingsService);

    this.$location = $location;
    this.localStorageService = localStorageService;
  }

  _createClass(SettingsService, [{
    key: 'load',
    value: function load() {
      this.index = this.localStorageService.get('selectedSettingIndex');
      this.settings = this.localStorageService.get('settings');
      if (this.settings && this.index) {
        this.setting = this.settings[this.index];
      }
    }
  }
  ]
  );
}

```

```

    }

    // If there are no saved settings, redirect to /settings for first setting input
    if (!this.setting) {
        this.$location.path('/settings').replace();
    }
}
}, {
    key: 'save',
    value: function save(newSettings, newIndex) {
        this.settings = newSettings;
        this.index = newIndex;
        this.localStorageService.set('selectedSettingIndex', newIndex);
        this.localStorageService.set('settings', newSettings);
    }
}, {
    key: 'get',
    value: function get() {
        if (!this.setting) {
            this.load();
        }

        return this.setting;
    }
}, {
    key: 'getIndex',
    value: function getIndex() {
        if (!this.setting) {
            this.load();
        }

        return this.index;
    }
}, {
    key: 'getSettings',
    value: function getSettings() {
        if (!this.setting) {
            this.load();
        }

        return this.settings;
    }
}, {
    key: 'getDefaultSetting',
    value: function getDefaultSetting() {
        return {
            name: 'New Setting',
            address: location.hostname,
            port: 9090,
            log: '/rosout',
            imagePreview: { port: 0, quality: 70, width: 640, height: 480 },
            battery: true,
            batteryTopic: '',
            advanced: false
        };
    }
}
}));

return SettingsService;
})();

angular.module('rosc').service('Settings', SettingsService);
'use strict';

function topicDirective() {
    return {
        scope: { topic: '=' },
        template: '<ng-include src="' + vm.fileName + '></ng-include >',
        controllerAs: 'vm',
        controller: function controller($scope, $timeout, $http, Settings, Quaternions) {

            var _this = this;
            console.log("Creating new topic for name: ", $scope.topic.name, "; type: ", $scope.topic.type);

            var roslibTopic = new ROSLIB.Topic({
                ros: ros,
                name: $scope.topic.name,
                messageType: $scope.topic.type,
                queue_size: 1
            });
            var path = 'app/topics/';

            this.topic = $scope.topic;
            this.toggleSubscription = toggleSubscription;
            this.publishMessage = publishMessage;
            this.isSubscribing = false;

```



```

this.setting = Settings.get();
this.Quaternions = Quaternions;
this.fileName = path + 'default.html';

// Check if file exists
$scope.$watch('topic.type', function () {
  if (!$scope.topic.type) {
    return;
  }
  var fileName = path + $scope.topic.type + '.html';
  _this.topic = $scope.topic;
  $http.get(fileName).then(function (result) {
    if (result.data) {
      _this.fileName = fileName;
    }
  });
});

function toggleSubscription(data) {
  var _this2 = this;
  if (!data) {
    console.log("ROSLIBTOPIC: ", roslibTopic);
    roslibTopic.subscribe(function (message) {
      $timeout(function () {
        // get the incoming message for the given topic
        console.log(message);
        _this2.message = message;
      });
    });
  } else {
    roslibTopic.unsubscribe();
  }
  this.isSubscribing = !data;
}

function publishMessage(input, isJSON) {
  var data = isJSON ? angular.fromJson(input) : input;
  var message = new ROSLIB.Message(data);
  roslibTopic.publish(message);
}
}
}
};

angular.module('rosc').directive('ccTopic', topicDirective);

/**
 * Controller for the main dashboard
 *
 * This controller is different than the default topic controller provided by the Ros Control
 * Center
 * Here, we want to mix together data from multiple topics in one display.
 * In order to do this we need to subscribe to each topic and make their data available in a way
 * that they don't overwrite eachother
 *
 * To make the data available, we create a dictionary called *message*.
 * We then break each topics name and type into keys that are used to create a nested dictionary
 * structures.
 * For example, the topic that contains the vehicle state information is:
 * - name: /VehicleState
 * - type: /rsl_rover_msgs/vehicle_state
 *
 * The data for that topic will then live at:
 * - messages.VehicleState.rsl_rover_msgs.vehicle_state
 *
 * The data itself comes through as a JSON object which is then converted into a dictionary
 * So to get the wheel_speed of the rover we access:
 * - messages.VehicleState.rsl_rover_msgs.vehicle_state.wheel_speed
 */
function dashboardDirective() {
  return {
    scope: { topic: '=' },
    template: '<ng-include src="\vm.fileName"></ng-include>',
    controllerAs: 'vm',
    controller: function controller($scope, $timeout, $http, Settings, Quaternions) {
      var _this = this;

      // given a topic name and type, we create a nested dictionary structure
      // each string before or after a '/' becomes a new key to an empty dictionary
      this.MessageToDict = function(name, type) {
        console.log(name, type);
        var messages = {};
        var sub_message = messages;
        var name_splice = name.split("/");
        for (var i = 1; i < name_splice.length; i++) {
          sub_message[name_splice[i]] = {};
        }
      };
    }
  };
}

```

```

    sub_message = sub_message[name_splice[i]];
  }
  var type_splice = type.split("/");
  for (var i = 0; i < type_splice.length; i++) {
    sub_message[type_splice[i]] = {};
    sub_message = sub_message[type_splice[i]];
  }
  return messages;
}

// general topics that we want to visualize
this.topics = [
  {"name": "/EnvData/curly", "type": "rsl_rover_msgs/env_data", "throttle":200}, //
    sensor_box 1
  {"name": "/EnvData/moe", "type": "rsl_rover_msgs/env_data", "throttle":200}, //
    sensor_box 2
  {"name": "/EnvData/larry", "type": "rsl_rover_msgs/env_data", "throttle":200}, //
    sensor_box 3
  {"name": "/VehicleState", "type": "rsl_rover_msgs/vehicle_state", "throttle":100}, //
    vehicle state information
];

// the gas sensor topics are special, so we need to deal with them seperately
// we want to monitor them as a group and aggregate their values
// but we only want to use certain sensors for certain gases
this.gasTopics = {
  CO: {topics:[this.topics[0], this.topics[1], this.topics[2]], sensors:["MQ7", "MQ9",
    "MQ8"], throttle:200},
  CO2: {topics:[this.topics[0], this.topics[1], this.topics[2]], sensors:["MQ7", "MQ9",
    "MQ8"], throttle:200},
  Propane: {topics:[this.topics[0], this.topics[1], this.topics[2]], sensors:["MQ2", "MQ5",
    "MQ6", "MQ9"], throttle:200},
  Methane: {topics:[this.topics[0], this.topics[1], this.topics[2]], sensors:["MQ4"],
    throttle:200}
}
this.roslibTopics = {}
this.messages = {};

// build a ROSLIB Topic for each topic in the list
// and construct the holder for all the different message types
for (var topic in this.topics) {
  this.roslibTopics[_this.topics[topic].name] = new ROSLIB.Topic({
    ros: ros,
    name: _this.topics[topic].name,
    messageType: _this.topics[topic].type,
    throttle: _this.topics[topic].throttle,
    queue_size: 0
  });

  angular.merge(this.messages, this.MessageToDict(this.topics[topic].name, this.topics[topic].type));
}
console.log(this.messages);
var path = 'app/topics/';

this.topic = $scope.topic;
this.isSubscribing = false;
this.setting = Settings.get();
this.Quaternions = Quaternions;
this.fileName = path + 'default.html';

// Check if file exists
$scope.$watch('topic.type', function () {
  var fileName = path + "dashboard/dashboard2.html";
  _this.topic = $scope.topic;
  $http.get(fileName).then(function (result) {
    if (result.data) {
      _this.fileName = fileName;
    }
  });
});

this.roslibTopics['/EnvData/curly'].subscribe(function(message) {
  $timeout(function() {
    _this.messages['EnvData']['curly']['rsl_rover_msgs']['env_data'] = message;
  }, 1000);
});

this.roslibTopics['/EnvData/moe'].subscribe(function(message) {
  $timeout(function() {
    _this.messages['EnvData']['moe']['rsl_rover_msgs']['env_data'] = message;
  }, 1000);
});

this.roslibTopics['/EnvData/larry'].subscribe(function(message) {
  $timeout(function() {
    _this.messages['EnvData']['larry']['rsl_rover_msgs']['env_data'] = message;
  }, 1000);
});

```

```

});

this.roslibTopics['/VehicleState'].subscribe(function(message) {
  $timeout(function() {
    _this.messages['VehicleState']['rsl_rover_msgs']['vehicle_state'] = message;
  }, 1000);
});

/*for (topic in this.roslibTopics) {
  var t = this.roslibTopics[topic];
  console.log("Subscribing to ", t.name);
  //subscribe to topic and store messages in appropriate place
  t.subscribe(function(message) {
    $timeout(function () {
      var name_splice = t.name.split("/");
      var accessor;
      accessor = _this.messages[name_splice[1]];
      if(name_splice.length > 2) {
        console.log("CHECKING NAME SPLICE: ", name_splice[2]);
        accessor = accessor[name_splice[2]];
      }
      var type_splice = t.messageType.split("/");
      //console.log(name_splice, type_splice);
      //console.log(message);
      accessor[type_splice[0]][type_splice[1]]=message;
      console.log(accessor);
    }, 1000);
  });
}*/
}
};
angular.module('roscc').directive('dashTopic', dashboardDirective);

/**
 * Controller to initialize the a LIDAR view
 *
 * This function will use the ROSLIB and ROS3DJS libraries to render a live point cloud on a page.
 * It requires four different parts:
 * - ROS connection
 * - TF Client
 * - URDF model
 * - PointCloud
 *
 * The TF client is what makes everything come together. It does all of the translations to make
 * sure that the URDF model and the PointCloud
 * are rendered in the same scene.
 * We also use a SceneNode to have a little more control over the initialization.
 * The default viewer object makes some assumptions that we did not want to abide by.
 */
function angularLidarViz(){
  return {
    controller: function controller($scope, $timeout, $http, Settings, Quaternions) {
      $scope.init = function(height, divID) {
        /**
         * Setup all visualization elements when the page is loaded.
         */
        // Connect to ROS.
        this.settings = Settings.get();
        var _this = this;
        var ros = new ROSLIB.Ros({
          url : "ws://" + _this.settings.address + ":" + _this.settings.port
        });

        // Create the main viewer.
        var width = $("#lidar_viz").width();
        var viewer = new ROS3D.Viewer({
          divID : divID,
          width : width,
          height : height,
          antialias : false
        });

        // Add a grid.
        viewer.addObject(
          new ROS3D.Grid({
            cellSize: 0.5,
            num_cells: 100
          })
        );

        // Setup a client to listen to TFs.
        // Base_link will redner everything in relation to the base of the rover
        var tf_base = new ROSLIB.TFClient({
          ros : ros,
          angularThres : 0.01,

```

```

    transThres : 0.01,
    rate : 5.0,
    fixedFrame : '/base_link'
  });

  // setup a TF client for the world
  // this will render something with relation to the general WORLD that the rover is in
  var tf_cloud = new ROSLIB.TFClient({
    ros : ros,
    angularThres : 0.01,
    transThres : 0.01,
    rate : 5.0,
    fixedFrame : '/map'
  });

  // we want our scene to be focused around the WORLD in which the rover is in
  // for our purposes we want the tfClient and the frameId to reference the same topic
  var urdfScene = new ROS3D.SceneNode({
    tfClient : tf_cloud,
    frameId : '/map',
  });

  // add the scene to the viewer object
  viewer.scene.add(urdfScene);

  // create a new pointcloud object
  // our pointcloud is rendered via the /ass_cloud topic (short for /assembled_cloud)
  // we use the tf-base TF client in order to render the point cloud in relation to the rover
  var pointcloud = new ROS3D.PointCloud2({
    ros: ros,
    topic: "/ass_cloud",
    tfClient: tf_base,
    rootObject: urdfScene,
    size: 0.7,
    max_pts: 75000 //save up to 75000 points in the scene at any given time
  });

  // Setup the URDF client.
  // we use the TF Base client here too in order to render the vehicles position relative to
  // itself
  // NOTE: the URDF model is stored locally at /urdf
  // if the model ever updates, we need to update it here too
  var urdfClient = new ROS3D.UrdfClient({
    ros : ros,
    tfClient : tf_base,
    path : 'http://localhost:8000/urdf/',
    rootObject : urdfScene,
    loader : ROS3D.COLLADALOADER2
  });
}
}
}
angular.module('rosc').directive('lidarViz', angularLidarViz);

```

## User Interface Speed Chart

```

'use strict';
angular.module("").requires.push('highcharts-ng');

var ctrl = angular.module('rosc').controller("speedChartController", function ($scope, $timeout,
  $parse){
  console.log("Scope: ", $scope);
  $scope.speedChartConfig = {
    options: {
      chart: {
        type: 'gauge',
      },
      // the value axis
      yAxis: {
        min: 0,
        max: 10,

        minorTickInterval: 'auto',
        minorTickWidth: 1,
        minorTickLength: 10,
        minorTickPosition: 'inside',
        minorTickColor: '#666',

        tickPixelInterval: 30,
        tickWidth: 2,

```

```

    tickPosition: 'inside',
    tickLength: 10,
    tickColor: '#666',
    labels: {
      step: 2,
      rotation: 'auto'
    },
    title: {
      text: 'm/s'
    },
    plotBands: [{
      from: 0,
      to: 5,
      color: '#55BF3B' // green
    }, {
      from: 5,
      to: 7,
      color: '#DDDF0D' // yellow
    }, {
      from: 7,
      to: 10,
      color: '#DF5353' // red
    }
  ]
},
pane: {
  startAngle: -150,
  endAngle: 150,
  background: [{
    backgroundColor: {
      linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
      stops: [
        [0, '#FFF'],
        [1, '#333']
      ]
    },
    borderWidth: 0,
    outerRadius: '109%'
  }, {
    backgroundColor: {
      linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
      stops: [
        [0, '#333'],
        [1, '#FFF']
      ]
    },
    borderWidth: 1,
    outerRadius: '107%'
  }, {
    // default background
  }, {
    backgroundColor: '#DDD',
    borderWidth: 0,
    outerRadius: '105%',
    innerRadius: '103%'
  }
  ]
},
}, //end options
title: {
  text: "Speed (m/s)"
},
series: [{
  name: 'speed',
  data: [0],
  tooltip: {
    valueSuffix: ' m/s',
  },
  dataLabels: {
    format: "{y:.1f}"
  }
}],
plotOptions: {
  series: {
    marker: {
      enabled: false
    }
  },
  dataLabels: {
    format: "{y:.2f}"
  }
},
func: function(chart) {
  $timeout(function() {
    chart.reflow();
  }, 10);
},
useHighStock: true,

```

```

    });

    $scope.init = function(topic_name, chart_height) {
        $scope.topicName = topic_name;
        if (chart_height != undefined) {
            $scope.speedChartConfig.options.chart.height = chart_height;
        }
    }

    if ($scope.chart_height){
        $scope.speedChartConfig.options.chart.height = $scope.chart_height;
    }

    function deref(obj, s) {
        var i = 0;
        if (!s) {
            return undefined;
        }
        s = s.split('.');
        while (i < s.length) {
            obj = obj[s[i]];
            if (obj === undefined)
                return obj;
            i = i + 1;
        }
        return obj;
    }

    function getTopicName() {
        return $scope.topicName;
    }

    // watch the message and update the chart whenever the value updates
    var topicName = $scope.topicName;
    $scope.$watch(function($scope) {
        var val = deref($scope, $scope.topicName);
        return val;
    }, function(val){
        if (val){
            $scope.speedChartConfig.series[0].data[0] = Math.abs(val);
        }
    }, false);
    /*var _scope = $scope;
    $scope.$watch(model, function(newValue, oldValue){
        console.log(newValue, oldValue, model);
        if (newValue){
            console.log('Updating value!')
            _scope.speedChartConfig.series[0].data[0] = Math.abs(newValue);
        }
    })*/
});

```

## User Interface Gas Sensor Charts

```

'use strict';
angular.module("roscc").requires.push('highcharts-ng');

var ctrl = angular.module('roscc').controller("gasChart", function ($scope, $timeout){
    $scope.gasChart = {
        options: {
            "chart": {
                "type": "solidgauge",
            },
            exporting: {
                "enabled": false
            },
            "pane": {
                "center": [
                    "50%",
                    "85%"
                ],
                "size": "100%",
                "startAngle": "-90",
                "endAngle": "90",
                "background": {
                    "backgroundColor": "#EEE",
                    "innerRadius": "60%",
                    "outerRadius": "100%",
                    "shape": "arc"
                }
            },
            "tooltip": {
                "enabled": false
            },
        },
    },
});

```

```

    "yAxis": {
      "stops": [
        [
          0.1,
          "#55BF3B"
        ],
        [
          0.5,
          "#DDDF0D"
        ],
        [
          0.75,
          "#DF5353"
        ]
      ],
      "min": 0,
      "max": 100,
      "lineWidth": 0,
      "minorTickInterval": null,
      "tickPixelInterval": 400,
      "tickWidth": 0,
      "title": {
        "margin": 0,
      },
      "labels": {
        "y": 10
      },
      "showFirstLabel": false,
      "showLastLabel": false,
    },
    "title": {
      "text": "Gas",
      "margin": 0
    }
  }, //end options
  "plotOptions": {
    "solidgauge": {
      "dataLabels": {
        "y": 10,
        "borderWidth": 0,
        "useHTML": true
      }
    }
  },
  'series': [{
    'name': 'gas',
    'data': [0],
    'dataLabels': {
      'format': '<div style="text-align:center"><span style="font-size:8px;color:' +
        ((Highcharts.theme && Highcharts.theme.contrastTextColor) || 'black') + "'>{y}
        .1f</span><br/>'
    }
  }],
  func: function(chart) {
    $timeout(function() {
      chart.reflow();
    }, 100);
  },
  useHighStock: true
};
$scope.init = function(gas_name, chart_height, chart_title) {
  /**
   * topic-names is a list of all the different gas sensor topics
   */
  $scope.gas_name = gas_name;
  $scope.chart_height = chart_height;
  $scope.chart_title = chart_title;
  if (chart_height !== undefined) {
    $scope.gasChart.options.chart.height = chart_height;
  }
  if ($scope.chart_title) {
    $scope.gasChart.options.title.text = chart_title;
  }
  $scope.topicNames = $scope.vm.gasTopics[gas_name];
};
function deref(obj, s) {
  var i = 0;
  if (!s) {
    return undefined;
  }
  s = s.split('.');
  while (i < s.length) {
    obj = obj[s[i]];
    if (obj === undefined)
      return obj;
    i = i + 1;
  }
}

```

```

return obj;
};
function getMessageName(topic) {
    var name_splice = topic.name.split("/");
    var type_splice = topic.type.split("/");
    return ("vm.messages."+name_splice[1] + "." + name_splice[2] + "." + type_splice[0] + "." +
        type_splice[1] + ".");
};

$scope.$watch(function($scope) {
    var val = 0;
    // for each of the sensor packs, there are certain sensors that are better than others for
    // certain gases
    // these are listed under *sensors*
    // So we want to add and average these different sensor values for each sensor pack
    $.each($scope.topicNames.topics, function(e) {
        var sensor_pack = $scope.topicNames.topics[e];
        var sensor_average = 0;
        var count = 0;
        // iterate over each sensor
        // console.log($scope.topicNames.sensors);
        $.each($scope.topicNames.sensors, function(s) {
            var sensor = $scope.topicNames.sensors[s];
            var sensor_path = getMessageName(sensor_pack)+sensor+"."+ $scope.gas_name;
            var sensor_val = deref($scope, sensor_path);
            if (sensor_val) {
                sensor_average += sensor_val;
                count++;
            }
        });
        sensor_average = sensor_average/count;
        val += sensor_average;
    });
    return (val/$scope.topicNames.topics.length);
},function(val){
    if(val){
        $scope.gasChart.series[0].data[0] = Math.abs(val);
    }
}, false);
});
console.log("Loaded controller: ", ctrl);

```

## User Interface HTML Dashboard

```

<div class="panel panel-default" ng-class="'panel-success ': vm.toggle" id="dashboard-panel">
  <div class="panel-heading clearfix">
  </div>
  <div class="panel-body">
    <p>Timestamp: {{ vm.messages.VehicleState.rsl_rover_msgs.vehicle_state.header.stamp.secs *
        1000 | date:'yyyy-MM-dd HH:mm:ss Z'}}</p>
    <div class="container-fluid">
      <div class="col-lg-2">
        <div id="cameras">
          
          
          
          
        </div>
      </div>
      <div class="col-lg-7">
        <div lidar-viz id ="lidar_viz" style="height:500px; width:100%" ng-init="init(500,
          'lidar_viz ')">
        </div>
      </div>
      <div class="col-lg-3">
        <!-- SPEDOMETER -->
      </div>
    </div>
  </div>

```



```

<div ng-controller="speedChartController" ng-init="init('vm.messages.VehicleState.
    rsl_rover_msgs.vehicle_state.wheel_speed', 200);">
    <highchart id="speed_chart" config="speedChartConfig" class="chart"></
        highchart>
</div>
<div class="row">
    <div class="col-lg-6">
        <div ng-controller="gasChart">
            <highchart id="c02Chart" config="gasChart" class="chart" ng-init="init
                ('CO2', 120, 'CO2')"></highchart>
        </div>
    </div>
    <div class="col-lg-6">
        <div ng-controller="gasChart">
            <highchart id="c0Chart" config="gasChart" class="chart" ng-init="init
                ('CO', 120, 'CO')"></highchart>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-lg-6">
        <div ng-controller="gasChart">
            <highchart id="GasChart" config="gasChart" class="chart" ng-init="init
                ('Methane', 120, 'CH4')"></highchart>
        </div>
    </div>
    <div class="col-lg-6">
        <div ng-controller="gasChart">
            <highchart id="PropaneChart" config="gasChart" class="chart" ng-init="
                init('Propane', 120, 'C3H8')"></highchart>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>

```

## User Interface HTML Vehicle State

```

<div class="panel panel-default" ng-class="{ 'panel-success': vm.toggle}">
    <div class="panel-heading clearfix">
        <button id="vehicle_state_sub" class="btn btn-default btn-sm pull-right"
            ng-click="vm.toggleSubscription(!vm.toggle)" ng-model="vm.toggle" uib-btn-checkbox>{{ vm.
                toggle ? 'Unsubscribe' : 'Subscribe' }}</button>
        <h3 class="panel-title">
            {{ vm.topic.abbr }}
            <small style="font-size: 12px;">({{ vm.topic.type }})</small>
        </h3>
    </div>
    <div class="alert alert-danger" ng-show="vm.message.estop != null && vm.message.estop != false
        ">
        ESTOP Occured! Code: <strong>{{vm.message.estop_code}}</strong>
    </div>
    <div class="panel-body">
        <div class="row">
            <div class="col-sm-12">
                <div class="form-group">
                    <label>Timestamp</label>
                    {{ vm.message.header.stamp.secs * 1000 | date:'yyyy-MM-dd HH:mm:ss Z' }}
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-sm-6">
                <form class="form-horizontal form-margin">
                    <div class="form-group">
                        <div class="row">
                            <div ng-controller="wheelAngleChart">
                                <highchart id="actual_steering_chart" config="wheelAngleConfig"
                                    topicName="vm.message.actual_steering" class="chart"></highchart>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-md-6">
                            <label>Actual Steering</label>
                            <div class="input-group">
                                <input type="number" class="form-control" ng-model="vm.message
                                    .actual_steering" ng-readonly="vm.isSubscribing">
                                <span class="input-group-addon">degrees</span>
                            </div>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>

```

```

        <div class="col-md-6">
            <label>Desired Steering</label>
            <div class="input-group">
                <input type="number" class="form-control" ng-model="vm.message
                    .desired_steering" ng-readonly="vm.isSubscribing">
                <span class="input-group-addon">degrees</span>
            </div>
        </div>
    </div>
</div>
<div class="form-group">
    <label>Wheel Pos</label>
    <div class="input-group">
        <input type="number" class="form-control" ng-model="vm.message .
            wheel_pos" ng-readonly="vm.isSubscribing">
        <span class="input-group-addon">m/s<sup>2</sup></span>
    </div>
</div>
<div class="form-group">
    <label>Desired Throttle</label>
    <div class="input-group">
        <input type="number" class="form-control" ng-model="vm.message .
            desired_throttle" ng-readonly="vm.isSubscribing">
        <span class="input-group-addon">m/s</span>
    </div>
</div>
</form>
</div>
<div class="col-sm-6">
    <form class="form-horizontal form-margin">
        <div class="form-group">
            <div ng-controller="speedChartController" ng-init="topicName='vm.message .
                wheel_speed'">
                <highchart id="speed_chart" config="speedChartConfig" class="chart"></
                    highchart>
            </div>
            <label>Wheel Speed</label>
            <div class="input-group">
                <input id='wheel_speed_val' type="number" class="form-control" ng-
                    model="vm.message . wheel_speed" ng-readonly="vm.isSubscribing">
                <span class="input-group-addon">m/s</span>
            </div>
        </div>
        <div class="form-group">
            <div class="row">
                <div class="col-md-6">
                    <label>Desired Gear</label>
                    <div class="input-group">
                        <input type="string" class="form-control" ng-model="vm.message
                            . desired_gear" ng-readonly="vm.isSubscribing">
                    </div>
                </div>
                <div class="col-md-6">
                    <label>Actual Gear</label>
                    <div class="input-group">
                        <input type="string" class="form-control" ng-model="vm.message
                            . actual_gear" ng-readonly="vm.isSubscribing">
                    </div>
                </div>
            </div>
        </div>
        <div class="form-group">
            <label>Voltage Warn</label>
            <div class="input-group">
                <input type="bool" class="form-control" ng-class="{ 'bg-danger': vm.
                    message . voltage_warn}"
                    ng-model="vm.message . voltage_warn" ng-readonly="vm.isSubscribing">
            </div>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>

```

## Tachometer Arduino: tach\_arduino.ino

```

#include <Wire.h>
#include <Encoder.h>

```

```

#define SLAVE_ADDRESS 0x60
#define FLOATS_SENT 1
#define DELAY_TIME 50

long positionA;
long positionB;
unsigned long timeA;
unsigned long timeB;
byte data[8];
double rpm=0;
double wheel_speed=0;

Encoder myEnc(2, 3);

void setup() {
pinMode(2, INPUT_PULLUP);
pinMode(3, INPUT_PULLUP);
pinMode(9, OUTPUT); //PWM Output b/c legacy compatability
// pinMode(13, OUTPUT);

Serial.begin(9600);

// initialize i2c as slave
Wire.begin(SLAVE_ADDRESS);

// define callbacks for i2c communication
Wire.onRequest(sendData);
Serial.println("SETUP COMPLETE");
}

void loop() {
timeA = millis();
positionA = myEnc.read();
delay(DELAY_TIME);
timeB = millis();
positionB = myEnc.read();

//Calculates the wheel speed averaged over roughly 1/5 second
float enc_spd = float(positionB - positionA)/float(timeB-timeA)*1000.0;
// Serial.println(String(positionA) + "," + String(positionB) + "," + String(int(enc_spd)));
// enc_spd = enc_spd*0.5*0.07735;

// rpm=enc_spd;
wheel_speed = .07735*rpm;

byte* byte_spd = (byte *) &enc_spd;
byte* byte_pos = (byte *) &positionB;

//A bit hackish but should work
data[0] = byte_spd[0];
data[1] = byte_spd[1];
data[2] = byte_spd[2];
data[3] = byte_spd[3];
data[4] = byte_pos[0];
data[5] = byte_pos[1];
data[6] = byte_pos[2];
data[7] = byte_pos[3];
Serial.println(rpm);

// wheel_speed = (wheel_speed + 40)*(3.1875);
}

void sendData(){
Wire.write((byte *) &data, 8);
}

```

## Console Arduino: Console\_Code\_v2.ino

```

//RSL Rover 2014
//Console Code
// Serial 1: To Vehicle (drive by wire)
// Serial 22: To Vehicle (XBee)

#include <LiquidCrystal.h> //Library for LCD screen
#include <EasyTransfer.h> //Library for serial communication
#include "comm_definitions.h"
#include "pin_definitions.h"
#include "config.h"
#include "fscale.h"

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //LCD Screen initialization

```

```

//Serial communication objects to facilitate comms between console & rover
CONSOLE.TO.ROVER txdata;
ROVER.TO.CONSOLE rxdata;
EasyTransfer ETin, ETout;
//EasyTransfer ETdebug;
int packet_id=0;

//Strings (for more\ info see string initializations in setup)
//String command_type;
//String command_mode;
//String throttle_or_steer;
//String steering_string_to_send;
//String speed_string_to_send;
//String gear_string_to_send;
char desired_gear; // Stores the gear selected by the button on the console
char previous_desired_gear; // Stores the last "desired_gear" to detect a change
//String comma;
//String aux_string;
//String aux_string_previous;
//String A_status;
//String B_status;
//String C_status;
//String D_status;
//String E_status;
//String horn_status;
//String F_status;
//String return_string;
//String string_from_usb;
//String string_from_usb_steering;
//String string_from_usb_speed;
//String string_from_usb_gear;
String act_speed; //Stores a string of the current speed as reported by the rover to display on
the LCD
String act_steer; //Same as above but for steering
char act_gear; //The current rover gear as reported by the rover
bool armed_status; //Is the rover armed?
bool temp; //Temperature warning flag
bool voltage; //Voltage warning flag
long int mark;

//int counter = 0; //used to control the intermittent
// sending of data
//int comma_index.1 = 0; //Index of first comma in a string (
// for parsing)
//int comma_index.2 = 0; //Index of second comma in a string (
// for parsing)
//int comma_index.3 = 0; //Index of third comma in a string (
// for parsing)
//int comma_index.4 = 0; //Index of fourth comma in a string (
// for parsing)

int desired_steering = 0; //Desired steering position (int from
-1000(left) to 1000(right) with center at 0)
int steer_pos = 0; //Used to turn string act_steer into an
int to display actual steering position in real time
int desired_speed; //The variable desired_speed holds the
input for the speed potentiometer on the joystick, then is mapped from -1000 to 1000 and sent
in a string to the Vehicle Mega
int governor; //Reading frmo governor potentiometer
used to saturate throttle commands

int start_time=0; //Used to store starting time to meter
the flash rate
int stop_time=0; //Used to store ending time to meter the
flash rate
int elapsed_time=0; //Used to calculate elapsed time to
meter the flash rate
int LED.state=LOW; //Used to set the state of an LED (HIGH
for on, LOW for off)

int voltage_input = 0; //Used to store the initial analog input
reading for console battery voltage
float Batt_Voltage = 0; //Used to store the value of the
calculated battery voltage from the voltage_input

bool serial_sw_last = false; //Stores last switch state

void setup() //Runs before the main loop to
initialize everything
{
Serial.begin(115200); //Serial to/from USB or serial monitor (
sets baud rate and opens serial port)
Serial.setTimeout(Serial_timeout); //If the serial buffer misses the '\r'
character, it will read a really long string. Setting the timeout ensures that if the
controller recieves a long garbage string, it will not waste time reading it
Serial1.begin(9600); //Serial to/from the Vehicle via drive by
wire (sets baud rate and opens serial port)

```

```

Serial1.setTimeout(Serial_timeout); //If the serial buffer misses the '\r'
character, it will read a really long string. Setting the timeout ensures that if the
controller receives a long garbage string, it will not waste time reading it
Serial2.begin(115200); //Serial to/from the Vehicle via x-bee
radio (sets baud rate and opens serial port)
Serial2.setTimeout(Serial_timeout); //If the serial buffer misses the '\r'
character, it will read a really long string. Setting the timeout ensures that if the
controller receives a long garbage string, it will not waste time reading it

ETin.begin(details(rxdata), &Serial2);
ETout.begin(details(txdata), &Serial2);
//ETdebug.begin(details(txdata), &Serial);

lcd.begin(20, 4); //Sets up and opens port to LCD screen

pinMode(ind_H, OUTPUT); //Sets up digital pin ind_H as a digital
output
pinMode(ind_L, OUTPUT); //Sets up digital pin ind_L as a digital
output
pinMode(ind_N, OUTPUT); //Sets up digital pin ind_N as a digital
output
pinMode(ind_R, OUTPUT); //Sets up digital pin ind_R as a digital
output
pinMode(ind_P, OUTPUT); //Sets up digital pin ind_P as a digital
output
pinMode(ind_1, OUTPUT); //Sets up digital pin ind_1 as a digital
output
pinMode(ind_2, OUTPUT); //Sets up digital pin ind_2 as a digital
output
pinMode(ind_3, OUTPUT); //Sets up digital pin ind_3 as a digital
output
pinMode(ind_4, OUTPUT); //Sets up digital pin ind_4 as a digital
output
pinMode(ind_5, OUTPUT); //Sets up digital pin ind_5 as a digital
output
pinMode(ind_6, OUTPUT); //Sets up digital pin ind_6 as a digital
output

lcd.setCursor(0, 0); lcd.print("HERE"); delay(80); //Startup procedure for LCD screen on
console
lcd.setCursor(0, 1); lcd.print("WE"); delay(80);
lcd.setCursor(0, 2); lcd.print("GO"); delay(150);
lcd.setCursor(0, 0); lcd.print(" ");
lcd.setCursor(0, 1); lcd.print(" ");
lcd.setCursor(0, 2); lcd.print(" ");

for (int flash=37; flash<=47; flash++) //Startup procedure for LED flash cycle on
console
{
digitalWrite(flash, HIGH);
delay(LED_delay);
digitalWrite(flash, LOW);
}
digitalWrite(ind_1, HIGH);

// command_type = String("C"); //Sent at the beginning of a string
sent to the vehicle: C for command, ? for queries (not yet involved), etc
// command_mode = String(""); //Sent in string to vehicle to
indicate mode: A for actuator, S for speed control modes
// throttle_or_steer = String(""); //Sent in command string to vehicle to
indicate whether the command is a steering or speed related command: V for speed related
commands, W for steering related commands
// steering_string_to_send= String("C,A,W,0"); //Steering command string sent to
vehicle (initialized to center command)
// speed_string_to_send = String("C,A,V,0"); //Speed related command sent to
vehicle (initialized to zero meaning no brake and no throttle)
// gear_string_to_send = String("C,G,P"); //Gear change command sent to vehicle
(initialized to park)
desired_gear = 'P'; //Sent in the gear_string_to_send to indicate the
desired gear (initialized to park)
previous_desired_gear = 'P'; //Stores the previously desired gear to insure that
the gear change string only gets sent if a different desired gear is input
// comma = String(","); //Used to separate different parts of
a command string sent to vehicle so the string can be parsed
// aux_string = String("XXXXXXX"); //String to send to vehicle to
indicate the position of the auxiliary pushbuttons or rockers
// aux_string_previous = String("XXXXXXX"); //String to store aux_string to
determine if the status of any buttons has changed
// A.status = String(""); //String sent in aux_string to
indicate that auxiliary button A is in the on position ("A" if read HIGH, "X" if read LOW)
// B.status = String(""); //String sent in aux_string to
indicate that auxiliary button B is in the on position ("B" if read HIGH, "X" if read LOW)
// C.status = String(""); //String sent in aux_string to
indicate that auxiliary button C is in the on position ("C" if read HIGH, "X" if read LOW)
// D.status = String(""); //String sent in aux_string to
indicate that auxiliary button D is in the on position ("D" if read HIGH, "X" if read LOW)
// E.status = String(""); //String sent in aux_string to

```

```

        indicate that auxiliary button E is in the on position ("E" if read HIGH, "X" if read LOW)
// F.status = String(""); //String sent in aux_string to
        indicate that auxiliary button F is in the on position ("F" if read HIGH, "X" if read LOW)
// horn.status = String(""); //String sent in aux_string to
        indicate that auxiliary button H (Horn) is in the on position ('H' if read HIGH, "X" if read
        LOW)
// return_string = String(""); //Stores string sent back to console
        from vehicle either as feedback or a fault code
// string_from_usb = String(""); //Stores string sent to console from
        the USB input
act_speed = String("x"); //Stores the wheel speed value from the
        vehicle feedback to display on LCD screen
act_steer = String("x"); //Stores the steering position value from
        the vehicle feedback to display on LCD screen
act_gear = 'x'; //Stores the current gear that the vehicle is in
        from the vehicle feedback to display on the console LEDs
armed_status = false; //Sent from vehicle to console to alert console
        that the vehicle is armed and ready to take commands ("A" for armed, "X" for not armed)
temp = false; //Sent from vehicle to console to alert console
        that the vehicle's temperature warning light is on ("T" for temperature light on, "X" for
        temperature light off)
voltage = false; //Sent from vehicle to console to alert console
        that one of the vehicle's systems is at a low voltage ("V" for under voltage, "X" for healthy
        voltage levels)

pinMode(high_gear , INPUT); //Sets up digital pin high_gear as a
        digital input
pinMode(low_gear , INPUT); //Sets up digital pin low_gear as a digital
        input
pinMode(neutral_gear , INPUT); //Sets up digital pin neutral_gear as a
        digital input
pinMode(reverse_gear , INPUT); //Sets up digital pin reverse_gear as a
        digital input
pinMode(park_brake , INPUT); //Sets up digital pin park_brake as a
        digital input
pinMode(speed_vs_actuator , INPUT); //Sets up digital pin speed_vs_actuator as
        a digital input
pinMode(serial_dbw_rc , INPUT); //Sets up digital pin serial_dbw_rc as a
        digital input
pinMode(aux_A , INPUT); //Sets up digital pin aux_A as a digital
        input
pinMode(aux_B , INPUT); //Sets up digital pin aux_B as a digital
        input
pinMode(aux_C , INPUT); //Sets up digital pin aux_C as a digital
        input
pinMode(aux_D , INPUT); //Sets up digital pin aux_D as a digital
        input
pinMode(aux_E , INPUT); //Sets up digital pin aux_E as a digital
        input
pinMode(horn , INPUT); //Sets up digital pin horn as a digital
        input
pinMode(aux_F , INPUT); //Sets up digital pin aux_F as a digital
        input

mark = millis();
}

int map_joystick(int minimum, int min_dead, int max_dead, int maximum, int pos) //Maps
        joystick input from minimum to maximum value (-1000 to 1000) for speed and steering inputs
        while taking into account deadband
{
    if (pos>max_dead)
    {
        pos=map(pos, max_dead, maximum, 0, 1000);
        //pos = (int) round( fscale( (float) max_dead, (float) maximum, 0.0, 1000.0, (float) pos, 0.0) );
    }
    else if (pos<min_dead)
    {
        pos=map(pos, min_dead, minimum, 0, -1000);
        //pos = (int) round( fscale( (float) min_dead, (float) maximum, 0.0, -1000.0, (float) pos, 0.0) );
    }
    else
    {
        pos=0;
    }
    return -1*pos;
}

void aux_switch_read() //Function to
        read the auxilliary switch digital inputs and send a string to Vehicle Mega indicating their
        status
{
// if(digitalRead(aux_A)==HIGH) //Checks to
        see if aux_A pin is HIGH
// {A.status= String("A");} //If the
        button is on, assign the "A" character to its place in aux_string
}

```

```

// else{A.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string //Checks to
// if(digitalRead(aux.B)==HIGH) //Checks to
// see if aux.B pin is HIGH
// {B.status= String("B");} //If the
// button is on, assign the "B" character to its place in aux_string //If the
// else{B.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string
// if(digitalRead(aux.C)==HIGH) //Checks to
// see if aux.C pin is HIGH
// {C.status= String("C");} //If the
// button is on, assign the "C" character to its place in aux_string //If the
// else{C.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string
// if(digitalRead(aux.D)==HIGH) //Checks to
// see if aux.D pin is HIGH
// {D.status= String("D");} //If the
// button is on, assign the "D" character to its place in aux_string //If the
// else{D.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string
// if(digitalRead(aux.E)==HIGH) //Checks to
// see if aux.E pin is HIGH
// {E.status= String("E");} //If the
// button is on, assign the "E" character to its place in aux_string //If the
// else{E.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string
// if(digitalRead(aux.F)==HIGH) //Checks to
// see if aux.F pin is HIGH
// {F.status= String("F");} //If the
// button is on, assign the "F" character to its place in aux_string //If the
// else{F.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string
// if(digitalRead(horn)==HIGH) //Checks to
// see if horn pin is HIGH
// {horn.status= String('H');} //If the
// button is on, assign the 'H' character to its place in aux_string //If the
// else{horn.status= String("X");} //If the
// button is off, assign the "X" character to its place in aux_string

txdata.aux[0] = digitalRead(aux.A);
txdata.aux[1] = digitalRead(aux.B);
txdata.aux[2] = digitalRead(aux.C);
txdata.aux[3] = digitalRead(aux.D);
txdata.aux[4] = digitalRead(aux.E);
txdata.aux[5] = digitalRead(aux.F);
txdata.aux[6] = digitalRead(horn);

// aux_string = String("A") + A.status + B.status + C.status + D.status + E.status + F.status +
// horn.status; //Formulate auxilliary switch string
//
// if(aux_string != aux_string.previous) //So the
// string is only sent when the status of a button or rocker changes
// {
// Serial.println(aux_string);
// Serial1.println(aux_string); //Send
// auxilliary switch string over drive by wire
// Serial2.println(aux_string); //Send
// auxilliary switch string over x-bees
// }
//
// aux_string.previous = aux_string;
}

void return_command()//String return_string) //Function
to parse string coming from vehicle and assign variables based on string
{
// if(return_string.startsWith("F")) //If the
// string starts with "F" it is a feedback string not an error string
// {
// comma_index_1 = return_string.indexOf(','); //Records
// index value of first comma in string for parsing
// comma_index_2 = return_string.indexOf(', ', comma_index_1 + 1); //Records
// index value of second comma in string for parsing
// comma_index_3 = return_string.indexOf(', ', comma_index_2 + 1); //Records
// index value of third comma in string for parsing
//
// act_steer = return_string.substring(comma_index_1 + 1, comma_index_2); //Parses
// and records value of the current steering position
// act_speed = return_string.substring(comma_index_2 + 1, comma_index_3); //Parses
// and records value of the current wheel speed
// act_gear = return_string.substring(comma_index_3 + 1); //Parses
// and records value of the current gear
// }

if(ETin.receiveData()){
act_steer = String(rxdata.rover_steering);
}

```

```

act_speed = String(rxdata.rover_speed);
act_gear = rxdata.rover_gear;

// else if(return_string.startsWith("E")) //If the
// vehicle is sending the console an error code
// {
//   armed_status = return_string.substring(1, 2); //Parse
//   armed_status out of error code
//   voltage = return_string.substring(2, 3); //Parse
//   voltage out of error code
//   temp = return_string.substring(3, 4); //Parse
//   temp out of error code
// }

armed_status = rxdata.armed_status;
voltage = rxdata.voltage_error;
temp = rxdata.temp_error;

// else if(return_string.charAt(1)=='E') //Same as
// previous else if statement. This should not be necessary but every once in a while the
// previous statement lets an error code go and this catches it and keeps it from going
// undetected
// {
//   armed_status = return_string.substring(2, 3);
//   voltage = return_string.substring(3, 4);
//   temp = return_string.substring(4, 5);
// }

if(armed_status){digitalWrite(ind_2, HIGH);} //Check if the vehicle
// is armed. Illuminate ARMED LED if it is
else {digitalWrite(ind_2, LOW);}

if(temp){digitalWrite(ind_4, HIGH);} //Check if the vehicle
// 's temperature warning light is on. Illuminate ERROR LED if it is
else {digitalWrite(ind_4, LOW);}

if(voltage){digitalWrite(ind_3, HIGH);} //Check if the vehicle
// is under voltage. Illuminate LOW BATT LED if it is
else {digitalWrite(ind_3, LOW);}
}

//void serial_interface_function() //If the
// serial versus remote control or drive by wire rocker is in the serial position, this function
// is called to take read the input string from the USB port and deal with it
//{
// if(Serial.available())
// {
//   string_from_usb = Serial.readStringUntil('\r'); //Read
//   string from USB port until return character
// }
// }
// if (string_from_usb.substring(2,3) == "G") //Checks to
// see if it is a gear change command
// {
//   gear_string_to_send = string_from_usb; //If the
//   input string is a gear change string, set it equal to gear_string_to_send
//   if(gear_string_to_send != "Clear") //Checks to
//   make sure that the gear change string only gets sent once (sets equal to "Clear" later in the
//   code after it has been sent)
//   {
//     Serial1.println(gear_string_to_send); //Send
//     gear_string_to_send to vehicle via drive by wire
//     Serial2.println(gear_string_to_send); //Send
//     gear_string_to_send to vehicle via x-bees
//     delay(20);
//   }
//   digitalWrite(ind_H, LOW); //turns off
//   all gear indicator LEDs that may have been on
//   digitalWrite(ind_L, LOW);
//   digitalWrite(ind_N, LOW);
//   digitalWrite(ind_R, LOW);
//   digitalWrite(ind_P, LOW);
// }
// }
// else if (string_from_usb.substring(4,5) == "W") //If the
// USB input string is not a gear change command this checks to see if it is a steering command
// {
//   steering_string_to_send = string_from_usb; //If the
//   input string is a steering string, set it equal to steering_string_to_send
// }
// }
// else if (string_from_usb.substring(4,5) == "V") //If the

```



```

        USB input string is not a gear change or steering command this checks to see if it is a speed
        related command
    // {
    //     speed_string_to_send = string_from_usb; //If the
    //     input string is a speed related command, set it equal to speed_string_to_send
    // }
    //
    // if( act.gear == desired.gear) //Once the
    //     vehicle has shifted gears, stop sending gear change string and get back to sending steering
    //     and speed commands
    // {
    //     gear_string_to_send = String(" Clear"); //Clears
    //     gear_string_to_send so that the previous part of the code knows the vehicle is done changing
    //     gears and stops sending the gear change command
    //
    //     if (counter == 1) //Used to
    //         alternate between sending steering and speed commands (sends the steering command on the first
    //         iteration and speed related command on the second)
    //         {
    //             Serial.println(steering_string_to_send);
    //             Serial1.println(steering_string_to_send); //Send
    //             steering command over drive by wire
    //             Serial2.println(steering_string_to_send); //Send
    //             steering command over x-bees
    //             delay(20);
    //         }
    //
    //         else if (counter >= 2) //Used to
    //             alternate between sending steering and speed commands (sends the steering command on the first
    //             iteration and speed related command on the second)
    //             {
    //                 Serial.println(speed_string_to_send);
    //                 Serial1.println(speed_string_to_send); //Send
    //                 speed related command over drive by wire
    //                 Serial2.println(speed_string_to_send); //Send
    //                 speed related command over x-bees
    //                 delay(20);
    //                 counter = 0; //Reset
    //             }
    //             counter
    //         }
    //
    //         counter = counter +1; //Count
    //         iterations
    //     }
    // }
    //
    // if( act.gear == 'H') {digitalWrite(ind_H, HIGH); digitalWrite(ind_L, LOW); digitalWrite(ind_N,
    // LOW); digitalWrite(ind_R, LOW); digitalWrite(ind_P, LOW);} //Illuminates LED
    // corresponding to the current gear of the vehicle and writes all other LEDs LOW
    // else if (act.gear == 'L') {digitalWrite(ind_L, HIGH); digitalWrite(ind_H, LOW); digitalWrite(ind_N,
    // LOW); digitalWrite(ind_R, LOW); digitalWrite(ind_P, LOW);}
    // else if (act.gear == 'N') {digitalWrite(ind_N, HIGH); digitalWrite(ind_L, LOW); digitalWrite(ind_H,
    // LOW); digitalWrite(ind_R, LOW); digitalWrite(ind_P, LOW);}
    // else if (act.gear == 'R') {digitalWrite(ind_R, HIGH); digitalWrite(ind_N, LOW); digitalWrite(ind_L,
    // LOW); digitalWrite(ind_H, LOW); digitalWrite(ind_P, LOW);}
    // else {digitalWrite(ind_P, HIGH); digitalWrite(ind_R, LOW); digitalWrite(ind_N, LOW); digitalWrite
    // (ind_L, LOW); digitalWrite(ind_H, LOW);}
    // }
    // }

    void rc_dbw_function() //Separate
    // function for when the vehicle is operating in drive by wire or remote control mode as opposed
    // to serial mode
    {
    // string_from_usb = String("C"); //Clears
    // string_from_usb

    if(digitalRead(high-gear)) //Reads digital
    // inputs from gear pushbuttons and assigns the desired.gear to a string indicating the gear if
    // one of the pushbuttons is pressed
    {desired_gear = 'H';}
    else if(digitalRead(low-gear))
    {desired_gear = 'L';}
    else if(digitalRead(neutral-gear))
    {desired_gear = 'N';}
    else if(digitalRead(reverse-gear))
    {desired_gear = 'R';}
    else if(digitalRead(park.brake))
    {desired_gear = 'P';}
    else
    {desired_gear=previous_desired_gear;} //If no gear
    // pushbuttons are pressed, this sets the desired.gear=previous_desired_gear

    if(desired_gear != previous_desired_gear) //Checks to make
    // sure the new desired gear is not equal to the previously desired gear
    {
    for (int flash=ind_H; flash<=ind_P; flash++) //Flash sequence
    
```

```

        during gear change
    {digitalWrite(flash, LOW);}
    }
    previous_desired_gear=desired_gear; //Sets
        previous_desired_gear=desired_gear

    if(act_gear != desired_gear) //Checks to make
        sure the desired gear is differnet from the actual gear and does not do anything further if
        the vehicle is currently in the gear desired
    {
        //command_mode = "G"; //Sets the
        command mode to "G" for gear (This will tell the Vehicle Mega that the command is a gear
        change)
        //gear_string_to_send = command.type + comma + command_mode + comma + desired_gear; //Formulates
        the gear change command to send to the vehicle
        //Serial.println(gear_string_to_send);
        //Serial1.println(gear_string_to_send); //Sends the
        gear change command to the vehicle via drive by wire
        //Serial2.println(gear_string_to_send); //Sends the
        gear change command to the vehicle via x-bee
        // delay(20);

        txdata.gear = desired_gear; //Send a packet with 0
        speed and brake on
        txdata.throttle_pos = -500;

        stop_time=millis(); //Assigns the
        stop time used to calculate the elapsed time which enforces the flash rate
        elapsed_time=stop_time-start_time; //Calculates the
        elapsed time used to enforces the flash rate
        if (elapsed_time > flash_rate) //Checks to see
            if the elapsed time is greater than the flash rate
        {
            start_time=millis(); //Assigns the
            start time used to calculate the elapsed time which enforces the flash rate
            LED.state = (!LED.state); //If the elapsed
            time is greater than the flash rate, the LED switches states

            if(desired_gear == 'H') {digitalWrite(ind_H, LED.state);} //Writes LED.state
            (HIGH or LOW) to digital pin corresponding to the gear that is changing
            else if (desired_gear == 'L') {digitalWrite(ind_L, LED.state);}
            else if (desired_gear == 'N') {digitalWrite(ind_N, LED.state);}
            else if (desired_gear == 'R') {digitalWrite(ind_R, LED.state);}
            else {digitalWrite(ind_P, LED.state);}
        }
        }

        //If not changing gears, set the corresponding gear LED
        else if(act_gear == 'H') {digitalWrite(ind_H, HIGH);digitalWrite(ind_L, LOW);digitalWrite(ind_N,
        LOW);digitalWrite(ind_R, LOW);digitalWrite(ind_P, LOW);} //If the vehicle is in the gear that
        is desired, illuminate the corresponding LED
        else if (act_gear == 'L') {digitalWrite(ind_L, HIGH);digitalWrite(ind_H, LOW);digitalWrite(ind_N,
        LOW);digitalWrite(ind_R, LOW);digitalWrite(ind_P, LOW);}
        else if (act_gear == 'N') {digitalWrite(ind_N, HIGH);digitalWrite(ind_L, LOW);digitalWrite(ind_H,
        LOW);digitalWrite(ind_R, LOW);digitalWrite(ind_P, LOW);}
        else if (act_gear == 'R') {digitalWrite(ind_R, HIGH);digitalWrite(ind_N, LOW);digitalWrite(ind_L,
        LOW);digitalWrite(ind_H, LOW);digitalWrite(ind_P, LOW);}
        else {digitalWrite(ind_P, HIGH);digitalWrite(ind_R, LOW);digitalWrite(ind_N, LOW);digitalWrite(
        ind_L, LOW);digitalWrite(ind_H, LOW);}

        // governor = analogRead(gov-pin); //Read
        governor analog input
        governor = map(analogRead(gov-pin), 1023, 0, 0, 1000); //Map or
        scale the governor's input from 0 to 1000

        if(act_gear == desired_gear) //Checks to
            ensure that the vehicle is in the gear that is desired
        {
            // if(counter==1) //If on
            the first iteration, formulate and send speed related commands (counter used to alternate
            between speed related commands and steering related commands)
            // {
            if(digitalRead(speed_vs_actuator)) //Checks whether the
                console indicates speed or actuator mode and assigns the command mode to the desired mode
            {
                // command_mode = String("S");
                txdata.speedmode = 'S';
            }
            else
            {
                // command_mode = String("A");
                txdata.speedmode = 'A';
            }
        }

        desired_speed = analogRead(speed_pot_pin); //Reads speed
        analog input from joystick

```

```

desired_speed= map-joystick(joy_min_speed, joy_min_speed_deadband, joy_max_speed_deadband,
joy_max_speed, analogRead(speed_pot.pin)); //Maps speed analog input from
joystick with speed potentiometer deadband

// if(act_gear == 'R') //If the
// vehicle is currently in reverse, the desired speed commands should be reversed so it //If the
// intuitively makes sense to drive (in reverse if you push the joystick back, you will receive
// throttle input and if you push it forward the brakes will be activated)
// {
//     desired_speed = -1*desired_speed;
// }
//GOVERNOR: (Sanity Check?)
if(governor<desired_speed) //If the desired
// speed is greater than the governor input, govern the speed or throttle input from the
// joystick
{
desired_speed = governor;
}

// throttle_or_steer = String("V"); //Set
// throttle_or_steer equal to "V" to indicate that this is a speed related command
// speed_string_to_send= command_type + comma + command_mode + comma + throttle_or_steer +
// comma + desired_speed; //Formulate speed related command
// Serial.println(speed_string_to_send);
// Serial.println(speed_string_to_send); //Send
// speed related command to vehicle via drive by wire
//Serial2.println(speed_string_to_send); //Send speed
// related command to vehicle via x-bee
//delay(20);
// }

// if(counter>=2) //If on
// the second iteration, formulate and send steering commands (counter used to alternate between
// speed related commands and steering related commands)
// {
//     desired_steering = analogRead(steering_pot.pin); //Reads analog
//     input from steering potentiometer on joystick
//     desired_steering= map-joystick(joy_min_steer, joy_min_steer_deadband, joy_max_steer_deadband,
//     joy_max_steer, desired_steering); //Maps or scales the analog steering input
//     from joystick
//     throttle_or_steer = String("W"); //Set
//     throttle_or_steer equal to "W" to indicate that this is a steering command
//
//     steering_string_to_send= command_type + comma + command_mode + comma + throttle_or_steer +
//     comma + desired_steering; //Formulate steering command
//     Serial.println(steering_string_to_send);
//     Serial.println(steering_string_to_send); //Send
//     steering command to vehicle via drive by wire
//     Serial2.println(steering_string_to_send); //Send
//     steering command to vehicle via x-bee
//     delay(20);
//     counter = 0; //Reset
//     counter
// }

// counter = counter +1; //Keep
// count of iteration
txdata.throttle_pos = desired_speed;
txdata.gear = desired_gear;
} // end of if(act_ger == desired_gear)

txdata.steering_pos = desired_steering;
txdata.packet_id=packet_id;
packet_id++;
ETout.sendData();
// ETdebug.sendData();
}

void batt_check() //Function
// to check battery voltage in console
{
// voltage_input = analogRead(Batt_Voltage.pin); //Reads
// analog input of console battery voltage if in remote control mode or 12 volt voltage from
// vehicle if in drive by wire mode
Batt_Voltage = voltage_input * .01468; //Scales
// battery voltage input to display numerical value in volts
lcd.setCursor(0, 3); lcd.print(Batt_Voltage); //Prints
// battery voltage or 12 volt input to LCD screen
}

void loop() //Main loop
// (iterates over and over)

```

```

{
mark = millis();
// if(digitalRead(serial_dbw_rc)==HIGH) //If the
vehicle is in serial mode, call the serial_interface_fiunction
// {
// if(!serial_sw_last) //If the last state was LOW (manual), then clear the buffer
// {
// while(Serial.available()) {Serial.read();}
// serial_sw_last = true;
// }
// serial_interface_fiunction();
// }
// else //If the
vehicle is in drive by wire or remote control mode, call the rc_dbw_fiunction
// {
rc_dbw_fiunction();
// serial_sw_last = false;
// }
aux_switch_read(); //Reads status
of auxiliary rocker switches and pushbuttons and sends string to vehicle

//there's a loop here so that we run the receive function more often than the
//transmit function. This is important due to the slight differences in
//the clock speed of different Arduinos. If we didn't do this, messages
//would build up in the buffer and appear to cause a delay.

for(int i=0; i<2; i++){
return_command();
}

// if(Serial1.available()) //If the
Vehicle Mega is sending serial data back to the consol via wired serial link, read it and
store it as return_string
// {return_string = Serial1.readStringUntil('\r');
// return_command(return_string);} //Call
function return_command and pass the string return_string to it
//
// if(Serial2.available()) //If the
Vehicle Mega is sending serial data back to the consol via x-bee, read it and store it as
return_string
// {return_string = Serial2.readStringUntil('\r');
// return_command(return_string); //Call
function return_command and pass the string return_string to it
// Serial.println(return_string);} //Forward
debug strings to usb port for debugging

lcd.setCursor(0, 0); lcd.print("Speed ="); //write "Speed =" to LCD
screen
lcd.setCursor(8, 0); lcd.print(" "); //write space to LCD
screen
lcd.setCursor(8, 0); lcd.print(act_speed); //write current wheel
speed to LCD screen
steer_pos=act_steer.toInt(); //Transform
steering feedback from a string to an integer
steer_pos=steer_pos / 10; //
Transofrm steering feedback from -1000 to 1000 value from -100 to 100 so it can be displayed
as a percentage
lcd.setCursor(0, 1); lcd.print("Steering ="); //write "Steering =" to LCD
screen
lcd.setCursor(11, 1); lcd.print(" "); //write space to LCD screen
if (steer_pos<0) {lcd.setCursor(11, 1); lcd.print('L'); steer_pos=-steer_pos;
//If steering is negative, make positive percentage and put a
L before it to indicate left
lcd.setCursor(12,1); lcd.print(steer_pos); lcd.print("%");}
else if (steer_pos>0) {lcd.setCursor(11, 1); lcd.print('R');
//If steering is positive, put a R before it
to indicate right
lcd.setCursor(12,1); lcd.print(steer_pos); lcd.print("%");}
else {lcd.setCursor(11, 1); lcd.print("Center");} //If steering feedback is zero,
write center
lcd.setCursor(0,2); lcd.print("Governor ="); //write "Governor =" to LCD
screen

```

```

lcd.setCursor(11, 2); lcd.print("          ");
//write space to LCD screen
lcd.setCursor(11, 2);
if (digitalRead(speed_vs_actuator)==HIGH)
    //If in speed mode, map
    governor from 0 to maximum vehicle speed and display this in miles per hour
{
governor = map(governor, 0, 1000, 0, max_speed);
lcd.print(governor); lcd.print(" MPH");
}
else
    //If in actuator mode, map governor from 0 to 100 indicating a percentage of throttle input
{
governor = map(governor, 0, 1000, 0, 100);
lcd.print(governor); lcd.print("%");
}
lcd.setCursor(17, 3); lcd.print("RSL");
//Write RSL to the
LCD screen to indicate ownership
//Serial.println(LOOP_TIME - (millis()-mark));
delay(LOOP_TIME - (millis()-mark));
}

```

## Console Arduino: comm\_definitions.h

```

//Includes comm definitions for binary serial packets

struct CONSOLE_TO_ROVER{
int packet_id;
int steering_pos;
int throttle_pos;
char gear;
char speedmode;
bool aux[7];
};

struct ROVER_TO_CONSOLE{
int rover_speed;
int rover_steering;
char rover_gear;
bool voltage_error;
bool temp_error;
bool armed_status;
};

```

## Console Arduino: config.h

```

int flash_rate = 300; //Rate in milliseconds for LED flashing
int LED_delay=40; //Time in milliseconds for initial LED
flash to indicate that the vehicle is about to begin the process of shifting gears
int max_speed = 40; //Maximum speed of vehicle (gives the
governor something to map to in speed mode)
int Serial_timeout = 25; //Set the serial timeout for hardware
serial ports

// Joystick calibration values
int joy_min_speed = 3;
int joy_max_speed = 813;
int joy_min_steer = 1;
int joy_max_steer = 939;

// Joystick deadband values -- in analog counts
int joy_min_speed_deadband = 311;
int joy_max_speed_deadband = 379;
int joy_min_steer_deadband = 316;
int joy_max_steer_deadband = 395;

const int LOOP_TIME = 80; // in milliseconds

```

## Console Arduino: pin\_definitions.h

```

int steering_pot_pin = 0; //Analog input pin associated with the
steering potentiometer on the joystick
int speed_pot_pin = 1; //Analog input pin associated with the
steering potentiometer on the joystick
int gov_pin = 2; //Analog input pin associated with the
governor potentiometer

int high_gear = 26; //Digital input pin number that reads
position of high gear pushbutton
int low_gear = 25; //Digital input pin number that reads
position of high gear pushbutton
int neutral_gear = 23; //Digital input pin number that reads
position of high gear pushbutton
int reverse_gear = 22; //Digital input pin number that reads
position of high gear pushbutton
int park_brake = 24; //Digital input pin number that reads
position of high gear pushbutton

int speed_vs_actuator=27; //Digital input pin number that reads
position of speed versus actuator mode rocker switch
int serial_dbw_rc=29; //Digital input pin number that reads
position of serial versus drive by wire or remote control mode rocker switch

int aux_A=30; //Digital input pin number for
auxilliary rocker switch A
int aux_B=31; //Digital input pin number for
auxilliary rocker switch B
int aux_C=32; //Digital input pin number for
auxilliary rocker switch C
int aux_D=33; //Digital input pin number for
auxilliary rocker switch D
int aux_E=34; //Digital input pin number for
auxilliary rocker switch E

int horn=35; //Digital input pin number for
auxilliary pushbutton H which is currently the horn
int aux_F=36; //Digital input pin number for
auxilliary pushbutton E

int ind_H=37; //LED pin associated with High Gear
int ind_L=38; //LED pin associated with Low Gear
int ind_N=39; //LED pin associated with Neutral Gear
int ind_R=40; //LED pin associated with Reverse Gear
int ind_P=41; //LED pin associated with Park Gear
int ind_1=42; //LED pin to indicate that the console
is on
int ind_2=43; //LED pin to indicate that the vehicle
is armed and ready to accept commands
int ind_3=44; //LED pin to indicate low battery on
vehicle (This LED will illuminate if the 24 volt system falls below 21 volts or if the 12 volt
system falls below 10 volts
int ind_4=45; //LED pin to indicate that the vehicle's
temperature light is on (if this light stays on for more than 20 seconds, the emergency stop
will be activated)
int ind_5=46; //Auxiliary LED pin number
int ind_6=47; //Auxiliary LED pin number
int Batt_Voltage_pin = 3; //Digital pin number to read battery
voltage from f

```

## Vehicle Mega Arduino: Vehicle\_Mega\_2016\_v2.ino

```

//RSL Rover 2016
//Vehicle Mega code

#include <Wire.h>
#include <EasyTransfer.h> //Library for serial communication
//#include <EasyTransferI2C_NL.h>
#include "comm_definitions.h"
#include "pin_definitions.h"
#include "config.h"

//Serial: USB: From ROS Computer
//Serial1: From Consol
//Serial2: To Steering and Transmission Motor Controller
//Serial3: To Speed Controller

//Setting up the serial communication buffers and processing objects
ROVER_TO_CONSOLE console_txdata;
CONSOLE_TO_ROVER console_rxdata;
ROVER_TO_SPEED_ARDUINO spd_txdata;
//SPEED_ARDUINO_TO_ROVER spd_rxdata;
ROVER_TO_ROS ros_txdata;
//TACHLTOROVER tach_rxdata;

```

```

//ETin processes packets from the console to the VehicleMega
//ETout processes packets from the VehicleMega to the console
//ETspd.out processes packets from the VehicleMega to the speed arduino
EasyTransfer ETin, ETout, ETspd.out, ETros; //ETspd.in,
//EasyTransferI2C_NL ETtach;

String command_type;
String command_mode;
String throttle_or_steer;
String steering_command;
String speed_string_to_send;
String steering_string_to_send;
String gear_string_to_send;
char desired_gear;
char current_gear;
String string_from_motor_controller;
String mc_state;
String steering_query;
String feedback_to_consol;
String feedback_to_consol_prefix;
String feedback_to_ROS;
String position_prefix;
String suffix;
String space;
String comma;

bool A_status;
bool B_status;
bool C_status;
bool D_status;
bool E_status;
bool horn_status;
bool F_status;

String error_string;
String error_string_previous;
String temp;
String voltage;

int voltage_input = 0; //Integer placeholder for
    analogRead of voltage divider circuits
float Twenty_Four_V_Voltage = 0; //24 Volt system voltage
float Twelve_V_Voltage = 0; //12 Volt system voltage

int temp_count = 0; //Counter used to ensure
    temp_start_time begins timing when the temperature light first comes on
unsigned long temp_start_time = 0; //Absolute time recorded when
    the temperature light first turns on
unsigned long temp_end_time = 0; //Absolute time recorded every
    time an iteration occurs with the temperature light on
unsigned long temp_time = 0; //Time that the temperature
    light has been on (temp_end_time - temp_start_time)

int e_stop_state = LOW; //if high, e-stop will be
    activated (acts as a toggle and if statements can be added anywhere in the code to toggle
    emergency stop mode on)
int e_brake_state = HIGH; //Emergency brake state: HIGH
    is on, LOW is off
int contact_with_consol = LOW; //Once the Vehicle Mega makes
    initial contact with the console, this state turns HIGH

int counter = 0; //used to control the
    intermittent sending of data
int estoppin = 0;

int desired_speed = 0; //Used to send a desired speed
    to the speed controller when changing gears
float wheel_speed = 0; //Current wheel speed

int desired_throttle = 0;

int gear_position = 0; //Desired position to send to
    gear actuator
int channel=1; //Channel used to formulate
    strings to send to motor controllers (either 1 for steering or 2 for transmission command)
int steering_position=0; //Desired steering position
    parsed from console
int act_steering_position = 0; //Current steering position as
    queried from the steering motor controller (to be sent as feedback to the console)
int comma_index_1; //Index of first comma in a
    string (for parsing)
int comma_index_2; //Index of second comma in a
    string (for parsing)
int comma_index_3; //Index of third comma in a
    string (for parsing)
unsigned long e_stop_time_1 = 0; //Records the start time when
    the last console contact occurred

```

```

unsigned long e_stop_time_2 = 0; //Records the end time when the
    next console contact occurred
unsigned long e_stop_time = 0; //Difference between
    e_stop_time_2 and e_stop_time_1 (compared to dead-man-timout)
int estop_code = 0;

unsigned long tic = 0; //Used for evaluating how
    long it takes to run the main loop
unsigned long toc = 0;

float tach_spd_i2c = 0.0; //Variable for storing the speed
    reported by the tach arduino
long tach_pos_i2c = 0.0; //Variable for storing the
    position as reported by the tach arduino
byte i2c_in[32]; //Buffer for i2c
float engine_rpm = 0.0;

union float_tag { //Unions used for storing the
    i2c data as a byte but getting it as a float or long
    byte b[4];
    float fval;
} u_f;
union long_tag {
    byte b[4];
    long lval;
} u_l;

long int last_steering_sent;
long lastSendTime;

void setup() //Runs before the main loop to
    initialize everything
{
    pinMode(ebrake_relay_pin, OUTPUT); //Sets the emergency brake relay
    pin to output (same as parking brake)
    digitalWrite(ebrake_relay_pin, e_brake_state); //Writes the startup emergency
    brake state to emergency brake pin

    pinMode(e_stop_relay_pin, OUTPUT); //Sets the emergency stop relay
    pin to output
    digitalWrite(e_stop_relay_pin, e_stop_state); //Writes the startup emergency
    stop state to emergency stop pin

    pinMode(horn_relay_pin, OUTPUT); //Sets the horn relay pin to
    output
    digitalWrite(horn_relay_pin, LOW); //Writes the startup horn state to
    horn pin (LOW is off)

    pinMode(beacon_relay_pin, OUTPUT); //Sets the beacon relay pin to
    output
    digitalWrite(beacon_relay_pin, LOW); //Writes the startup emergency
    stop state to emergency stop pin

    // Open serial communications and wait for port to open: //Serial to/from USB or serial
    Serial.begin(115200); //Serial to/from USB or serial
    monitor (sets baud rate and opens serial port)
    Serial.setTimeout(Serial_timeout); //If the serial buffer misses the
    '\r' character, it will read a really long string. Setting the timeout ensures that if the
    controller recieves a long garbage string, it will not waste time reading it
    Serial1.begin(115200); //Serial to/from the Console (
    sets baud rate and opens serial port)
    Serial1.setTimeout(Serial_timeout); //If the serial buffer misses the
    '\r' character, it will read a really long string. Setting the timeout ensures that if the
    controller recieves a long garbage string, it will not waste time reading it
    Serial2.begin(115200); //Serial to/from steering and
    transmission motor controller (sets baud rate and opens serial port)
    Serial2.setTimeout(Serial_timeout); //If the serial buffer misses the
    '\r' character, it will read a really long string. Setting the timeout ensures that if the
    controller recieves a long garbage string, it will not waste time reading it
    Serial3.begin(115200); //Serial to/from Speed
    Controller (sets baud rate and opens serial port)
    Serial3.setTimeout(Serial_timeout); //If the serial buffer misses the
    '\r' character, it will read a really long string. Setting the timeout ensures that if the
    controller recieves a long garbage string, it will not waste time reading it

    pinMode(20, INPUT_PULLUP);
    pinMode(21, INPUT_PULLUP);
    Wire.begin(); //Setup I2C Bus as Master to
    communicate with the tach arduino

    ETin.begin(details(console_rxddata), &Serial1);
    ETout.begin(details(console_txddata), &Serial1);
    ETspd_out.begin(details(spdx_txddata), &Serial3);
    //ETspd_in.begin(details(spdx_rxddata), &Serial3);
    ETros.begin(details(ros_txddata), &Serial);

    pinMode(temp_warning, INPUT); //Sets the temp_warning pin as an

```



```

    input (HIGH or LOW)
pinMode(reverse, INPUT); //Sets the reverse gear pin as an
    input (HIGH or LOW) //Sets the neutral gear pin as an
pinMode(neutral, INPUT);
    input (HIGH or LOW) //Sets the low gear pin as an
pinMode(low, INPUT);
    input (HIGH or LOW) //Sets the high gear pin as an
pinMode(high, INPUT);
    input (HIGH or LOW) //Sets the wheel speed pin as an
pinMode(wheel_speed_pin, INPUT);
    input (PWM) //Sets the wheel speed pin as an
pinMode(47,INPUT);

// consol_input_string = String(""); //String from console
// ros_input_string = String(""); //Command string from ROS
command_type = String(""); //Parsed from consol_input_string:
    C for command, ? for queries (not yet involved), etc
command_mode = String(""); //Parsed from consol_input_string:
    A for actuator, S for speed control modes
throttle_or_steer = String(""); //Parsed from consol_input_string:
    W for steering command, V for speed related commands
steering_command = String(""); //Parsed from consol_input_string:
    Value from -1000 to 1000
steering_string_to_send = String(""); //Formulated string to send as a
    steering motor command to steering and transmission motor controller
speed_string_to_send = String(""); //Formulated string to send as a
    command to speed controller
gear_string_to_send = String(""); //Formulated string to send as a
    transmission motor command to steering and transmission motor controller
desired_gear = ' '; //Parsed from consol_input_string: H for
    high, L for low, N for neutral, R for reverse, P for park
current_gear = 'N'; //Current gear that the vehicle is in: H
    for high, L for low, N for neutral, R for reverse, P for park

steering_query = String("?TR 1"); //Query to be sent to steering and
    transmission motor controller (Asks motor controller what the current steering position is as
    a value from -1000 to 1000)
string_from_motor_controller = String(""); //String sent from steering and
    transmission motor controller
mc_state = String(""); //State of the steering and
    transmission motor controller: "Ready" when ready to take commands "Starting" when performing
    startup procedure
feedback_to_consol = String(""); //Feedback string to console
    includes wheel speed, current gear, and current steering position
feedback_to_consol_prefix = String("F"); //Prefix for feedback_to_consol so
    console recognizes this as feedback and not an error string
feedback_to_ROS = String(""); //Feedback to send over USB Serial

suffix = String("\r"); //Return character to send at the
    end of command or query to motor controller (denotes the end of a string of data)
space = String(" "); //Space needed in motor controller
    commands
comma = String(","); //Comma used mainly to separate
    variables for data logging
position_prefix = String("!g"); //"!g" is how absolute position
    commands to motor controllers begin

// A_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button A is in the on position ("A" if read HIGH, "X" if read LOW)
// B_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button B is in the on position ("B" if read HIGH, "X" if read LOW)
// C_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button C is in the on position ("C" if read HIGH, "X" if read LOW)
// D_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button D is in the on position ("D" if read HIGH, "X" if read LOW)
// E_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button E is in the on position ("E" if read HIGH, "X" if read LOW)
// F_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button F is in the on position ("F" if read HIGH, "X" if read LOW)
// horn_status = String(""); //String parsed from aux_string to
    indicate that auxiliary button H (Horn) is in the on position ("H" if read HIGH, "X" if read
    LOW)

error_string = String("XXXX");
error_string_previous = String("XXXX");
temp = String("X");
voltage = String("X");

//Casues Vehicle Mega to wait for contact from the consol to enter the main loop so that the
    emergency stop is not hit if the console is powered up after the vehicle
Serial.println("M,Done setting up ... waiting on console contact");

top:
if (!ETin.receiveData())
{
    delay(50);
    goto top;
}

```

```

}
else
{
Serial.println("M, Console Contacted! Waiting 4.5s");
delay(4500);
Serial.println("M, Done Waiting!");
}
last_steering_sent=millis();
lastSendTime = millis(); // For comm sending delays
Serial.println("M, Starting Main Loop");
}

void readFromSpd()
{
//if (ETspd.in.receiveData())
//{
//engine_rpm = spd_rxdata.engine_rpm;
//}
}

void readFromTach()
{
// if (ETtach.receiveData(TACH_SLAVE_ADDRESS)) {
// tach_pos_i2c = tach_rxdata.pos;
// tach_spd_i2c = tach_rxdata.spd;
// }

Wire.requestFrom(TACH_SLAVE_ADDRESS, 8); // request 8 bytes from slave device 0x60
int readcount = 0;
while (Wire.available()) { // slave may send less than requested
i2c_in[readcount] = Wire.read(); // receive a byte as character
readcount++;
}
/*
float tach_wheespd_i2c = 0.0;
long tach_pos_i2c = 0.0;
*/
//Serial.println("M, Good Tach Contact");
//Convert raw bytes to a float and a long for speed and position respectively using unions
u.f.b[0]=i2c.in[0];
u.f.b[1]=i2c.in[1];
u.f.b[2]=i2c.in[2];
u.f.b[3]=i2c.in[3];
tach_spd_i2c = u.f.fval;
u.l.b[0]=i2c.in[4];
u.l.b[1]=i2c.in[5];
u.l.b[2]=i2c.in[6];
u.l.b[3]=i2c.in[7];
tach_pos_i2c = u.l.lval;
wheel_speed = tach_spd_i2c; //0.148749652 *
console_txdata.rover_speed = tach_spd_i2c;
// Serial.println("M, TachSPD:" + String(int(tach_pos_i2c)));
}

void error_check() //Checks the vehicle's systems for
errors (currently the only errors being checked for are temperature light, under voltage, and
letting the console know that the vehicle is armed)
{
if(digitalRead(temp_warning) == HIGH) //Checks the digital pin that is tapped
into the temperature light on the dash (if the temperature light is on, the pin will read HIGH
)
{
console_txdata.temp_error = true;
ros_txdata.temp_warn = true;
temp = String("T"); //Puts a "T" in the designated temperature
warning place in error_string (indicating that the temperature light is on)
temp_count = temp_count + 1; //Counts iterations that the temperature
light has been on

if(temp_count == 1) //Ensures that the timer will start for the
temperature light when the light first comes on
{
temp_start_time = millis(); //Assigns the start time on the first
iteration that the temperature light has been on
}

temp_end_time = millis(); //Assigns the end time each iteration that
the temperature light is on for
temp_time = temp_end_time - temp_start_time; //Calculates the total time that the
temperature light has been on for

if(temp_time >= temp_time.limit) //Checks to see if the temperature light
has been on for longer than the designated temperature time limit
{
Serial.println("M, ESTOP DUE TO ENGINE TEMP");
estop_code=3;
}
}
}

```

```

e_stop_state = HIGH; //If the temperature light has been on for
                    longer than the temperature time limit, the emergency stop is activated
}
}
else //If the temperature light is off
{
console_txdata.temp_error = false;
ros_txdata.temp_warn = false;
temp = String("X"); //Puts a "X" in the designated temperature
                    warning place in error_string (indicating that the temperature light is off)
temp_count = 0; //Resets the counter
}
}

void check_voltage() //Function to check the on-
                    vehicle voltage levels
{
voltage_input = analogRead(Twelve_V_Voltage-pin); //Reads analog value from
Twelve_V_Voltage = voltage_input * .01468; //Scales voltage_input
voltage_input = analogRead(Twenty_Four_V_Voltage-pin); //Reads analog value from
Twelve_V_Voltage = voltage_input * .03205; //Scales voltage_input

if((Twelve_V_Voltage <= threshold_e_stop_12_v) || (Twenty_Four_V_Voltage <= threshold_e_stop_24_v)
) //Checks to make sure voltage levels are above emergency stop threshold levels and activates
emergency stop if they are not
{
Serial.println("M, ESTOP DUE TO LOW BATTERY");
Serial.println("M, 24V -> " + String(int(round(Twenty_Four_V_Voltage*1000)))+ "E-3");
Serial.println("M, 12V -> " + String(int(round(Twelve_V_Voltage*1000)))+ "E-3");
console_txdata.voltage_error = true;
estop_code=2;
e_stop_state = HIGH;
}

else if((Twelve_V_Voltage <= threshold_warning_12_v) || (Twenty_Four_V_Voltage <=
threshold_warning_24_v) //Checks to make sure voltage levels are above temperature light
warning threshold levels and activates temperature light if they are not
{
voltage = String("V"); //Puts a "V" in the designated
temperature warning place in error_string (indicating that one of the vehicle's systems is
under voltage)
console_txdata.voltage_error = true;
ros_txdata.voltage_warn = true;
}

else
{
voltage = String("X"); //Puts a "X" in the designated
temperature warning place in error_string (indicating that the vehicle's voltage levels are
not too low)
console_txdata.voltage_error = false;
ros_txdata.voltage_warn = false;
}
}

void aux_switch_parse() //Function to parse the
                        auxiliary switch string from console
{
// A_status = consol_input_string.substring(2, 3); //Status of rocker switch A on
// console("A" for on, "X" for off)
// B_status = consol_input_string.substring(3, 4); //Status of rocker switch B on
// console("B" for on, "X" for off)
// C_status = consol_input_string.substring(4, 5); //Status of rocker switch C on
// console("C" for on, "X" for off)
// D_status = consol_input_string.substring(5, 6); //Status of rocker switch D on
// console("D" for on, "X" for off)
// E_status = consol_input_string.substring(6, 7); //Status of rocker switch E on
// console("E" for on, "X" for off)
// F_status = consol_input_string.substring(7, 8); //Status of pushbutton F on
// console("F" for on, "X" for off)
// horn_status = consol_input_string.substring(8, 9); //Status of pushbutton H on
// console("H" for on, "X" for off)

A_status = console_rxdata.aux[0];
B_status = console_rxdata.aux[1];
C_status = console_rxdata.aux[2];
D_status = console_rxdata.aux[3];
E_status = console_rxdata.aux[4];
F_status = console_rxdata.aux[5];
horn_status = console_rxdata.aux[6];

ros_txdata.A = A_status;
ros_txdata.B = B_status;

```

```

ros_txdata.C = C_status;
ros_txdata.D = D_status;
ros_txdata.E = E_status;
ros_txdata.Horn = horn_status;
ros_txdata.F = F_status;

if(horn_status) //If the horn button on the
  console has been pressed, activate the horn relay
{
  digitalWrite(horn_relay_pin, HIGH);
}
else //If the horn button on the
  console has not been pressed, make sure the horn is off
{
  digitalWrite(horn_relay_pin, LOW);
}
}

void gear_change() //Function called when a gear change is desired
{
  desired_gear = console_rxdata.gear;

  if (desired_gear != current_gear) //Only changes if the desired gear
    and current gear are differnet
  {
    desired_speed = -500; //Sets up a string to sent to the
    Speed Controller to apply the brake while the gear is changed so that throttle will be zero
    and the brake will be applied for you if you are on a hill changing gears
    // command_type = String("C");
    // command_mode = String("A");
    // throttle_or_steer = String("V");
    // speed_string_to_send= space + command_type + comma + command_mode + comma + throttle_or_steer
    // + comma + desired_speed;

    // Serial3.println(speed_string_to_send); //Sends string to Speed
    Controller to apply brakes
    if (desired_gear == 'H') //If desired gear is High
    {
      gear_position = 1000; //Transmission actuator position
      associated with High
      e_brake_state = LOW; //Ensures emergency brake is off
      digitalWrite(ebrake_relay_pin, e_brake_state);
    }

    else if(desired_gear == 'L') //If desired gear is Low
    {
      gear_position = 103; //Transmission actuator position
      associated with Low
      e_brake_state = LOW; //Ensures emergency brake is off
      digitalWrite(ebrake_relay_pin, e_brake_state);
    }

    else if(desired_gear == 'N') //If desired gear is Neutral
    {
      gear_position = -449; //Transmission actuator position
      associated with Neutral
      e_brake_state = LOW; //Ensures emergency brake is off
      digitalWrite(ebrake_relay_pin, e_brake_state);
    }

    else if(desired_gear == 'R') //If desired gear is Reverse
    {
      gear_position = -1000; //Transmission actuator position
      associated with Reverse
      e_brake_state = LOW; //Ensures emergency brake is off
      digitalWrite(ebrake_relay_pin, e_brake_state);
    }

    else if(desired_gear == 'P') //If desired gear is Park
    {
      gear_position = -449; //Transmission actuator position
      associated with Neutral
      e_brake_state = HIGH; //Applies parking brake
    }
    else
    {
      desired_gear = current_gear;
    }
  }

  channel = 2;
  gear_string_to_send = position_prefix + space + channel + space + gear_position;
  Serial2.println(gear_string_to_send);
  // Serial.println(gear_string_to_send);
  // current_gear_function();
}

```

```

else //When the gear is done changing,
    the brake is released
{
    //desired_speed = 0; //Brake off and throttle at zero
    //speed_string_to_send= space + space + command_type + comma + command_mode + comma +
    // throttle_or_steer + comma + desired_speed;
    //while(Serial1.available()){Serial1.read();} //clear serial buffer
    //Serial3.println(speed_string_to_send); //Send the speed command on the
    // the Speed Controller
}
}

void current_gear_function() //Updates the current gear
{
    if(digitalRead(high) == HIGH) //If the high gear indicator
        light on the vehicle is on
    {
        current_gear = 'H';
    }

    else if(digitalRead(low) == HIGH) //If the low gear indicator light
        on the vehicle is on
    {
        current_gear = 'L';
    }

    else if(digitalRead(neutral) == HIGH) //If the neutral gear indicator
        light on the vehicle is on
    {
        if (e_brake_state == LOW) //If parking brake is off, and
            vehicle in neutral, the vehicle is simply in neutral
        {
            current_gear = 'N';
        }
        else if(e_brake_state == HIGH) //If parking brake is on, and vehicle
            in neutral, the vehicle is in park
        {
            current_gear = 'P';
        }
    }

    else if(digitalRead(reverse) == HIGH) //If the reverse gear indicator
        light on the vehicle is on
    {
        current_gear = 'R';
    }
    console_txdata.rover_gear = current_gear;
}

void control_command() //String consol_input_string
    //Called when the string from Vehicle Mega is a control command
{
    steering_position = console_rxdata.steering_pos;
    spd_txdata.speedmode = console_rxdata.speedmode;

    if(desired_gear == 'P')
    {
        desired_throttle = -800;
    }
    else
    {
        desired_throttle = console_rxdata.throttle_pos;
    }

    spd_txdata.throttle_pos = desired_throttle;
    //Serial.println(spd_txdata.throttle_pos);
    if(millis()-last_steering_sent > steering_cmd_rate){
        steering_string_to_send = position_prefix + " 1 " + steering_position;
        Serial2.println(steering_string_to_send);
    }

    if(abs(wheel_speed)<=1) //If the console input
        string is a gear change string and the absolute value of wheel speed is below 1 mph (Trying
        not to grind gears!)
    {
        gear_change();//consol_input_string; //
        Call gear change function
    }
}

boolean IsNumeric(String str) {
    for(char i = 0; i < str.length(); i++) {
        if ( !(isDigit(str.charAt(i)) || str.charAt(i) == '.') ) {
            return false;
        }
    }
}

```

```

}
return true;
}

void parseSteeringControllerFeedback()
{
if (Serial2.available()) //
    Read String from transmission and steering motor controller
{
string_from_motor_controller = Serial2.readStringUntil('\r');
}
if(string_from_motor_controller.startsWith("TR")) //If
    the string is a response to the steering position query
{
string_from_motor_controller = string_from_motor_controller.substring(3); //
    Parse the numerical value from the query response
if(IsNumeric(string_from_motor_controller) & string_from_motor_controller != "") {
act_steering_position = string_from_motor_controller.toInt(); //
    Convert this numerical value from a string to intiger
console_txdata.rover_steering = act_steering_position;
}
if(act_steering_position == 0)
{
Serial.println(string_from_motor_controller);
}
}

else if(string_from_motor_controller.startsWith("Starting")) //If
    the motor controller is executing its startup procedure
{
mc_state = string_from_motor_controller; //Set
    mc_state to "Starting"
console_txdata.armed_status = false;
}

else if(string_from_motor_controller.startsWith("Ready")) //
    When the motor controller is finished executing the startup procedure, the mc_state changes to
    "Ready"
{
mc_state = string_from_motor_controller;
digitalWrite(beacon_relay_pin, HIGH); //
    Lights up beacon to demonstrate vehicle is armed and ready
console_txdata.armed_status = true;
//Serial2.println("^TELS \"?TR:#200\"");
}
}

void loop()
{
int mark = millis();
tic = millis();
if(ETin.receiveData()) //
    If the rover recieves a valid packet from the console
{
// Serial.print("DATA! "); Serial.println(console_rxddata.packet_id);
e_stop_time_1 = millis(); //Record emergency stop start time (if contact with console is
    lost, the start time will stop updating itself)
}

// if(Serial.available())
// // If data is available on the USB Serial ... AKA from ROS
// {
//     if(A_status) // AND if
//         the "A" aux switch is ON
//     {
//         //ros_input_string = Serial.readStringUntil('\r');
//         //Process the serial buffer as a command string
//         //TODO: ROS COMMAND INTERPRETATION
//     }
//     else
//     {
//         while(Serial.available() > 0) {
//             //Flush the serial buffer so that when autonomous mode is switched on, no cached commands are
//             //executed
//             char t = Serial.read();
//         }
//     }
// }

//calculate the difference in time between our last console contact and now
//if this time exceeds *dead_man_timeout* then we need to estop the vehicle
//because we have lost contact with the console

e_stop_time_2 = millis(); //
    Record emergency stop end time
e_stop_time = e_stop_time_2 - e_stop_time_1; //
    Time between start and stop for dead man switch

```

```

if(e_stop_time >= dead_man_timeout) //If
    difference in time is greater than the dead-man timeout, toggle on the emergency stop system
{
    estop_code=1;
    e_stop_state = HIGH; //Indicates the emergency stop system is engaged
    Serial.println("M,ESTOP Due to Communication Timeout");
}

if(e_stop_state == HIGH) //If
    emergency stop state is HIGH, write low to the emergency stop relay to engage the emergency
    stop system
{
    ros_txdata.estop = true;
    digitalWrite(e_stop_relay_pin, LOW);
    Serial.println("M,ESTOP! Relay Set!");
}
else //
    Otherwise write the emergency stop relay pin HIGH to keep the emergency stop system off
{
    ros_txdata.estop = false;
    digitalWrite(e_stop_relay_pin, HIGH);
}

if(mc_state == "Ready") //
    Ensuring that the motor controller is not executing its startup procedure
{
    if(counter%5==0)
    {
        Serial2.println(steering_query); //Prints
        steering_query to transmission and steering motor controller
    }
    counter++;

    control_command();
    aux_switch_parse();
}

parseSteeringControllerFeedback();
readFromTach();
readFromSpd();
current_gear_function(); //
    Function call to determine the current gear
check_voltage(); //
    Function call to check voltages
error_check(); //
    Function call to check the vehicle for errors and either alert the console or activate the
    emergency stop
if(millis() - lastSendTime > MAX_XBEE_SEND_RATE) {
    lastSendTime = millis();
    ETout.sendData(); //
        Send data packet back to consol
    ETspd_out.sendData(); //Send data packet to the speed arduino
    sendFeedbackToROS();
}
// Serial.print("LT: ");
// Serial.println(millis()-mark);
}

void sendFeedbackToROS()
{
    //Packet format from Rover -> ROS
    // Comma delimited
    //1: R (char)
    //2: Timestamp (unsigned long): milliseconds since arduino epoch
    //3: Loop time (unsigned long): number of millieconds to complete loop
    //4: Actual Steering Position (int): steering position [-1000 1000] as reported by the motor
    controller
    //5: Commanded Steering Position (int): current steering setpoint (format same as actual steering
    pos)
    //6: Wheel speed: (float) Wheel speed in mph [-40,40]
    //7: Desired Speed: (float) setpoint from [-1000 1000]???? maybe?
    //8: Current Gear (char): [P,N,L,H,R]
    //9: Desired Gear (char): [P,N,L,H,R]
    //toc = millis();
    //feedback_to_ROS = "R," + String(millis()%10000) + comma + ((toc-tic)) + comma +
        act_steering_position + comma + steering_position + comma + (wheel.speed*1000) + comma +
        desired_speed + comma + current_gear + comma + desired_gear;
    ros_txdata.wheel_speed = wheel_speed*60.0/444.0; //converts to wheel revs
    ros_txdata.wheel_pos = float(tach_pos_i2c)/444.0; //convert to rpm
    ros_txdata.desired_throttle = desired_throttle;
    ros_txdata.desired_gear = desired_gear;
    ros_txdata.actual_gear = current_gear;
    ros_txdata.desired_steering = steering_position;
    ros_txdata.actual_steering = act_steering_position;
    ros_txdata.time = millis();
    ros_txdata.estop_code = estop_code;
    ros_txdata.engine_rpm = engine_rpm;
}

```

```

ETros.sendData();
//Serial.println(feedback_to_ROS);
}

```

## Vehicle Mega Arduino: comm\_definitions.h

```

// Structures to define binary communication protocols
struct CONSOLE_TO_ROVER{
int packet_id;
int steering_pos;
int throttle_pos;
char gear;
char speedmode;
bool aux[7];
};

struct ROVER_TO_CONSOLE{
int rover_speed;
int rover_steering;
char rover_gear;
bool voltage_error;
bool temp_error;
bool armed_status;
};

struct ROVER_TO_SPEED_ARDUINO{
int throttle_pos;
char speedmode;
};

struct SPEED_ARDUINO_TO_ROVER{
float engine_rpm;
};

struct TACH_TO_ROVER {
float spd;
long int pos;
};

struct ROVER_TO_ROS {
long time;
float wheel_pos;
float wheel_speed;
int desired_throttle;
char desired_gear;
char actual_gear;
int desired_steering;
int actual_steering;
boolean temp_warn;
boolean voltage_warn;
boolean estop;
boolean A;
boolean B;
boolean C;
boolean D;
boolean E;
boolean Horn;
boolean F;
int estop_code;
float engine_rpm;
};

```

## Vehicle Mega Arduino: config.h

```

//Configurations and thresholds

const int temp_time_limit = 20000; //Time in milliseconds to
    allow the temperature light to stay on before activating emergency stop (important safeguard
    to prevent serious engine damage)

const int threshold_warning_12_v = 11; //12 Volt threshold to
    give low battery warning on console
const int threshold_warning_24_v = 21; //24 Volt threshold to
    give low battery warning on console
const int threshold_e_stop_12_v = 7; //12 volt threshold to
    activate emergency stop
const int threshold_e_stop_24_v = 19; //24 volt threshold to
    activate emergency stop

```



```

const int Serial_timeout = 100; //Set the serial timeout
    for hardware serial ports

const int dead_man_timeout = 750; //If the Vehicle Mega
    loses contact with the console for more than the dead_man_timeout (in milliseconds), the
    emergency stop will be hit

const int steering_cmd_rate = 25;

const int MAX_XBEE_SEND_RATE = 80;

```

## Vehicle Mega Arduino: pin\_definitions.h

```

//This file defines the vehicle mega pinout

#define TACH_SLAVE_ADDRESS 0x60 //Slave address for the tach arduino for i2c
#define SPD_SLAVE_ADDRESS 0x61 //Slave address for the arduino speed controller

const int Twenty_Four_V_Voltage_pin = 8; //24 Volt system voltage
    input pin
const int Twelve_V_Voltage_pin = 9; //12 Volt system voltage
    input pin

const int temp_warning = 10; //Digital input pin for
    vehicle's temp warning light
const int reverse = 9; //Digital input pin for
    vehicle's reverse gear light
const int neutral = 8; //Digital input pin for
    vehicle's neutral gear light
const int low = 7; //Digital input pin for
    vehicle's low gear light
const int high = 6; //Digital input pin for
    vehicle's high gear light

const int ebrake_relay_pin = 2; //Emergency brake relay
    pin
const int horn_relay_pin = 3; //Horn relay pin
const int e_stop_relay_pin = 4; //Emergency stop relay pin
const int beacon_relay_pin = 5; //Relay pin for beacon on
    rollcage

const int wheel_speed_pin = 11; //PWM wheel speed pin from
    the Axle Tachometer Interpreter

```

## Environmental Sensing Unit: enviro.ino

```

#include "pin_definitions.h"
#include "comm_definitions.h"
#include <TimerOne.h>
#include <iDHTLib.h>
#include <EasyTransfer.h>

volatile int dustval = 0; //Analog reading of the dust sensor. Set in the software interrupt IRQ
volatile long cnt = 0; //Counts number of software interrupt timer iterations between 555
    timer input and taking the ADC reading

//DHT22 - Humidity callback declaration
void dhtLib_wrapper(); // must be declared before the lib initialization

//DHT22 - Humidity Lib instantiate
idDHTLib DHTLib(DHT_PIN, digitalPinToInterrupt(DHT_PIN), dhtLib_wrapper);

// East Transfer (communicate to ROS)
EasyTransfer ET;

// DHT Callback Function
void dhtLib_wrapper() {
    DHTLib.dht22Callback(); // Change dht11Callback() for a dht22Callback() if you have a DHT22
    sensor
}

// Hardware ISR triggered on the falling edge of the VLED signal
void callback()
{
    cnt = 0;
    Timer1.attachInterrupt(SW_IRQ);
}

```

```

// Software ISR triggered by the hardware ISR – for reading the dust sensor
void SW_IRQ()
{
  if(cnt > 4) // Wait 5 iterations
  {
    digitalWrite(DEBUG_PIN, HIGH); // Set output pin high (purely for debugging purposes)
    dustval = analogRead(A0); //Read the dust sensor signal in
    digitalWrite(DEBUG_PIN, LOW); // Set output pin low (purely for debugging purposes)
    Timer1.detachInterrupt(); // Disable software interrupt until the hardware interrupt is
      triggered again
  }
  cnt++;
}

void setup() {
  //Setup the Serial communications
  Serial.begin(115200); //Debug
  Serial.println("\nStartup");
  Serial.print("Begin Intialization ... ");
  // Initialize the timer for driving the fan PWM and software interrupts
  Timer1.initialize(40);

  //Register EasyTransfer
  ET.begin(details(packet), &Serial);

  //Set pin mode for diagnostic output for determining if we are reading the
  // dust sensor at the right time
  pinMode(DUST_SENSOR_DEBUG,OUTPUT);
  digitalWrite(DUST_SENSOR_DEBUG,LOW);

  //Setup the interrupt to catch the VLED line falling to start triggering the analog read process
  pinMode(DUST_SENSOR_PULSE_IN,INPUT);

  //Set the fan speed to 90% of maximum
  setFanSpd(90);

  //Wait 2 seconds to allow the DHT sensor to initialize
  delay(2000);
  Serial.println("DONE");
}

void loop() {
  //Read the humidity sensor
  readHumiditySensor();
  //Store the dust sensor value (set by interrupt)
  packet.dust_v = dustval;
  //Read the gas and temperature sensors
  readGasSensors();
  readTempSensor();
  //Send the data back to ROS
  ET.sendData();
  //Attach hardware interrupts, wait 1 sec and then detach to safely access volatile variables
  attachInterrupt(digitalPinToInterrupt(DUST_SENSOR_PULSE_IN),callback, FALLING);
  delay(1000);
  detachInterrupt(digitalPinToInterrupt(DUST_SENSOR_PULSE_IN));
}

void readHumiditySensor() {
  //Code more or less copied from the library example – minus some debug statements
  DHTLib.acquire();
  while (DHTLib.acquiring());
  int result = DHTLib.getStatus();
  if(result == IDHTLIB_OK)
  {
    packet.dht_humid = DHTLib.getHumidity();
    packet.dht_temp = DHTLib.getCelsius();
  }
}

void readGasSensors() {
  //Reading in all gas sensor values
  packet.MQ4 = analogRead(MQ4_PIN);
  packet.MQ135 = analogRead(MQ135_PIN);
  packet.MQ9 = analogRead(MQ9_PIN);
  packet.MQ2 = analogRead(MQ2_PIN);
  packet.MQ5 = analogRead(MQ5_PIN);
  packet.MQ6 = analogRead(MQ6_PIN);
  packet.MQ7 = analogRead(MQ7_PIN);
  packet.MQ8 = analogRead(MQ8_PIN);
}

void readTempSensor() {
  packet.TMP36 = (analogRead(TMP36_PIN) * (5000.0/1024.0) - 500)/10;
}

```

```

void setFanSpd( float percent ) {
//We need to supply a 25 kHz PWM signal and the standard analogWrite will only supply one at
    500Hz. The Timer1 library
//used below is able to supply the propper waveform.
Timer1.pwm(FAN_CTL_PIN, int(round(percent * 1024.0 / 100.0)));
}

```

## Environmental Sensing Unit: comm\_definitions.h

```

struct SENSOR_PACKET {
long timestamp;
float dht.temp;
float TMP36;
float dht_humid;
float dust_v;
float MQ4;
float MQ135;
float MQ9;
float MQ2;
float MQ5;
float MQ6;
float MQ7;
float MQ8;

} packet;

```

## Environmental Sensing Unit: pin\_definitions.h

```

const int DEBUG_PIN = 13;
const int DUST_SENSOR_AIN = 0;
const int DUST_SENSOR_PULSE_IN = 21;
const int DUST_SENSOR_DEBUG = 13;

const int DHT_PIN = 3;

const int MQ4_PIN = 1;
const int MQ135_PIN = 2;
const int MQ9_PIN = 3;
const int MQ2_PIN = 4;
const int MQ5_PIN = 5;
const int MQ6_PIN = 6;
const int MQ7_PIN = 7;
const int MQ8_PIN = 8;

const int TMP36_PIN = 9;

const int FAN_CTL_PIN = 11;

```

# Appendix I: Safety Protocol

## VEHICLE OPERATION RULES

### General Rules:

1. When operating the vehicle, one person must have a cell phone with enough battery to make a call. In case of emergency, call 911.
2. The vehicle is not street-legal. Do not drive on public roads.
3. Confirm explicit permission from the property owner before operating the vehicle on private property.
4. If trailering the vehicle, use proper loading and strapping procedures.
5. If operating in a parking lot or near buildings or people, use traffic cones to block off a testing zone.
6. Two people must be present when operating the vehicle. Always assign someone to watch surroundings for possibly safety concerns such as people walking or driving by. This person must be ready to stop and direct traffic if necessary.
7. Do not drive the vehicle in small buildings or confined spaces. The exhaust will quickly become hazardous.
8. Do not drive the vehicle when you are impaired. Do not drive when you are too tired, stressed or hurried to drive carefully.
9. Never annoy or distract the vehicle driver or operator unless there is a safety risk.
10. Do not operate the vehicle unless you have been properly trained and approved to do so.
11. Do not leave the vehicle unattended.
12. Always wear the seatbelt while driving or riding in the vehicle.

### Pre-Drive Checklist:

1. Visually check the tires for low pressure or damage. If necessary, consult the tire's specifications to check tire pressure (maximum pressure is marked on the sidewall).
2. Visually check the vehicle for fluid leaks. This includes:
  - a. Brake fluid
  - b. Differential or gear oil
  - c. Motor oil
  - d. Gasoline
3. Briefly test the horn.
4. Describe intended testing methods to all who are present; this will make operation malfunctions much easier to notice.
5. Confirm that all components and wiring are properly secured, especially near the wheels. No cables should be hanging down from the vehicle at any point.
6. Release all emergency stop buttons on the vehicle and console.
7. Power on the console and the vehicle electronics and wait for the ARMED light to come on. This signifies that the vehicle is ready to take commands and has gone through its power-up sequence before the motor is started.
8. Shift the transmission into Neutral (or Park on the console) before starting the motor.
9. Apply the pedal brake when starting the vehicle. The choke may need to be adjusted to start the vehicle and maintain idle speed. If this is not a familiar process, seek help.

V2 3/28/15

## Safe Operating Rules

This document outlines the minimum Safety Standards for operating the RSL Rover on jacks while applying the throttle required by Santa Clara University and OSHA. RSL Rover safety should exceed these requirements. The purpose of RSL safety is to eliminate:

1. Accidents and Injuries
2. Property Damage
3. Equipment Abuses and Damage

Meeting these requirements requires understanding the system and the laws or regulations that may apply. These operating rules include all the rules required by OSHA, state OSHA and Santa Clara University. All operators need a copy of these Operating Rules and each operator must acknowledge them by signing that they have received a copy of these rules and understand them.


Training requires documentation. Undocumented training has the same basic legal effect as no training. The operator must acknowledge receipt of these rules for documentation to occur under OSHA, state OSHA and Santa Clara University requirements.

### OPERATING RULES:

1. Only operators authorized by the management, and trained in the safe operation of the RSL rover will be permitted to operate and test such vehicle. Methods will be devised to train operators in safe operation of an off-road vehicle. A minimum of three trained operators must be present in order to perform testing.
2. Initial operating procedures must be performed at least once before commencing testing. Attention will be given to the proper functioning of tires, horn, lights, battery, controller, brakes, and steering mechanism. All emergency stop systems must be tested and checked to make sure they are all working as intended.
3. A minimum of 6 jacks must be used in the raising of the vehicle. Place these jacks in locations that distribute the weight of the vehicle equally. Apply some force to the side of the vehicle and shake it. The vehicle should stay securely on the jacks and hardly move.
4. A secondary safety measure must be implemented. Tether the car with a chain to either a secure spot on the ground or a spot in the testing area. This location must be able to withstand the force of the weight of the car and/or the movement. This failsafe system will only activate if the vehicle happens to fall off the jacks.
5. The testing area must be clear of any objects within 2 feet of the vehicle. The operators/testers need to be at least 6 feet away from the vehicle at all times when the throttle is being used. Have one of the operators making sure no bystanders are within 50 feet of the test area.
6. The vehicle will not exceed a speed of 8mph which is approximately 20% of full throttle.
7. **NO RIDERS WILL BE PERMITTED ON THE VEHICLE.** A person may not ride on the rover due to the possibly instability/shaking that may occur.

# Appendix J: Conference Slides

SANTA CLARA UNIVERSITY



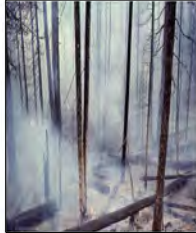
Aaron Burns - MECH  
Giovanni Briggs - COEN  
Hesham Naja - MECH  
Patrick Barone - MECH  
Zoe Demertzis - COEN

Advisor: Dr. Christopher Kitts

www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Post-Fire Risks



Responders exposed to...

- Carbon monoxide
- Carbon dioxide
- Low oxygen
- Air particulate
- Toxins from retardant
- Other carcinogens

Professor David Puseer  
www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Presentation Road Map

- Project Motivation
- Background
- Our Solution
- Subsystem Breakdown
- Requirement Verification




www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Problem Statement

Our goal is to design and implement an **unmanned vehicle** that **gathers and relays** information on potentially **hazardous environmental conditions** back to its operators.



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Post-Fire Investigation Protocol

```

    graph TD
      A[Determine fire contained] --> B[Environmental/air quality assessment]
      B --> C[Structural assessment/shoring]
      C --> D[Etc.]
  
```

National Fire Protection Association  
www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Customer Needs

Results of interviews with two fire fighters

Category	Specific Needs
Sensors	Imaging sensors Multiple gas detectors Air particulate Temperature
Communication	Strong communication link Clear data presentation
Vehicle	Rugged Temperature/weather resistant Travel long distances



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Current Technologies

	<p>Northrop Grummann's Andros F6</p> <ul style="list-style-type: none"> <li>+ 5 cameras</li> <li>- No housing for air quality sensors</li> </ul>
	<p>Sensefly's UAV EBee</p> <ul style="list-style-type: none"> <li>+ Thermal imaging/mapping</li> <li>- No ability to assess air quality on ground</li> </ul>

www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Legacy Work

- Drive by Wire
- Remote Operator Console
- Analog Override System
- Emergency Stop



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### RSL Rover Solution



Information to Mobile Command Center

Air Quality / Hazardous Gas Sensors

Camera Feeds

LIDAR Data


Information to Vehicle Operators

www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Five Subsystems

1. Environment Sensing
  - Cameras
  - Payload Sensors
2. Mapping
  - GPS
  - LIDAR Sensors
3. Operation/UI
  - Controllers/Electronics
  - Inertial Measurement Unit
4. Communications
  - Network Electronics
5. Power
  - Batteries



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Design Objectives


- Hazardous Environment Evaluation
- LIDAR and GPS Mapping Capabilities
- Dual Purpose Vehicle
- Store Data Streams
- Information Dissemination

www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Robot Operating System (ROS)

- All of the rover subsystems are integrated with ROS
- Industry Standard
- This provides pluggable software functionality
- Centralized data collection
- Enables subsystem testing with recorded data

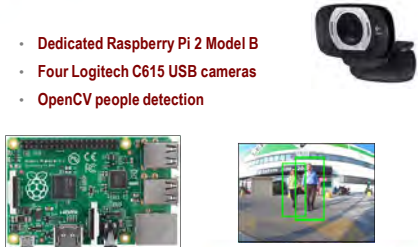



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Cameras

- Dedicated Raspberry Pi 2 Model B
- Four Logitech C615 USB cameras
- OpenCV people detection


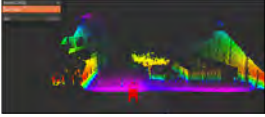



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Localization/Mapping

- Completed Navigation Sensor Integration
  - GPS
  - 2x LIDAR
  - IMU
  - Tachometer
  - Steering Sensor
- 3D LIDAR Visualization
- 2D Environment Mapping
- Complete System Data Acquisition

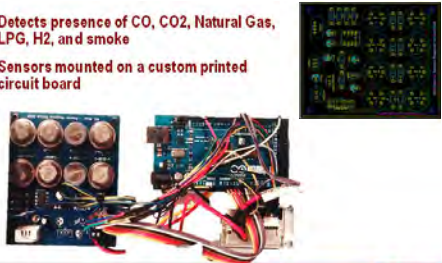



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Air Quality Sensing Unit

- Detects presence of CO, CO2, Natural Gas, LPG, H2, and smoke
- Sensors mounted on a custom printed circuit board



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### User Interface

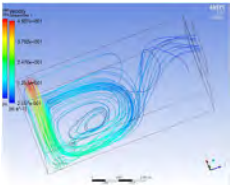


www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Sensor Housing

- Ensure that sensors receive appropriate airflow
- Multiple Iterations were analyzed
  - Some designs used fan as outlet
  - Single Inlet + Particulate Inlet
- Final Design
  - Fan Used as Inlet for smoothness
  - Double Outlet + Particulate Outlet



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Non-technical Considerations

- Economic
- Health & Safety
- Manufacturability
- Social
- Political
- Ethical



www.sccu.edu SCHOOL OF ENGINEERING



SANTA CLARA UNIVERSITY

### Testing Results



www.sccu.edu SCHOOL OF ENGINEERING


SANTA CLARA UNIVERSITY

### Latency Requirement Verification

- Tested latency of cameras and UI

Requirement	Result
Cameras latency < 1s	0.75s
UI camera stream latency < 1s	0.8s
UI vehicle state latency < 1s	2s

Vehicle Forward Camera

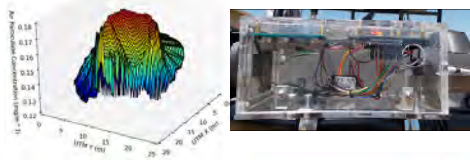


www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Environmental Sensor Requirement Verification

- Successfully tested the ability of sensors to detect gases and particulates caused by a fire




www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Range Requirement Verification

- Tested maximum distance between rover and driving controls

Requirement	Result
Wifi must reach 150m	250m
Driving control must work at 150m	> 1000m




www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Vehicle Blindspot Requirement Verification

- Evaluated the camera coverage and blind-spots


Requirement	Result
360 deg visibility	260 deg



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

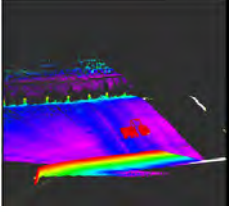
### GPS Requirement Verification



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Localization and Mapping Verification

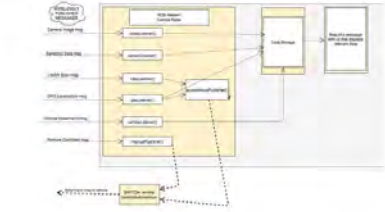


- 2D maps successfully generated while driving
- 3D point-clouds visualized while driving
- Accumulated 3D point-clouds assembled in post-processing

www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Software Architecture



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

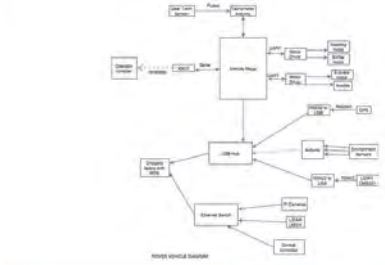
### Conclusion




www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### System Architecture



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

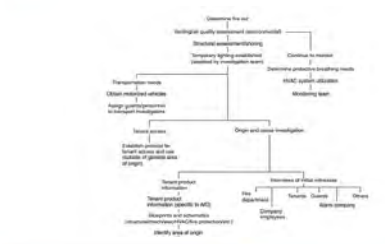
### Questions?



www.sccu.edu SCHOOL OF ENGINEERING

SANTA CLARA UNIVERSITY

### Post Fire Protocol



www.sccu.edu SCHOOL OF ENGINEERING

**SANTA CLARA UNIVERSITY**

**Requirements Flowdown**

**Vehicle Inspection: Document the requirements of the manufacturer's (supplier of equipment) on quality, temperature, and other data in a vehicle inspection form. (equipment manufacturer guidelines) (6/1)**

Requirement	Code	Activity
REQ 1.0: Document all information of the vehicle and all items used	ENR	Check Requirements
REQ 1.1: Document the location of the vehicle	ENR	Document vehicle information from vehicle - (ENR 100)
REQ 1.2: Provide all required data of the vehicle for driving	ENR	Conduct the maintenance of the data and items used
REQ 1.3: Check the vehicle for safety	ENR	Check for any items for safety or safety
REQ 1.4: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.5: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.6: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.7: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.8: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.9: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.10: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.11: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.12: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.13: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.14: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.15: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.16: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.17: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.18: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.19: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.20: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.21: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.22: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.23: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.24: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.25: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.26: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.27: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.28: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.29: Check for any items for safety	ENR	Check for any items for safety or safety
REQ 1.30: Check for any items for safety	ENR	Check for any items for safety or safety

**SCHOOL OF ENGINEERING**

**SANTA CLARA UNIVERSITY**

**References**

- Health Hazards of Smoke: Recommendations of the April 1997 Consensus Conference. Gen. Tech. Rep [GTR-383](#).
- MTDC. Missoula, MT: U.S. Department of Agriculture, Forest Service, Missoula Technology and Development Center.
- National Fire Protection Association. Guide for fire and explosion investigations. NASA Technical Standards, 2004
- Professor David Purner. Toxic hazards to fire fighters, including effects of the retardants during fires and post-fire investigation activities. 2009.

**SCHOOL OF ENGINEERING**