April 2019

Kinisi: A Platform for Autonomizing Off-Road Vehicles

Cassandra Marie Pepicelli
Worcester Polytechnic Institute

Cooper Wolanin
Worcester Polytechnic Institute

Gabriel Entov
Worcester Polytechnic Institute

Jonathan LongYing Tai
Worcester Polytechnic Institute

Lanhao Mao
Worcester Polytechnic Institute

See next page for additional authors

Author
Cassandra Marie Pepicelli, Cooper Wolanin, Gabriel Entov, Jonathan LongYing Tai, Lanhao Mao, and Samuel Stephen White

A Platform for Autonomizing Off-Road Vehicles

Completed by:

Gabriel Entov (ECE/RBE)
Lanhao Mao (RBE)
Cassandra Pepicelli (ECE)
Jonathan Tai (RBE)
Samuel White (RBE)
Cooper Wolanin (ECE/RBE)

Advised by:

Professor Alexander M. Wyglinski (ECE/RBE)
Professor Hugh C. Lauer (CS)

April 20th, 2019

A Major Qualifying Project Submitted to the Faculty of Worcester Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science.

Abstract

This project proposed a modular system that would autonomize off-road vehicles while retaining full manual operability. This MQP team designed and developed a Level 3 autonomous vehicle prototype using an SAE Baja vehicle outfitted with actuators and exteroceptive sensors. At the end of the project, the vehicle had a drive-by-wire system, could localize itself using sensors, generate a map of its surroundings, and plan a path to follow a desired trajectory. Given a map, the vehicle could traverse a series of obstacles in an enclosed environment. The long-term goal is to alter the software system to make it modular and operate in real-time, so the vehicle can autonomously navigate off-road terrain to rescue and aid a distressed individual.

Executive Summary

The US Department of Transportation says 94% of fatal crashes are due to human error. By removing the human element, it is proven that autonomous driving could dramatically reduce the number of vehicle accidents and human casualties. Over a span of 50 years, it is projected that over 600,000 lives could be saved with the use of autonomous vehicles. Additionally, on average, the weight of a car in America is 1880 kg, and roughly 74% of that weight is in metals. Manufacturing a car, autonomous or not, adds to 20% of the total carbon footprint of the world. As societies shift to autonomous cars, vehicles with autonomous capability would be manufactured as new vehicles. There is also the issue that there are already 1.2 billion vehicles deployed that are not already autonomous. If society is to change infrastructure to a completely driverless society, then immense resources are needed for new vehicle manufacturing and old vehicle disposal. Producing modular kits for autonomy would save lives, save time, save the environment, and would allow people to make the vehicles they already own autonomous.

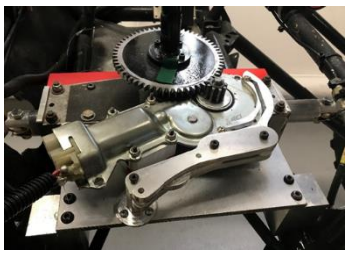
This project aimed to solve the previously mentioned safety and environmental concerns by developing an experimentation platform for autonomizing vehicles. To do this, the team developed a Level 3 autonomous off-road vehicle prototype. This platform can be used to help integrate autonomous vehicles into society without requiring the disposal of old non-autonomous vehicles or the creation of new autonomous vehicles. The platform also helps address the topic of off-road autonomy and can be used as a test bed for terrain that is much less predictable than urban environments.

Development Process

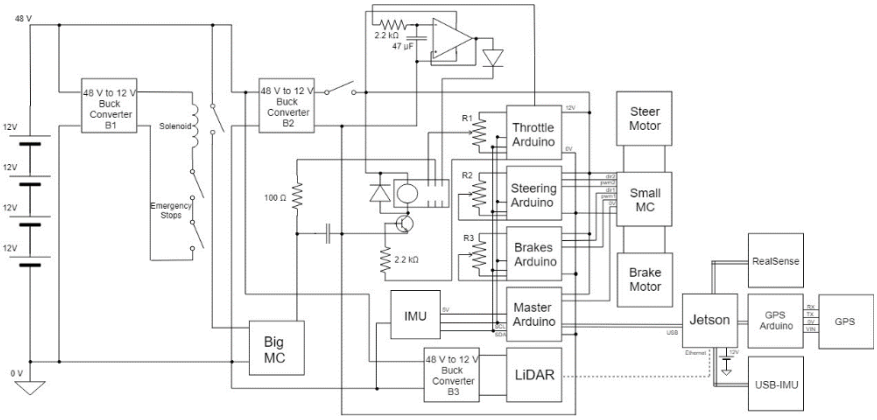
For a vehicle to operate independent of human input, three base functions must be addressed. The first function is the control behind the mechanical system. The vehicle must be able to mechanically accelerate, brake, and steer, while taking electrical signals as inputs. The second main function concerns the electrical system of the vehicle and sensors. The electrical system must be able to control the acceleration, braking, and steering of the vehicle as well as power the sensors. The array of sensors must obtain sufficiently detailed information to make calculated path planning and trajectory tracking decisions. The third function needed for a

vehicle to operate autonomously is software, which will process the sensor data and intelligently plan and execute actions in the physical world. A Main Computing Unit (MCU) is required to fuse the sensor data together, as well as display the data in real time, and send commands to the actuators.

Acceleration, braking, and steering, also categorized as the three major mechanical systems, were examined to determine the most efficient way to control them. The acceleration signal is entirely electrical, meaning it could be replicated as a signal from a microcontroller. The braking system required actuation of the physical brake pedal, so a linear actuator equipped with a potentiometer was used to control this system. Finally, the steering system was actuated using a rotational window motor that can be disconnected with a simple clutch. The shaft connecting the steering system to the steering wheel was equipped with a rotary potentiometer to detect the current steering position of the vehicle.



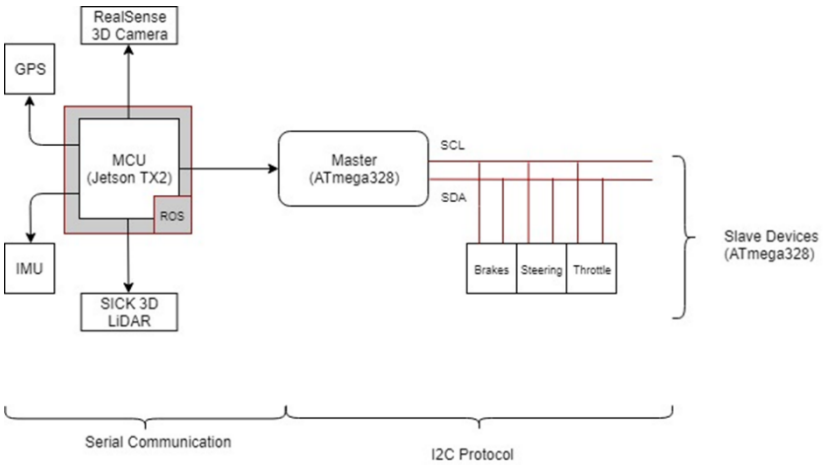
The initial electrical system of the car was bare bones; the vehicle was powered by four batteries, but none of the control or actuation electronics were designed or implemented. The team developed a schematic that incorporates the entire system for future teams to understand for debugging purposes, as well as to improve upon. The accelerator pedal was wired to an oscilloscope to investigate the electrical signal used to accelerate the vehicle. By tapping into the potentiometer signal line, a relationship was created between voltage and acceleration. The signal coming from the pedal is a DC signal, ranging from 0.8 V to about 5 V, however the maximum speed of the wheels occurs at approximately 3.3 V. The microcontroller that the team decided to use does not have a digital to analog converter (DAC) on board, so the team developed a simple DAC. The Pulse-Width Modulation (PWM) signal from the microcontroller was attenuated by



an active low pass resistor-capacitor (RC) filter. This signal was passed through an isolation amplifier that isolates the impedance of the pedal and the filter from the impedance of the motor controller. By connecting this signal to the motor controller, the team confirmed that the driven wheels of the vehicle could be controlled from computer commands. In order to maintain full manual operability of the vehicle, the team uses a relay to give the controls of the vehicle back to the user.

An Intel® RealSense™ Depth Camera, a SICK 3D LiDAR, a GPS module, and two consumer grade motion sensors were purchased and mounted to the vehicle for data collection purposes. An MCU is used to process the sensor array data and generate a path for the car to travel. This data processing and interpretation is done using various software packages and communicated with ROS, a community driven robotics platform. SLAM is used with a GPS and an IMU to assist with localizing the vehicle in its environment. Hybrid A* is the selected path searching algorithm as it satisfies the non-holonomic motion constraints of the vehicle and it generates optimal paths to desired goals. Hybrid A* allows the vehicle to iteratively generate paths to the map frontier's while exploring, until the vehicle reaches its final destination. ROS provides libraries, tools, and basic operating system services and is used as the primary framework of data communication between subsystems connected to the MCU.

To follow the desired path, the MCU sends commands to an array of microcontrollers that are used to actuate the vehicle accordingly. The message sent from ROS to the microcontrollers contains a SlaveID followed by data. Once the master node receives this message from the MCU, it commands the appropriate slave to move the actuator it is responsible for, to the desired position. The embedded control system is interrupt driven and runs in real time; it is able to process and command a slave before the next command comes from ROS. The vehicle and change direction of steering and brake suddenly just as a human would in an emergency situation.



Platform Performance

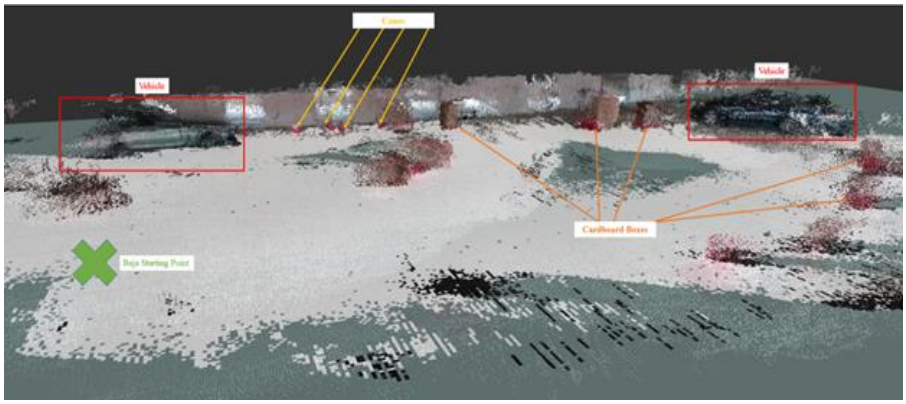
The integration of the mechanical systems, electrical system, sensors and software architecture involved extensive testing. The first feature that the team tested was the remote-control capability. Unfortunately, performance was not always consistent due to wires coming loose or parts breaking off the vehicle. Although a minor set-back, the team used any spare testing time as opportunities to collect ROS bags of sensor data, that could later be used in simulation and to test the software features in the lab. However, taking away from bugs and setbacks found at the testing site, the team could debug on the dolly inside of the Higgins MQP lab. These debugging sessions on the dolly allowed the team to continue making significant progress in between outdoor testing sessions. Once RC capability was sufficiently tested and the team was confident that a drive-by-wire vehicle had been created, the vehicle was given a pre-programmed path and could drive that path autonomously.

After this success, the team decided that it was time to add decision-making to the vehicle software capabilities. With the Jetson and its software communicating within its different subsystems using ROS, it was imperative that the team add the embedded control system as a "module" to the entire system. Commands sent to a TurtleBot platform were immediately translated and sent to the vehicle as well. This test was twofold: it showed a platform using this projects software in simulation, as well as demonstrated modularity. The same software was controlling a fourteen-pound robot, and a six-hundred-pound off-road vehicle.

Integrating the embedded control system with ROS allowed the team to integrate all of the systems together. The team quickly realized the Jetson was running out of computing power, and although the vehicle can simultaneously localize and map within the unknown environment, it does not have enough computational resources to simultaneously plan a path towards the goal and implement vehicle velocity control. It should be noted that this is not a function of hardware but rather a function of the implementation; the Jetson is capable of performing all of these tasks, but the team did not properly utilize its

GPU performance capabilities and did not program tasks in parallel. This was not implemented due to time constraints. To compensate for this, the team first made a map by manually driving the vehicle around obstacles the team set up.

Using the data from the sensors obtained from manually driving the vehicle through the aforementioned course, the team was able to create both a 3D point cloud and a 2D occupancy grid of the space. Using this 2D Occupancy Grid, the Hybrid A* path planning algorithm generates a path on this map to the goal destination. This path, which is an array of poses, avoids the obstacles that are in front and around the vehicle, and is sent to the velocity control software. The velocity control software works in conjunction with the localization software to determine the correct instruction set to send to the embedded control system, which ultimately moves the vehicle along the drawn path. The platform could use tuning depending on the vehicle it is operating on, but the vehicle is capable of autonomously navigating an environment given a map and is by definition, a Level 3 autonomous vehicle.



Acknowledgements

We would like to acknowledge the people who have aided in the successful completion of this senior project and shared their wisdom with us during the course of this experience. Their assistance made it possible to submit this finished report as a part of our degree requirements.

From Worcester Polytechnic Institute, we would like to show our gratitude to our project advisors, Professors Alexander Wyglinski and Professor Hugh Lauer for their timely feedback and recommendations on how to further develop this project and report. Both professors worked actively to provide the team with the resources and support needed to pursue our goals. We would like to further our gratitude towards Worcester Polytechnic Institute including the Electrical and Computer Engineering Department, the Robotics Engineering Program and the Computer Science Department. We thank the Mechanical Engineering Department for allowing us the use of their MQP lab space on campus to complete our project work.

We would like to thank the WPI Society of Automotive Engineers for their donation of the vehicle our project was completed on, as well as the use of their tools and services. This gratitude includes Professor David Planchard, the Advisor of the WPI Society of Automotive Engineers. We would also like to extend our gratitude to our project sponsors, Mitsubishi Electric Research Labs, Picotech, and SICK for their monetary and hardware contributions to this project. This project would not have been successful without their generous donations. In addition, we would like to thank Sharon Deffely, the Executive Director of Academic and Corporate Development at WPI. Her assistance streamlined the team's communication with our sponsors.

We would also like to thank Matthew Adiletta, Jarred Measmer and Joshua Dufault. Without their help, our project would not be where it is today. In this limited space, it is simply impossible to give the names of everyone who provided invaluable direction that assisted us in this project. We are highly grateful to everyone who guided us in completing this work. We sincerely thank all of you.

~The Baja MQP, 2018-2019

Table of Contents

List of Figures x

List of Tables xiii

List of Acronyms xiv

Authorship Page xv

Team Organization xvi

1: Introduction 1

1.1 Motivation 1

1.2 Current State of the Art 2

1.3 Technical Challenges 3

1.4 Addressing Societal Challenges 4

1.5 Report Organization 5

2: Evolution of Autonomous Vehicles 6

2.1 Defining an Autonomous Vehicle 6

2.2 Current Developments in Autonomous Vehicles 7

2.3 Summary 10

3: Project Outline 11

3.1 Requirements 11

3.2 Project Framework 12

3.3 Project Goals 14

4: Mechanical System Design 17

4.1 Research and Proposed Methodology 17

Introduction 17

Acceleration 18

Braking 19

Steering 24

4.2 Methods, Testing and Results 29

4.3 Summary 50

5: Electrical System Design 52

5.1 Research and Proposed Methodology 52

Cameras 52
Range Sensors 54
Ultrasonic Sensors 58
Global Position Systems 62
Motion Sensors 63
Main Computing Unit 67
Final Proposed Sensor Array 70
5.2 Methods, Testing, and Results 71
5.3 Summary 94
6: Software System Design 96
6.1 Research and Proposed Approach 96
Hardware Actuation Control and Sensor Data Acquisition 96
Sensor Fusion 98
Visual SLAM 101
Terrain Mapping 102
Path Planning 103
Trajectory Tracking 105
6.2 Methods, Testing and Results 106
6.3 Summary 124
7: System Integration & Results 126
8: Recommendations, Future Opportunities & Applications 131
Works Cited 134
Appendix A: Proposed Budget 144
Appendix B: CAD Model 145
Appendix C: Brakes: Linear Actuator Mount Drawings 146
Appendix D: Brakes: Linear Actuator Mount Finite Element Analysis (FEA) 159
Appendix E: Master-Slave Arduino Communication (master.ino) 170
Appendix F: Acceleration-Slave Arduino Code (slaveAccel.ino) 173
Appendix G: Braking-Slave Arduino Code (slaveLinAct.ino) 174
Appendix H: Steering-Slave Arduino Code (slaveSteering.ino) 176

List of Figures

Figure 1. Future Fatalities with/without Autonomous Vehicles (Reprinted from [1]) 1
Figure 2: Stages of Autonomous Vehicles (Reprinted from [14]) 7
Figure 3. The Giraffe (Left) & Panda (Right) Designed by Comma.AI (Reproduced from [20]) 10
Figure 4. Antilock Brake (Reprinted from [29]) 20
Figure 5: Measurement of Maximum Compression of the Hydraulic Brake Cylinders 21
Figure 6: Measuring the Force Required to Compress the Hydraulic Brake Cylinders 0.25” 22
Figure 7: Brake System Showing Location of Hydraulic Brake Cylinders 22
Figure 8: Electronic Power Steering Example (Reprinted from [32]) 25
Figure 9: Steer-by-Wire Example (Reprinted from [32]) 25
Figure 10: Fish Scale and 12” Long Wrench used to Measure Steering Input Torque 26
Figure 11: Measuring Steering Input Torque 27
Figure 12. Paper Miller TIG Calculator for Fillet Welds of ⅛-inch Steel 31
Figure 13. Miller TIG Welder Controls for Adjusting Weld Settings 32
Figure 14. Linear Actuator Mount for Brake Actuation as Shown from the Right Side 33
Figure 15. Linear Actuator Mounted to the Vehicle with Welded Steel Frame 33
Figure 16. View of the Linear Actuator Mount from the Driver’s Seat 34
Figure 17. Linear Actuator Mounted to the Vehicle as Shown from Above 34
Figure 18. Unmodified, Initial Design for Piston-to-Brake Pedal Interface 35
Figure 19. Travel Slot Lengthened to Keep Retaining Nut in Place Under Manual Braking 35
Figure 20. Demonstration that Under Manual Braking the Retaining Bolt does not Fall Out 36
Figure 21. Window Motor for Steering System 37
Figure 22. Gear Nomenclature (Reprinted from [35]) 38
Figure 23. Hand-drawn Four-bar Analysis 43
Figure 24. Four-bar Simulation Sketch 44
Figure 25. Four-bar and Window Motor Mounting Plate 45
Figure 26. Exploded Steering System (Left) and Completed Steering System (Right) 45
Figure 27. Steering System Machined Parts 46
Figure 28. Steering System Machined Parts Assembled 46
Figure 29. Top of Potentiometer Mount as Installed in Steering Rack 47
Figure 30. Bottom of Potentiometer Mount Next to Steering Rack 47
Figure 31. Potentiometer Interface 48
Figure 32. Potentiometer Interface Does Not Interfere with Steering Stop (Left) or Seal (Right) 48
Figure 33. Potentiometer Cradle and Gears 3D Design 49
Figure 34. Potentiometer Cradle and Gears Printed and Mounted 50
Figure 35. LD-MRS Time-of-Flight Measurement (Reprinted from [45]) 56
Figure 36. LD-MRS Four Scan Planes (Reprinted from [45]) 57
Figure 37. LD-MRS Scanning Range (Reprinted from [45]) 57

Figure 38. UC30-214 Detection Area (Reprinted from [51]) 60
Figure 39. UC30 Sensing Range (Reprinted from [52]) 61
Figure 40: Commonly Used PID Loop Control (Reproduced from [68]) 69
Figure 41. Sensor Breakdown of Baja Vehicle (Reproduced from [44]) 70
Figure 42. Baja Vehicle Outfitted with Sensors and Actuators from the Right Side 72
Figure 43. Baja Vehicle Outfitted with Sensors and Actuators from the Rear 73
Figure 44. Baja Vehicle Outfitted with Sensors and Actuators from the Left Side 73
Figure 45. Baja Vehicle Showcasing RealSense Placement 74
Figure 46. RealSense Depth Data (left) & Infrared Dot Projection on the Scene(right) 74
Figure 47. Vehicle Electronics Box Featuring IMU Module 75
Figure 48. Lab Wall and Bench in Higgins 75
Figure 49. Object Tracking and Partial Hallway Map Generation Using SLAM 76
Figure 50. RealSense and Custom Mount Model 76
Figure 51: LMS291 Data in Configuration 8m range, 180 deg field, 0.5 deg resolution 77
Figure 52: LMS291 Data in Configuration 80m range, 100 deg field, 1 deg resolution 77
Figure 53. SICK 3D LiDAR and Custom Mount Model 78
Figure 54. SICK Ultrasonics and Custom Side-Mount Model 79
Figure 55. SICK Ultrasonics and Custom Rear-Mount Model 79
Figure 56. Vehicle Electronics Box Featuring IMU Module 81
Figure 57. IMU/GPS Box Design 82
Figure 58. IMU/GPS Box Mounted to Vehicle 83
Figure 59. GPS Antenna Attached to Vehicle 83
Figure 60. Baja Vehicle Highlighting Placement of GPS/IMU Box and GPS Antenna 84
Figure 61. Active Low-pass Resistor-capacitor Filter with Diode 86
Figure 62. Relay Wiring Diagram Bottom View (Reproduced from [76]) 86
Figure 63. Switching Relay (Reproduced from [76]) 86
Figure 64. Version 1 vs. Version 2 of Perf Board 89
Figure 65. Vehicle Circuit with All Subsystems Wired to Emergency Stops 89
Figure 66. Vehicle Circuit with Actuation System Separated from Emergency Stops 90
Figure 67. Arduinos Powered Through DC Jacks 92
Figure 68. Jerk of an EV (Reproduced from [78]) 93
Figure 69. Vehicle Oscillations on Voltage Wires and Signal Wires 93
Figure 70. Low-Pass Filter for Signal Wire into Big Motor Controller 94
Figure 71. Final Full Vehicle Circuit 95
Figure 72. The Kalman Filter Gaussian Process (Reprinted from [89]) 100
Figure 73. System Flowchart: Data Processing to Vehicle Actuation 107
Figure 74. Data Fusion and Decision-Making Flowchart 108
Figure 75. Programming Structure 109
Figure 76. Communication between Sensors and Actuators 113
Figure 77. Manually Driven Baja Vehicle Path Taken at Gateway Garage on 4/15 123
Figure 78. Hybrid A* Generated Baja Vehicle Path (Green Line) 123

Figure 79. Software System Flow Diagram 125
Figure 80. Baja Vehicle Traversing Parking Lot with No Human Intervention 128
Figure 81. Obstacles Setup for Testing in Gateway Garage on 4/15 129
Figure 82. 3D Point Cloud of Obstacles Setup for Testing in Gateway Garage on 4/15 129
Figure 83. 3D Point Cloud & 2D Occupancy Grid of Obstacles Setup for Testing in Gateway Garage on 4/15 130
Figure 84: Baja Vehicle Chassis CAD 145

List of Tables

Table 1: Quantitative Decision Matrix for Braking Actuator 23
Table 2: Quantitative Decision Matrix for Steering Design 28
Table 3. Weld Setting Recommended for ⅛-inch Steel 32
Table 4. Measurements for Actuator Gear 38
Table 5. Derivations from Measurements for Actuator Gear 39
Table 6. Summary of Gear Properties 40
Table 7. Center Distance Calculations 40
Table 8. Tensile Strength and Shear Strength of Bolts 42
Table 9. Quantitative Decision Matrix for Choosing a Stereo Vision System 53
Table 10. Quantitative Decision Matrix for Ultrasonic Sensors 59
Table 11. Quantitative Decision Matrix for GPS Modules 62
Table 12. Quantitative Decision Matrix for Motion Sensor 65
Table 13. Static Orientation Accuracy Performance (from [60]) 66
Table 14: Dynamic Orientation Accuracy Performance (from [60]) 66
Table 15: MCU Decision Matrix 68
Table 16. MCU Class Design 110
Table 17. Occupancy Grid Class and Vehicle Snapshot Class 110
Table 18. Slave Class Design 111
Table 19. Pre-built Packages 112
Table 20: Budget 144

List of Acronyms

Acronym	**Definition**
ABS	Anti-lock Brake System
ADAS	Advanced Driver Assistance Systems
AHRS	Attitude Heading Reference System
ATV	All-terrain Vehicle
CAD	Computer Aided Design
CAN	Controller Area Network
CCD	Charge-Coupled Device
CCW	Counter-clockwise
CMOS	Complementary Metal–Oxide–Semiconductor
CPU	Computational Processing Unit
CW	Clockwise
DAC	Digital to Analog Converter
DCEN	Direct Current Electrode Negative
DGPS	Differential Global Positioning System
EKF	Extended Kalman Filter
EPS	Electronic Power Steering
FEA	Finite Element Analysis
GBFS	Greedy Best First Search
GPS	Global Positioning System
HPS	Hydraulic Power Steering
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	The Institute of Electrical and Electronics Engineers
IGVC	Intelligent Ground Vehicle Competition
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Range
MCU	Main Computing Unit
MEMS	Microelectromechanical Systems
MQP	Major Qualifying Project
P Controller	Proportional Controller
PD Controller	Proportional Derivative Controller
PID Controller	Proportional Integral Derivative Controller
PWM	Pulse-Width Modulation
RC	Resistor-Capacitor
ROS	Robot Operating System
SAE	Society of Automotive Engineers
SLAM	Simultaneous Localization and Autonomous Mapping
SPI	Serial Peripheral Interface
TIG	Tungsten Inert Gas
WPI	Worcester Polytechnic Institute

Authorship Page

This Major Qualifying Project proposal document has been contributed to by six undergraduate students. The team has been organized into subgroups based on the vehicle design and each members' strengths. The contributions of team members to the writing of this document are summarized below.

Chapter 1: Introduction

Gabriel Entov, Cassandra Pepicelli, Jonathan Tai, Samuel White, Cooper Wolanin

Chapter 2: Evolution of Autonomous Vehicles

Gabriel Entov, Cassandra Pepicelli, Jonathon Tai, Cooper Wolanin

Chapter 3: Project Outline

Gabriel Entov, Cassandra Pepicelli, Samuel White

Chapter 4: Mechanical System Design

Gabriel Entov, Samuel White, Cooper Wolanin

Chapter 5: Electrical System Design

Gabriel Entov, Lanhao Mao, Cassandra Pepicelli, Jonathan Tai, Samuel White, Cooper Wolanin

Chapter 6: Software System Design

Gabriel Entov, Lanhao Mao, Cassandra Pepicelli, Jonathan Tai, Samuel White, Cooper Wolanin

Chapter 7: System Integration

Gabriel Entov, Lanhao Mao, Cassandra Pepicelli, Jonathan Tai, Samuel White, Cooper Wolanin

Chapter 8: Recommendations, Future Opportunities & Applications

Gabriel Entov, Cassandra Pepicelli, Jonathan Tai

Works Cited:

Cassandra Pepicelli

Team Organization

The table below shows each team member, their major(s) and minor(s), and their year of graduation.

Name	Major(s)	Minor(s)	Graduation Year
Gabriel Entov	Electrical & Computer Engineering, Robotics Engineering		2019
Lanhao Mao	Robotics Engineering	Business and Computer Science	2019
Cassandra Pepicelli	Electrical & Computer Engineering	Business	2019
Jonathan Tai	Robotics Engineering		2019
Samuel White	Robotics Engineering		2020
Cooper Wolanin	Electrical & Computer Engineering, Robotics Engineering		2019

1: Introduction

1.1 Motivation

Autonomous vehicles are seen to be the future of transportation. “Nearly 37,500 Americans died on the roads [in 2016] […] The US Department of Transportation says 94 percent of fatal crashes are due to human error” [1]. By removing the human element, it is proven that autonomous driving could dramatically reduce the number of vehicle accidents and human casualties. Removing the human element could also greatly increase traffic efficiency, as well as allow for the piloting of search and rescue missions where humans cannot physically travel [1]. Figure 1 shows the amount of fatalities that would occur per decade from 2020 to 2070, both if the future involved autonomous vehicles versus if it did not involve autonomous vehicles. Over a span of 50 years, it is projected that over 600,000 lives could be saved with the use of autonomous vehicles.

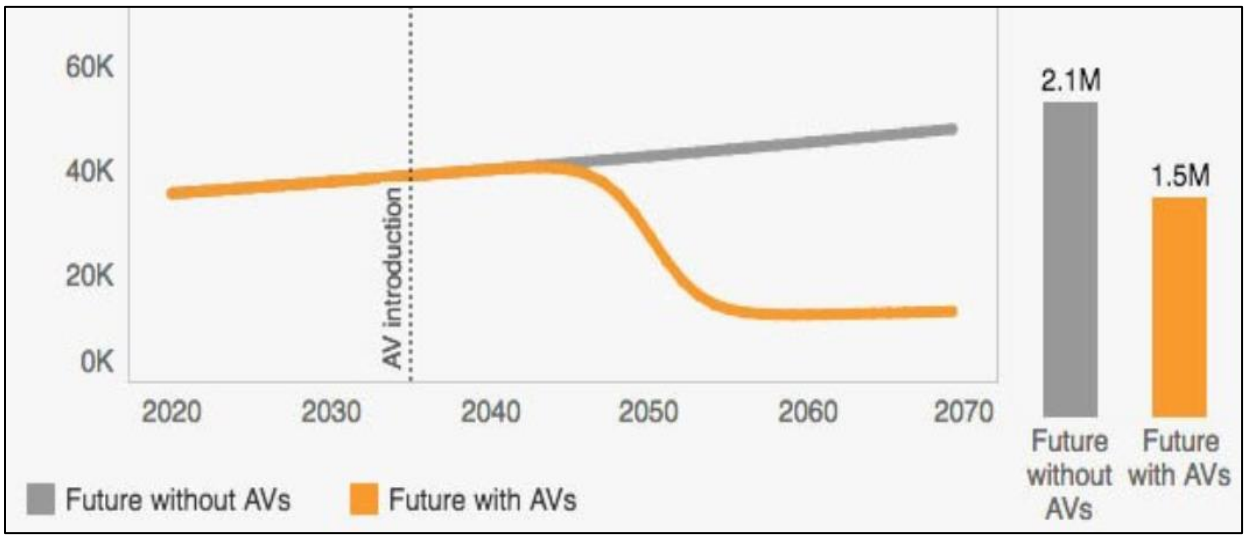


Figure 1. Future Fatalities with/without Autonomous Vehicles (Reprinted from [1])

However, for autonomous driving to become a mature integrated concept, various challenges need to be overcome. One of the challenges of autonomous driving is navigating on *off-road terrain*; navigating autonomous vehicles on *urban terrain* has already been solved. Compared to urban terrain, off-road terrain is much less predictable. In urban environments,

roads are generally predictable due to sufficient treatment and strict standards. Off-road environments do not have the same predictability. Due to uneven surfaces, lack of consistent landmarks, and a multitude of obstacles in the natural environment, off-road terrain offers a more challenging autonomous driving problem. Solving off-road autonomy will allow self-driving vehicles to operate safely and reliably in non-ideal and non-structured environments.

Another challenge of autonomous driving concerns the ability to implement it commercially on a wide scale in a society dominated by human-operated cars. A rising popularity of self-driving cars could have negative environmental impacts. A car is made up of many components, specifically a host of metals, alloys, rubbers, plastics, textiles and other materials [2]. On average, the weight of a car in America is 1880 kg [3], and roughly 74% of that weight is in metals [4]. In fact, according to Professor Mike Burners Lee from Lancaster University, manufacturing a car, autonomous or not, adds to 20% of the total carbon footprint of the world [5]. As societies shift to autonomous cars, vehicles with autonomous capability would be manufactured as new vehicles. If society is to change infrastructure to a completely driverless society, then immense resources are needed for new vehicle manufacturing and old vehicle disposal. Producing modular kits for autonomy would save time and would be safer for the environment. A modular kit would allow people to make the vehicles they already own autonomous.

This project aimed to solve the problem of autonomous driving with two questions. First, how can autonomous driving be integrated into a society that already has 1.2 billion vehicles deployed [6]? Second, driving in a city and driving in an off-road environment are two very different situations. How could this project create an autonomous vehicle that can navigate the vastly different environment that off-road driving provides? Before answers to these questions could be crafted, the team needed to research the current state of the art of off-road autonomous vehicles.

1.2 Current State of the Art

The current state of the art of off-road autonomous vehicles can be exemplified by two projects: Jaguar Land Rover [7] and Polaris [8].

Jaguar Land Rover is working on the problem of autonomous driving in off-road conditions [7]. They are looking more at unpaved road conditions and creating a vehicle that

would be able to navigate a safari autonomously. Jaguar uses a technique of data analysis called *terrain mapping* that takes the sensor data from the vehicle and creates a three-dimensional visual map of the terrain in front of the vehicle. This Jaguar project is primarily a research effort designed to incorporate communication between vehicles in the future such that one vehicle would be able to lead a pack of other Jaguar Land Rover vehicles through off-road terrain.

Polaris, an American manufacturer of all-terrain vehicles (ATVs), and other off-road vehicles, released a new autonomous dune buggy in February 2018, the MRZR X [8]. The vehicle was primarily designed for military use such that it could carry a 1500 payload and seat six individuals. It has been outfitted with technology in partnership with Applied Research Associates and Neya Systems for autonomous capability and a follow-leader mode. Its unmanned capabilities will enable the vehicle to assume a variety of roles from resupply to logistics support and evacuation. It was also designed such that if future improvements to the system are to be made, the vehicle can be easily upgraded, thus showing a form of modularity.

The Jaguar Land Rover and Polaris tackle two technical challenges of off-road autonomous vehicles: understanding the terrain around the vehicle and knowing what sensors are needed to enable autonomy in different environments. These technical challenges are discussed further in the following section.

1.3 Technical Challenges

The team considered several challenges around designing an off-road self-driving vehicle before starting the project.

1. The Terrain: The greatest challenge lies in understanding the terrain around the car. The terrain not only includes the surface that the vehicle is driving upon, but also foliage and other obstacles that may impede the vehicles movement.
2. The Sensor Array & Data Collection: Using sensors to collect accurate data from the terrain, interpret the data, and produce a data image that can be both understood and processed is a difficult technical challenge. The kinds of sensors that are necessary to create an appropriate view around the vehicle, as well as where the sensors would be placed in relation to each other needs to be considered.
3. Path Planning: Once the terrain can be analyzed, the vehicle needs to be able to navigate in its surroundings. Path planning algorithms need to be developed for the vehicle to be

able to move optimally. There are no existing algorithms that can be adapted or directly used for path planning; algorithms with specific requirements need to be developed and tested.

4. Obstacle Avoidance & Decision Making: With a map of the terrain created by the sensors and the path planning algorithm, the next challenge is to recognize and avoid obstacles in the route of the vehicle. This is also referred to as decision making and requires extensive and advanced programming to function successfully.
5. Modularity: To develop autonomous kits that serve the purpose of making any vehicle autonomous, the kits and software would need to be *modular*. The software would need to take in various vehicle dimensions and adapt data collection based on the new information of the vehicle.

These are the immediate challenges that helped direct the team's focus towards a comprehensive solution for making an off-road autonomous vehicle, specifically focusing on the societal impact of autonomous integration and its use in off-road environments.

1.4 Addressing Societal Challenges

The team's solution is designed to tackle two societal impacts: large-scale autonomous integration in society and autonomous vehicles in off-road environments. To reduce the potential environmental impact of fully integrating autonomous vehicles into society, the team proposed a modular system of sensors and actuators that accumulate into a Main Computing Unit (MCU). A modular autonomous system would allow various existing vehicles to gain autonomous driving capability and thus push forward society's transformation into a fully autonomous transportation community with least modification needed and minimal resources wasted.

To tackle the challenge of autonomous vehicles in off-road environments, this project team used a *Baja vehicle* owned by the WPI Society of Automotive Engineers. "Baja SAE consists of competitions that simulate real-world engineering design projects and their related challenges. Engineering students are tasked to design and build an off-road vehicle that will survive the severe punishment of rough terrain" [9]. Any vehicle created for this competition is coined a "Baja vehicle." The name of the competition originated from the SCORE Baja 1000, a Mexican off-road race on Mexico's Baja California Peninsula. The WPI chapter of SAE generously lent the team their Baja vehicle for this project. The vehicle available to this project

was built in 2006 and modified several times before being converted into an electric vehicle. This project implements a modular system design onto the Baja vehicle such that it can autonomously traverse various environments in an off-road transportation context. Details on the proposed solution can be found in Chapter 3.

1.5 Report Organization

This document explains in detail the situation and challenges of the project, the original project framework for these design challenges, the team's completed work, tests of that work, and results. In Chapter 2, the current state of autonomous driving is explained and evaluated through research of current autonomous driving companies and university research projects. In Chapter 3, the requirements of the project and the project framework is summarized, followed by the implemented design of the actuators and sensors. In Chapter 4, a detailed design procedure to transform a human operable off-road vehicle into a fully electric vehicle is explained, backed up with extensive research. Chapter 4 also explains the team's completed mechanical work. In Chapter 5, the team explored the potential sensors for the vehicle to gain information from the environment. This chapter includes a tutorial, research and evaluations of various situatable sensors. The work done with these sensors, as well as with the microcontrollers and microprocessors that control the actuators and sensors, is also detailed in Chapter 5. In Chapter 6, the intended algorithms used for autonomous driving and software infrastructure to achieve system modularity are explained. Chapter 6 also explains the software development work that the team has completed. Chapter 7 discusses the integration of the mechanical, electrical and software systems and the results of the implementation. Chapter 8 discusses the future opportunities and applications of this project.

2: Evolution of Autonomous Vehicles

Although a relatively new field, autonomous vehicles and general autonomy concepts have grown in relevance and popularity globally. “A new study predicts the value of the global autonomous vehicle market will grow more than tenfold from 2019 to 2026” [10]. Companies working towards developing autonomous cars include major businesses such as Tesla and Waymo. These companies believe autonomous vehicles are the future of safe driving environments and increased traffic efficiency, while also allowing an opportunity for off-road search and rescue missions [11]. The research documented in this chapter helped direct the team towards a design solution for this project, which is summarized in Chapter 3 and demonstrated in more detail in Chapters 4-6.

2.1 Defining an Autonomous Vehicle

The first mention of autonomous vehicles was on December 8th, 1926, in the Milwaukee Sentinel. This was referred to as the “Phantom Auto” and was controlled by radio from a car behind it. This was teleoperation, not real autonomy; a human was still controlling the car, just indirectly. The use of artificial intelligence in autonomous cars was first demonstrated at Carnegie Mellon University in the 1980s in their NavLab [12]. The field of artificial intelligence, even specific to autonomous vehicles, has grown since then. After over forty years, this application is ready for regulation. DARPA organized “The Grand Challenge” in 2007, where autonomous vehicles navigated their way through a desert over a 130-mile stretch. Since the DARPA challenge, Google has made incredible strides towards the autonomous car. Until 2016, the car had been known as “Google’s Self Driving Car,” but seven years after Google began research on self-driving cars, the company was rebranded to Waymo [13].

This evolution of autonomous vehicles led to the development of stages for defining different levels of autonomy. Figure 2 displays the various levels of autonomy, ranging from a fully manual vehicle to a fully autonomous one [14].

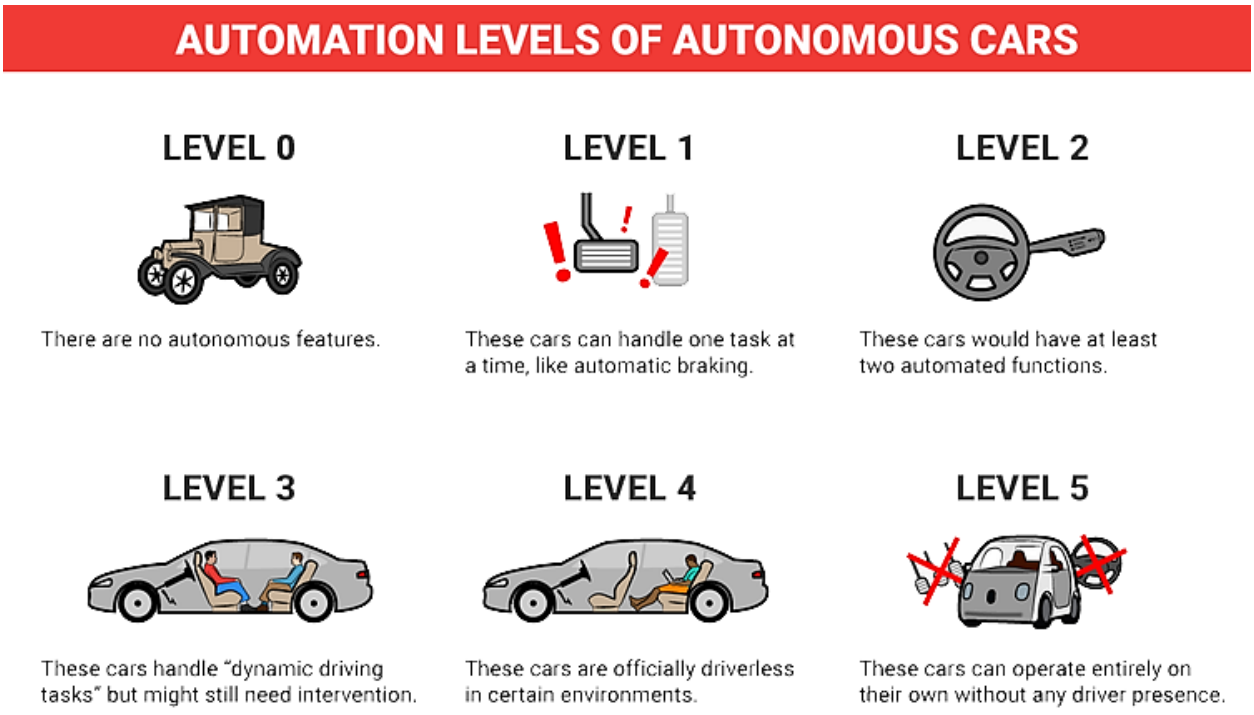


Figure 2: Stages of Autonomous Vehicles (Reprinted from [14])

This MQP team's project proposed a *level three autonomous vehicle*: a vehicle that can complete dynamic driving tasks but might need human intervention for certain vehicle functions. To determine what is necessary to create a level three autonomous vehicle, the team needed to research the current state of autonomous vehicles, and what the current state of the art is doing to be successful.

2.2 Current Developments in Autonomous Vehicles

In order to develop a level three autonomous vehicle, this MQP team researched the organizations that are making major strides in the autonomous industry. These organizations included Tesla, Waymo, the Prometheus 2010 MQP, and Comma.AI. Information relevant to this MQP from these organizations is summarized in this chapter.

Tesla

Tesla's Autopilot system is made up of multiple sensors placed around the car [15]. These sensors help the car understand its environment and navigate most highway situations. These sensors include a forward-looking radar, a camera, sonar, and a Global Positioning System

(GPS). The radar can see up to 500 ft (152.4 m) ahead of the car [16]. Radar is the primary sensor used to detect the surroundings of the vehicle, along with the front-facing cameras. A forward-facing camera on the windshield of the car serves as a backup to the radar. In addition to the radar and the camera, a 360-degree, ultrasonic sonar is used to detect nearby obstacles. The 12 ultrasonic sensors that make up the sonar can see small objects like children or animals and are functional at any speed. A GPS feature allows the car to detect its position on the road. A recent feature in Tesla vehicles called, "Tesla nav," allows the driver to automatically change lanes and exit the freeway. This feature also provides real-time data for precise localization and high-definition lane information [16].

Waymo

One of the primary leaders in autonomous car development is Waymo, a subsidiary of Google's parent company, Alphabet. Waymo originated as a Google project in 2009 and spun off into a separate subsidiary company in 2016. Waymo has a simple mission statement: "to make it safe and easy for everyone to get around—without the need for anyone in the driver's seat" [17].

Waymo drove over 300,000 miles autonomously in 2012, only three years after their debut. They started with simulation driving, eventually moving onto public roads and city streets, and released their first fully autonomous car in 2017 [17]. As their cars evolved, so did the sensors. The current Waymo vehicles use three major sensors: a LiDAR, a camera vision system, and a radar system, as well as supplementary sensors including GPS and an audio detection system. These sensors allow the Waymo car to detect obstacles in all directions at any time of the day, and at distances up to 1080 ft (329.184 m) [18].

Software development is an integral part to Waymo's system. Waymo focuses on three major aspects: perception, behavior prediction, and planning [18]. The perception component can identify objects on the road from the sensors data, while behavior prediction is looking to examine the trajectory and intent of each object detected. Planning gathers information from all other aspects to make decisions.

Prometheus MQP 2010

The Prometheus vehicle was an intelligent ground vehicle built in 2010 as a Worcester Polytechnic Institute (WPI) Major Qualifying Project (MQP) [19]. The autonomous vehicle was

developed to compete in the 2010 Intelligent Ground Vehicle Competition (IGVC). The competition specified minimum and maximum speed limits, size requirements, emergency shut offs, etc. Each autonomous vehicle had to be able to navigate an obstacle course of 800 ft (243.84 m) and avoid random objects while staying within painted lane markings. Vehicles had to travel between specified coordinates in less than five minutes and at less than five miles per hour [19].

"The vehicle used an array of sensors which included a differential GPS receiver, a digital compass, cameras, and a light detection and range (LiDAR) device" [19]. The project used a Differential Global Positioning System (DGPS) to determine the current geographic location of the vehicle, as well as a digital compass to determine the strength of the magnetic fields around the vehicle. The project used stereo vision to create a representation of the surrounding environment. Specifically, the project examined the difference between Complementary Metal–Oxide–Semiconductor (CMOS), Charge-Coupled Device (CCD) and 3CCD sensors. The project considered using an inertial measurement unit (IMU) but ultimately decided against it. The Prometheus project also included the use of a LiDAR to detect obstacles. At the end of the project, the Prometheus vehicle received the Rookie of the Year Award. This project and sensors used aided in research and selection of the team's sensors [19].

Comma.AI

Comma.AI is a company that uses a brain module, made up of the Giraffe Connector and Panda Connector, as seen in Figure 3, and the user's smartphone to control an individual's car autonomously [20]. There is an optional dash cam product that can replace the smartphone as the main computation unit and camera sensing unit. The system uses machine learning algorithms and the data it collects to create a more robust self-driving experience. The Comma.AI software is open source and encourages users to communicate on cloud-based team collaboration sites such as Slack and GitHub. In fact, the Comma.AI system suggests that it is targeted more towards a hacker community because of its multiple debug boards, data outputs, and analysis outlets [20].

Their system is only compatible with recent cars from specific brands, namely Toyota and Honda. The Giraffe Connector itself plugs directly into data busses and communication lines already present in many models. Functionally, it passes on signals from already implemented

sensors in modern vehicles to the main calculation unit and opens access to ports and information that would otherwise be inaccessible to the owner of the vehicle. This however does apply some limits to the autonomous functionality of the car based off of what pre-built sensors the car contains. Comma.AI claims that any other vehicle that is not of the Toyota or Honda brand would also be able to use their Panda and Giraffe modules to achieve self-driving capability. Comma.AI encourages their community to develop with their hardware [20].

Figure 3. The Giraffe (Left) & Panda (Right) Designed by Comma.AI (Reproduced from [20])

2.3 Summary

This chapter presented an overview of general key stakeholders in the autonomous vehicle industry. It examined the evolution of and previous developments in the autonomous field and defined "autonomy," as well as the various levels of autonomy. The most important takeaways from this chapter are the various sensors implemented in each stakeholder's autonomous project. Tesla uses a forward-looking radar, a camera, sonar, and a GPS. Waymo uses a LiDAR, a camera vision system, and a radar system, as well as supplementary sensors including GPS and an audio detection system. The Prometheus MQP used a differential GPS receiver, a digital compass, cameras, and a LiDAR. Comma.AI provides a method of turning a car autonomous using a smartphone and pre-built sensors in the vehicle, however this is not guaranteed to be applicable to all vehicles, and not all vehicles may have a complete sensor suite. Noting the similarities in the sensor suites of each of these organization's systems, this overview established a clear idea of what needed to be researched in order to design a sensor suite for this MQP.

3: Project Outline

3.1 Requirements

This Major Qualifying Project focused on two problems with current autonomous vehicles: the inability to operate in off-road environments and the inability for the system to retrofit existing vehicles (modularity). A modular autonomous system for off-road vehicles would tackle the different environmental issues off-road driving encounters, while providing a solution to adding autonomous driving technology to existing vehicles. To complete this project, the vehicle needed to be able to navigate off-road geographic areas and operate autonomously through a modular software system.

In order to create solutions to these two problems, the team created a set of requirements as follows:

- The team needed a vehicle that could drive off-road.
- The team needed a vehicle that could be operated entirely through electrical signals.
- The vehicle needed to operate at the third level of autonomy (explained in Figure 2).
    - The vehicle could operate on its own and interpret its environment but may need human intervention.
- The vehicle needed an array of sensors that enable it to “see” and interpret its surroundings from all sides and know its location.
- The software needed to be capable of operating under a specified set of input parameters, making it a modular system.
    - The software would process the inputs and influence the computations accordingly.

Upon analyzing these requirements, the team researched and proposed an array of sensors and processors that enabled the creation of an off-road, autonomous vehicle whose system is designed for modularity. This array of sensors and processors includes:

- One camera for front-facing short-range object detection.
- One range sensor for front-facing long-range object detection.

- Five or more ultrasonic sensors for side and rear-facing object detection.
- One motion sensor for measuring acceleration, velocity, and position in both translation and rotation.
- One global positioning system for location detection.
- One or more main computing units for controlling the entire system.
- Suite of distributed embedded controllers for controlling the actuation of the vehicle.

3.2 Project Framework

To address the need of navigating off-road environments, the team needed a vehicle already able to function in difficult terrains. This team considered two options available on the WPI campus. The first option was an electric golf cart that, while not designed for off-road driving, did have progress from previous MQP teams towards self-driving capabilities [21] - [22]. The second option was the electric SAE Baja vehicle which had no autonomous capabilities but was originally designed for driving off-road. The team decided that it would be more difficult to add off-road navigation capability to the golf cart than it would be to add autonomous capability to the electric Baja vehicle. This distinction resulted in the choice of the Baja vehicle as the platform for this project.

For a vehicle to operate independent of human input, three base functions must be addressed. The first function is the control behind the mechanical system. The vehicle must be able to mechanically accelerate, brake, and steer, while taking electrical signals as inputs. Most current vehicles are not mechanically setup for full autonomous control. The core mechanical functions of a vehicle - moving, stopping, and turning - require the driver of the vehicle to press the accelerator pedal, press the brake pedal, or turn the steering wheel. When the human driver is pressing a pedal or turning the steering wheel, they are manually actuating the mechanical systems. Setting a vehicle up mechanically for autonomous control means installing actuators that allow a computer to electrically command the vehicle to move, stop, or turn. The vehicle exists in a physical space and has mechanical components. Therefore, actuators are necessary to enable computer-control of the vehicle's functions. With regards to safety, the electro-mechanically actuated systems must always be able to be physically overridden by the driver.

The second main function concerns sensors, which will gather information about the environment and vehicle. The array of sensors must obtain sufficiently detailed information to make calculated decisions. Since the primary challenge of this project is autonomous navigation and path planning, sensors for "seeing" obstacles are necessary. The team chose to use a camera, a LiDAR, and ultrasonic depth sensors. The research documented in Chapter 2 supports the team's decision that this array of sensors, combined, would create the necessary autonomous driving capability.

The camera is needed to provide a *3D point cloud* of data [23], which will be used for detecting exactly what an object is so that the vehicle can be programmed to decide how it should traverse the object. The LiDAR is necessary because it allows for higher resolution data and faster sample times than the camera and assists in situations where poor weather conditions render the camera less useful for object detection. The combination of these primary sensors can generate large amounts of data representing what the vehicle sees. The sensors act as the eyes of the vehicle, and the integration of them allows the team to program the vehicle to avoid what they detect in the path of the vehicle. LiDAR and cameras are also used together in most autonomous vehicle projects (reference Chapter 2).

In addition to LiDAR and cameras, ultrasonic sensors are needed on the sides and back of the vehicle for additional obstacle avoidance. Ultrasonics are not as precise in determining the shape of an object, however they can serve as an additional detection method in situations when the vehicle is turning or backing up. An Inertial Measurement Unit (IMU) and GPS are also required for localizing the vehicle while allowing for less error and more precise positioning. This array of primary and secondary sensors is an optimal configuration of price and functionality that enhances the autonomous capabilities of the car.

The third function needed for a vehicle to operate autonomously is software, which will process the sensor data and intelligently plan and execute actions in the physical world. A Main Computing Unit (MCU) is required to fuse the data together, as well as display the data in real time, and send commands to the actuators. This project relies on a microprocessor for heavy computation and individual microcontrollers for the actuation of the vehicle controls. Although sensors enable the vehicle to see, software allows the vehicle to think, or act on the data it acquires. The software for this project has three functionally different parts:

- User Interface and Control
    - Embedded Programs are low level programs that communicate directly to the hardware on the vehicle.
- Path Planning
    - Refers to different algorithms that the vehicle uses to navigate from the starting point to the end goal.
- Sensor Fusion (Localization and Mapping)
    - The way that the vehicle will combine data from multiple sensors to improve the probability and reliability of both object detection and localization.
    - The way that the vehicle outputs world maps for path planning.

The software portion of the project is also required to create the modularity of the system. The team developed an interface that allows the user to select which vehicle is being used: the Baja vehicle or an alternate one. If the user chooses the Baja vehicle, no additional setup will be required, and the system will be ready to go. However, if the user selects an alternate vehicle, several values will need to be supplied to the software to correctly process inputted data for the new vehicle. The ability for the system to be placed onto another vehicle is what makes it modular.

At the end of the team's year of work, the Baja vehicle was outfitted with the proper actuators and sensors and had the necessary processing power and algorithms onboard to autonomously navigate off-road vehicles, while allowing for modularity.

3.3 Project Goals

In the fall of 2018, the team laid out the project goals. By the end of the third quarter of this academic year, also known as C-term, the goals of the project in consecutive order are as follows:

- The acceleration, braking, and steering of the vehicle will be able to be remote controlled. The software will have some integrated connections and access to ROS. Each sensor will have the primary coding done.

- The vehicle will be able to drive in simulation. This will not demonstrate modularity, it will operate on the Baja vehicle parameters. The software will collect and correct data.
- The vehicle will be able to use sensor data to autonomously drive in a straight line and turn away from objects.
- The software will be able to represent the modularity of our system through user inputs. The vehicle will be able to drive **in simulation** using parameters of other vehicles. The physical vehicle will be able to navigate from point A to point B based on location (GPS) input and correct path according to obstacles.
- The vehicle will be able to navigate to a given location autonomously while avoiding a series of obstacles. The location will be in a parking lot designed to demonstrate the ability to travel over different levels of terrain. The success of the project will be demonstrated on project presentation day via a video.

The team also laid out the goals for the Mechanical Design, Electrical Design and Software Design separately. For the Mechanical Design, the goals are as follows:

- The vehicle will be altered to include two new systems that would essentially replace human-operated braking and steering.
- The added mechanical systems will be simple to implement and will not inhibit necessary human actions, including braking and steering.
- The added mechanical systems will be safe and could be overridden by human control.
- The added mechanical systems will be reliable and could not be broken by any forces, stress, weather conditions, etc.

For the Electrical Design, the goals are as follows:

- The throttle of the vehicle will be electronically actuated, with the ability to switch between manual driver-mode and autonomous mode.
- The vehicle will be mounted with an array of sensors that will allow it to “see” a 360-degree picture of its surroundings.

- The vehicle will be mounted with the necessary sensors to localize itself in its environment.
- The vehicle will always know the location of the braking and steering systems.
- The acceleration, braking and steering systems will be controlled by a series of microcontrollers, a motor controller and a computer.

For the Software Design, the goals are as follows:

- Using software, the team will be able to acquire data from multiple mounted sensors and combine it together into a single 3D point cloud world map.
- Using software, the team will be able to use visual odometry and positioning sensors to accurately localize the vehicle in real time. The team will also be able to use the location of the vehicle to project the occupancy grid, representing where the vehicle can and cannot travel based on both the terrain and the obstacles detected from the sensors.
- Using software, the team will be able to plan and update a feasible path from the current location to a final location using the generated occupancy grid.
- Using software, the team will be able to transform and communicate the generated path into a set of time-dependent instructions in order to have the vehicle follow the desired trajectory.
- The software on the embedded system network will interpret these commands and send signals to each of the actuators on the vehicle in order to move it accordingly. This same system, through software, will also consider the grade of the slope the vehicle is driving on to better control its velocity.
- Using software, the team will be able to process sensor data and send vehicle commands to the embedded systems at 10 Hz.
- Using software, the team will be able to iteratively explore and navigate the unknown map while avoiding obstacles until reaching the final goal.

4: Mechanical System Design

The first aspect of the project that needed to be addressed to take a manual driver-operated vehicle and make it an autonomous one, was the mechanical actuation of the vehicle. To turn the Baja vehicle into an autonomous one, all the major driver inputs needed to self-controlled. The Baja vehicle, when originally given to the team in March 2018, did not have any method for autonomous movement, so actuation devices needed to be added to be able to move the vehicle without human control. Actuation devices were needed on the three major human inputs: acceleration, braking, and steering. The Baja vehicle also required additional maintenance to bring it into operable condition, including batteries, fuses, and brake fluid.

4.1 Research and Proposed Methodology

Introduction

The Baja vehicle uses four deep cycle batteries (p/n: WKDC12-100P) connected in series. Deep cycle batteries are discharged close to their maximum capacity [24]. The cost of these 12V, 100Ah batteries was approximately $175 each and according to the WPI's Society of Automotive Engineers (SAE), all four batteries needed to be replaced. Without batteries to power the vehicle, the vehicle would not be able to drive, which made investigating the batteries a stepping stone to being able to test the actuator systems.

Electric vehicles use the stored energy in rechargeable batteries to power electric motors that move the vehicle. Over time, rechargeable batteries lose maximum capacity, thus charging batteries of different capacities in series is problematic [25]. The charging system used is setup to charge batteries individually, meaning this is not a limiting factor for the Baja vehicle. However, during discharge the difference in capacity would be a problem because the battery with less capacity would be depleted before the other battery in series. This problem would cause the battery with the higher capacity to start to charge the lesser capacity battery, reversing the polarity and causing damage [26].

The Baja vehicle battery chargers require a 15A automotive blade type fuse for each positive charging lead [27]. While testing the batteries, the project team found that only one

battery seemed to be in critical condition due to a reversal of polarity [25]. SAE was requested to replace the faulty battery and the battery in series with it, which means a total of two batteries need to be replaced.

The brake system needed DOT 4 brake fluid for the rear brakes. DOT 4 brake fluid is a specific grade of brake fluid with specific boiling point specifications. After adding brake fluid to the rear brake fluid reservoir, the rear brakes were bled. This was accomplished by pressing on the brake pedal to compress the fluid in the brake lines with the master cylinder and then releasing the pressure at the bleeder valve on the rear brake caliper. The procedure was repeated until no air bubbles were in the fluid coming out of the bleeder valve. The rear brake fluid reservoir was kept full during the procedure to avoid sucking air into the brake lines. The brake fluid reservoir was topped off once the bleeding of the brakes was completed.

Aesthetically, the Baja vehicle was covered in dried dirt, had chipped black paint, and was missing several body panels. While the appearance of the Baja vehicle does not impact its functionality, it is more productive to work on a clean car. Additionally, the appearance will matter once testing the vehicle outside of the laboratory begins as it can be seen by the public.

This project focused on sensor fusion and algorithmic aspects of autonomy, but the project also presented a distinct mechanical design and manufacturing challenge in the solution. The team needed to outfit the Baja vehicle with actuators to control the steering, braking, and acceleration, as well as sensors used to gather proprioceptive and exteroceptive data. *Proprioceptive* data is information about the vehicle status, such as steering angle, position, and velocity. *Exteroceptive* data is information about the environment, such as camera images.

Acceleration

One of the three standard automotive driver controls is the accelerator pedal. Pressing the accelerator pedal makes the vehicle accelerate, while letting off the accelerator pedal makes the vehicle stop accelerating. There are many ways to implement control of the first main driving input: acceleration. *Drive-by-wire* is a term for replacing the physical cables or hydraulically operated automotive systems with electronically operated systems [28]. Essentially, the human foot is being replaced.

Traditionally the throttle on a gasoline engine vehicle is operated via a cable connecting the accelerator pedal to the throttle body [28]. Cars with computer-controlled fuel injection may

have a sensor that tells the computer the throttle position while still retaining the physical coupling between the pedal and the throttle [28]. Many modern cars have what is called "electronic throttle control," which removes the physical coupling between the pedal and the throttle [28]. This is the most reliable and safe drive-by-wire system because the fail-safe mechanism for the electronic throttle control system is similar to the mechanical cable fail-safe mechanism [28].

In the mechanical system, a spring closes the throttle if the cable breaks. In the electrical system, a faulty reading or no reading condition of the accelerator pedal can be designed to close the throttle. While some sensor redundancy is needed to validate the signal, the implementation of the electronic throttle control is relatively simple. Fully electric cars are required to have an electronic accelerator because there is no physical throttle.

The Baja vehicle, an electric vehicle with no power-assisted brakes or steering, required actuating the core controls while retaining manual operability. Essentially, the vehicle still needed to be manually operable by a human. The position of the accelerator pedal is monitored by a sensor. That data is fed into the motor controller, which in turn operates the electric motor that powers the vehicle.

The acceleration of the vehicle is easy to control since it is electronically powered. The signal coming from the throttle pedal can be interrupted to replace the pedals signal with that of the autonomous driving signal. In non-autonomous modes, the throttle pedal signal can continue to the motor controller un-altered.

Braking

The braking system is the second major automotive driver control. Pressing the brake pedal slows the vehicle down when it is in motion and keeps it from moving when the vehicle is at rest. Typically, when the brake pedal is pressed, pressure is applied to the master cylinder, which then applies pressure to the brake calipers on each wheel through the brake lines [28]. The braking pressure is amplified by a vacuum or hydraulic brake booster on most vehicles to reduce the effort required to press the brake pedal [28]. The baja vehicle does not require the braking pressure to be amplified because the vehicle is light weight. When actuating the brakes with electronics, one option was to retain a physical hydraulic coupling between the brake pedal and the brakes. The other option was to remove the physical hydraulic coupling, and instead use

electronics to interpret the brake pedal position and appropriate brake force to actuate. The team decided to leave the hydraulic braking system in place and to actuate hydraulic cylinders with an electromechanical device.

An anti-lock brake system (ABS) is an electro-hydraulic system (shown in Figure 4) that allows the brakes to be controlled without driver input by having electronic actuator control the brake pressure to the wheels [28]. Using ABS still allows the driver to have full control over the brakes when the system is not active. Many driver aids such as traction control, electronic stability control, and automatic braking are all based on ABS technology [28]. The options that eliminate physical connection between the brake pedal and the brakes have a sensor on the brake pedal and an actuator to operate the brake mechanism. This can be done by electro-hydraulic braking, where an electric actuator controls the pressure transmitted to each brake caliper through the brake lines. Alternatively, electromechanical braking has electronic actuators located inside the braking mechanism of each wheel; no hydraulic brake fluid is used.

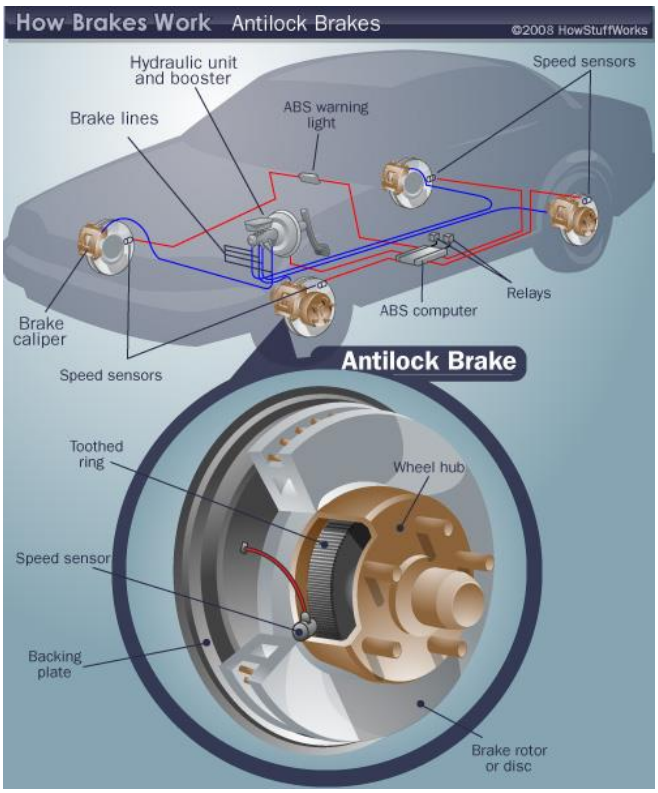


Figure 4. Antilock Brake (Reprinted from [29])

The electric motor can be used for up to 80% regenerative braking. Regenerative braking uses the inertia of the vehicle to rotate the wheels and electric motor, creating a large back EMF

that charges the batteries and in turn slows the vehicle down. Despite the ability to use regenerative braking, it is still necessary to actuate the hydraulic disk brakes while retaining the human operated brake pedal. This is because the human is needed for safety.

One solution was an electric braking pedal allowing acceleration and deceleration to be completely electronic. This would require relocating the hydraulic cylinders and brake reservoirs, replacing and rerouting brake lines, actuating two hydraulic cylinders, and installing an electronic pedal and braking logic for various modes.

An alternate solution was to leave the hydraulic braking system in place, use regenerative braking in specific conditions, and actuate the hydraulic cylinders for mechanical braking with an electromechanical device. To accomplish this, an actuator that produces a linear force with a small displacement can be and was used. The displacement is dependent of the stroke of the hydraulic brake cylinders, which is less than one-half inch (Figure 5). The force required to compress the hydraulic cylinders 0.25” was measured to be approximately 30 pounds. This measurement was taken using a 50-pound fish scale while measuring the displacement of the hydraulic brake cylinder piston. Figure 6 shows the measurement and Figure 7 the entire braking system. Compressing the hydraulic cylinders 0.25” is 50% of the maximum compression of the hydraulic brake cylinders for this vehicle. Brake force was measured at 0.25” because the force required to continue compressing the brakes past a quarter inch increases substantially. The force required to compress the brake to 0.5” is estimated to be between 80 and 120 lbs. of force.

Figure 5: Measurement of Maximum Compression of the Hydraulic Brake Cylinders

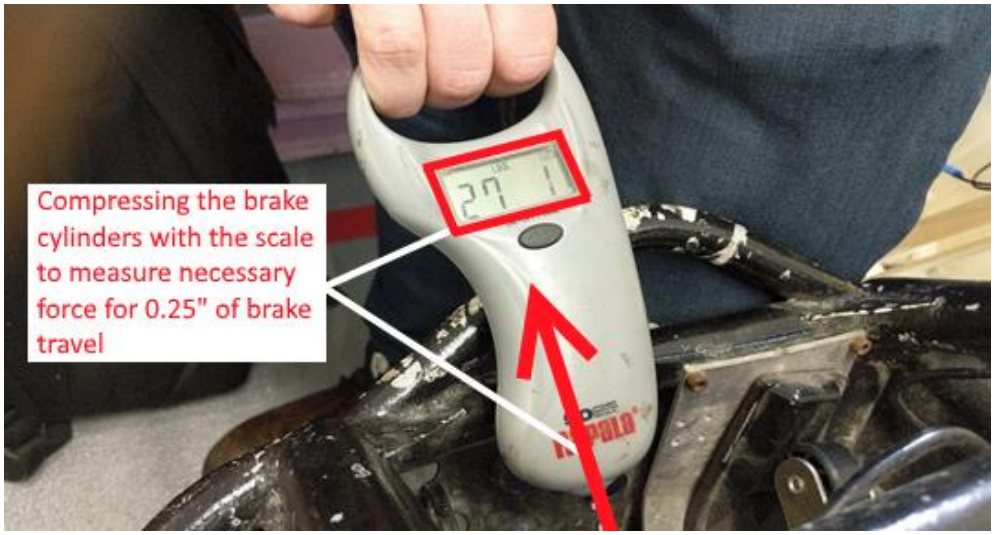


Figure 6: Measuring the Force Required to Compress the Hydraulic Brake Cylinders 0.25”

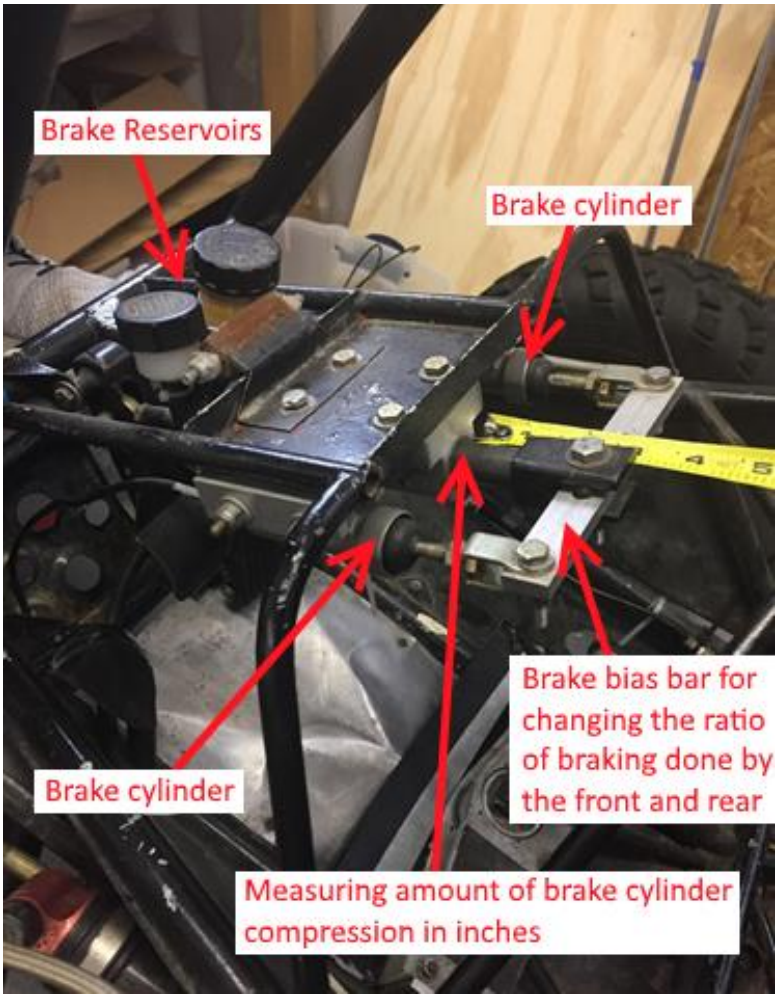


Figure 7: Brake System Showing Location of Hydraulic Brake Cylinders

Another braking requirement for actuating the vehicle was having a quick response time. This design consideration was important because the vehicle needs to be able to apply brakes quickly in an emergency. The ability to hold brakes compressed for long durations as well as modulate the brakes with a reasonably high duty cycle are important considerations when examining the reliability and safety of the brake actuation. Finally, in the case of emergency or electrical system failure, it was necessary to retain the ability of a human to override the computer and apply the brakes with adequate force; this requires a pedal. The options for actuating the brakes simply include a linear actuator and a lever-arm or cam driven by a motor. Table 1 contains a quantitative representation of the decision metrics.

Table 1: Quantitative Decision Matrix for Braking Actuator

Design Requirements	**Weight**	**Linear Actuator**	**Lever-Arm or Cam Driven by Electric Motor**
Controllability	**10**	10	3
Speed	**8**	6	9
Position holding	**8**	10	1
Duty cycle	**9**	7	4
Force	**10**	10	4
Cost	**7**	4	7
Implementation difficulty	**7**	7	3
Total	**1**	468	256
Percent total	**1**	79%	43%

The final decision was to use a sufficiently powerful linear actuator while allowing manual override ability. The linear actuator offers a controllable braking force application that can indefinitely hold its position. Additionally, the speed is sufficient for the short stroke necessary for this application and the force is more than adequate. The downside to using a linear actuator is that it costs more overall. However, the lever-arm or cam solution is less controllable than the linear actuator and would likely produce a non-linear brake force application. Additionally, the motor torque demands are much too high without designing a transmission for mechanical advantage.

Steering

The third and final major driver-controlled system for a vehicle is the steering wheel. Turning the steering wheel clockwise points the wheels of the vehicle to the right. Turning the steering wheel counterclockwise points the wheels of the vehicle to the left.

The industry standard for power steering has been hydraulic power steering (HPS) since 1951 [30]. HPS works by using a hydraulic pump driven by the engines serpentine belt to pressurize fluid. The steering wheel operates a control valve that adjusts how much assistance and in which direction the pressurized fluid helps move the rack, therefore reducing the effort to turn the steering wheel [31]. The benefits of HPS other than its successful history is that it provides road feedback. The major drawback of HPS is that constantly driving the hydraulic pump hurts performance as well as fuel mileage. Another drawback is that the steering feel is set and cannot be adjusted on the fly depending on the driving mode [30]. For example, putting a vehicle in "sport mode" could change the steering so that it feels heavier and tighter, in turn transmits more feedback from the road surface.

Electronic power steering (ESP) is the transitional technology between fully mechanical power steering (HPS) and fully electric power steering ("Steer-by-wire"). EPS uses a torque sensor attached to the steering input shaft to sense which way the driver wants to turn the wheel; an electric motor then drives the steering input shaft or the rack itself [31]. Steer-by-wire has a position sensor on the steering wheel and an electric motor attached to the rack that drives a ball-screw mechanism. A steering feel emulator provides steering feedback to the driver [28]. EPS and steer-by-wire reduce parasitic loss of performance and fuel mileage and allow the ability to dynamically change the steering feel [30]. Steer-by-wire requires the steering feedback to be balanced depending on the forces at the rack and the speed of the vehicle [32]. This means having more comfortable steering through more power assistance at low speeds when the force required at the rack are greatest. At higher speeds, the steering needs to have much less power assistance because slight changes in steering have large impacts because the force at the rack is much less. EPS and steer-by-wire both allow other driver assist systems to be implemented, such as lane keep and automatic parking [32]. A disadvantage of steer-by-wire systems is the lack of accurate feedback from the road due to poor tuning of the steering emulator [30]. Another complication is the safety and reliability of the system given that the driver must be able to control the vehicle in the event of any electrical fault. In a system with no mechanical fail safe,

such as with EPS or HPS, steer-by-wire systems must have redundancy in sensors, electrical processors, and actuators to maintain safety and reliability [32].

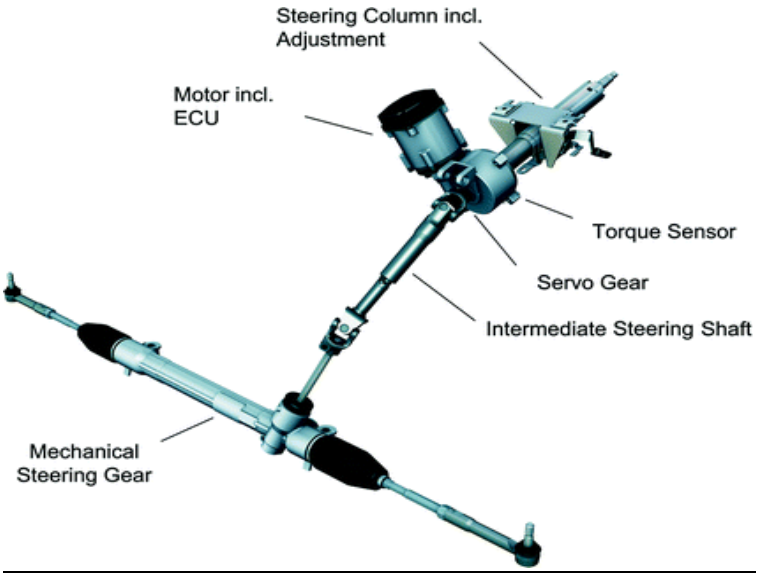


Figure 8: Electronic Power Steering Example (Reprinted from [32])

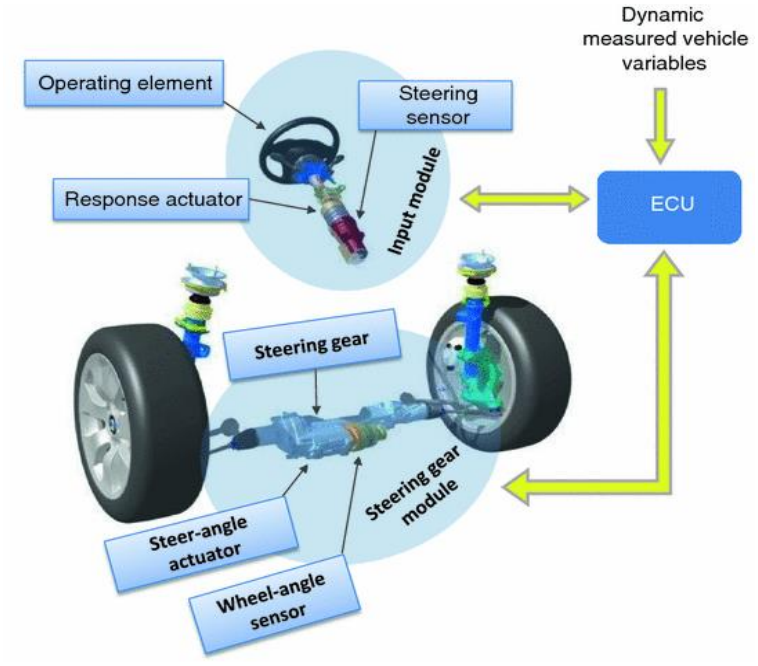


Figure 9: Steer-by-Wire Example (Reprinted from [32])

The steering wheel of the Baja vehicle is approximately 12” in diameter. Turning the steering wheel rotates a shaft that changes angle through a universal joint and then enters a manual rack and pinion. The input to the manual rack and pinion is connected to the pinion so

that rotation moves the rack side to side. Each side of the rack is connected to a tie rod that connects to steering knuckles. The tie rods and steering knuckles transforms the linear motion of the rack into a pivoting motion of the wheels. For example, rotating the steering wheel counter-clockwise (CCW) causes the steering input shaft to rotate CCW, therefore rotating the pinion CCW, causing the rack to translate to the right, which pulls the left tie rod inwards aiming the left wheel to the left and pushing the right tie rod outwards aiming the right wheel to the left.

The steering system was the most difficult subsystem to actuate autonomously and manually. A large torque requirement, precise controllability, rapid response, and the need for human control contributed to this difficulty. The team's measurements showed that the maximum torque at steering input shaft did not exceed 30 lb. ft. This measurement was taken by using a 50 lb. scale and a one-foot long wrench to rotate the steering input shaft. Figure 10 and Figure 11 contain a visualization of this measurement.

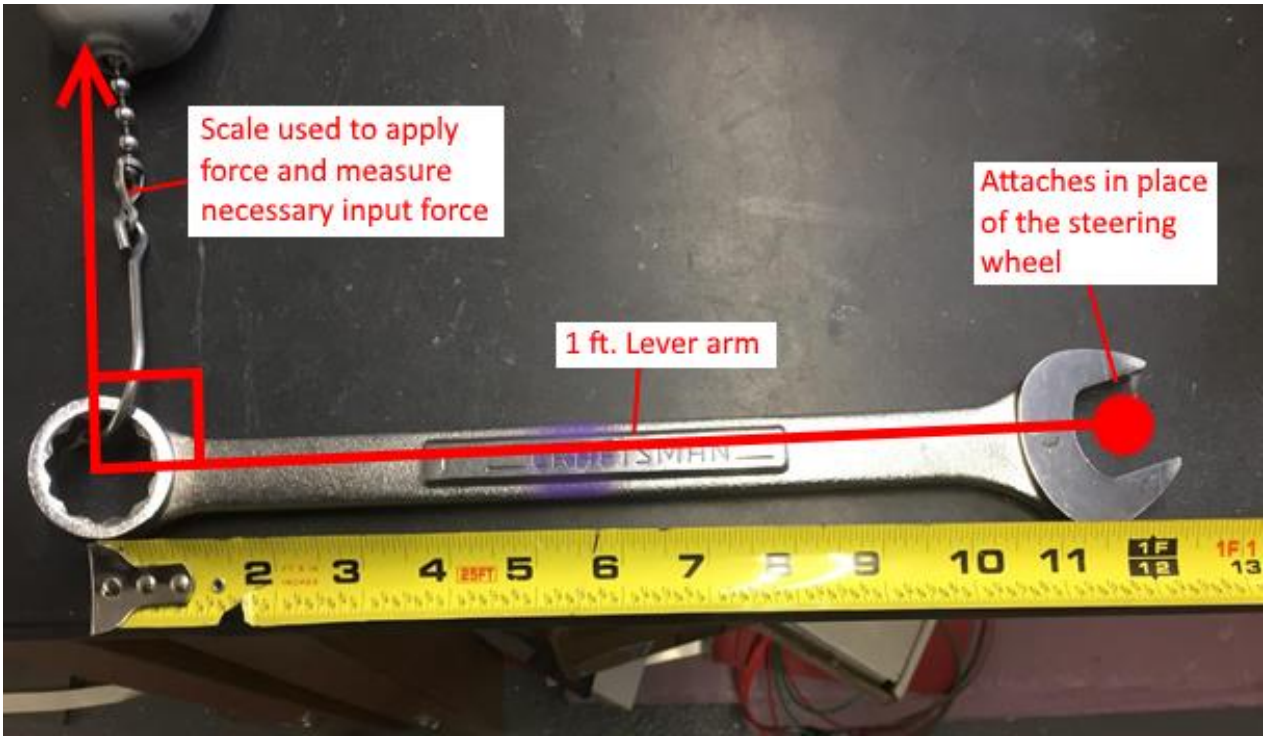


Figure 10: Fish Scale and 12" Long Wrench used to Measure Steering Input Torque

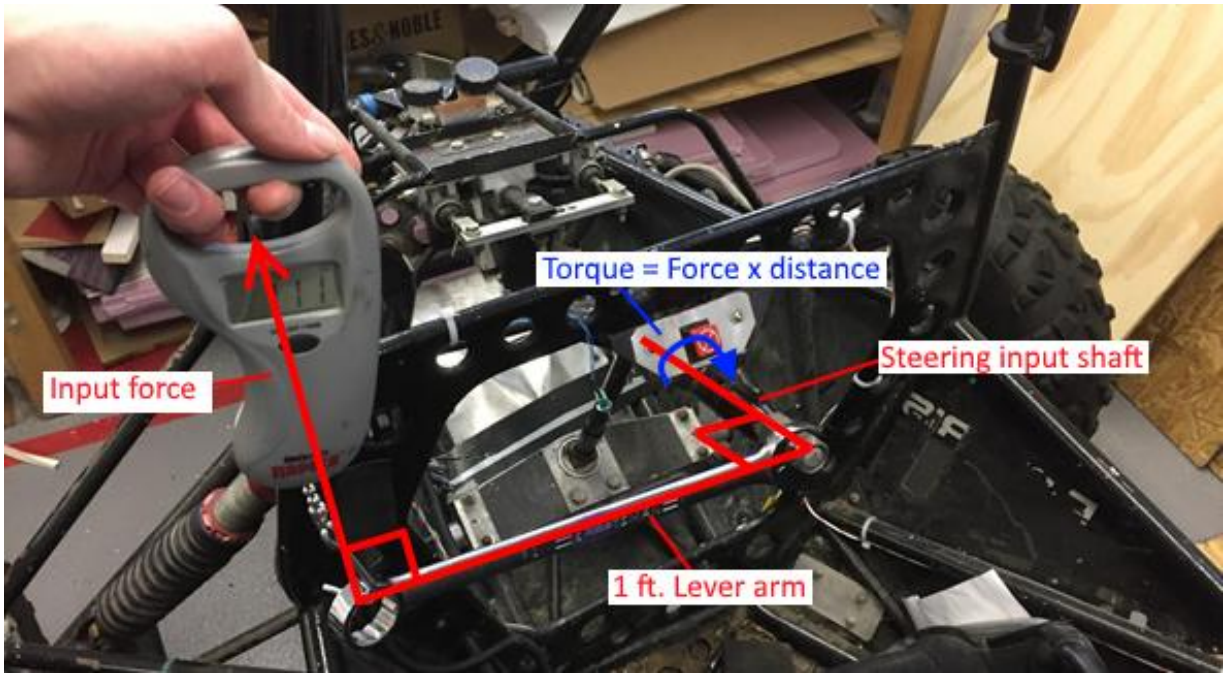


Figure 11: Measuring Steering Input Torque

The high torque demand along with the rapid response required a powerful motor with high mechanical advantage. It would have been difficult if not impossible to back drive in the event of an emergency or for unpowered operation. A potential solution was to use a clutch system to disengage the motor. Alternatively, the steering system could have been powered using a secondary power supply. However, this would not account for something going wrong in the actuation system. This solution would have posed a safety risk to the driver if the electro-mechanical system fails in some way and there is no way to manually steer the vehicle. An entirely electric solution would allow the mechanical linkage between the steering wheel and rack and pinion to be removed. In industry this is attractive because it saves weight and provides room for other things. In this project, an entirely electric solution would make packaging the actuation easier because it would allow for more space and more direct access. However, this implementation is more complex because it requires linking the steering wheel and rack and pinion electronically. Table 2 contains a comparison between keeping the mechanical linkage and decoupling the steering wheel and the rack.

Table 2: Quantitative Decision Matrix for Steering Design

Design Requirements	Weight	Steering wheel and rack are decoupled	Retain steering input shaft
Failsafe	**10**	0	10
Control responsibilities	**7**	3	7
Ease of switching modes	**10**	10	10
Implementation difficulty	**7**	5	7
Feel	**5**	6	6
Cost	**3**	7	7
Software	**2**	9	9
Total	**1**	225	367
Percent total	**1**	51%	83%

Considering the failsafe mechanism for both designs was paramount to maintaining safety in the event of an electronic malfunction. For example, a problem with the electro-mechanical actuation system would leave the operator unable to regain steering control if the rack and steering wheel were decoupled. If the steering input shaft is retained, disengaging the clutch that mates the motor to the steering input shaft would mean that quick steering is the only necessary action. This will give the operator full ability to mechanically turn the steering wheel. Decoupling the steering wheel and rack results in having to keep the vehicle wheels and steering wheel correctly aligned in both autonomous and manual modes. Retaining the steering input shaft requires only aligning the vehicle wheels to the steering wheel using torque sensing when operating in manual mode. In autonomous mode the steering wheel will be aligned through the linkage.

Both options allow for relatively seamless transitions from autonomous driving to human occupant control. Both solutions also require similar amounts of design and fabrication for the actuation portion. Removing the steering input shaft however would add to the overall complexity. This is crucial for ease of switching between autonomous and manual modes, as well as for maintaining some semblance of a connection between the two components during manual mode. Neither solution is going to have a significant improvement in feel, however both

result in less effort on the driver's part to turn the wheel. Both solutions cost about the same in material cost because many of components (motor, potentiometers, brackets, etc.) are shared between the designs. Both solutions are also relatively simple to implement in software, with neither one requiring significantly more code development than the other.

The second solution idea in which the steering input shaft between the steering wheel and the rack and pinion was retained was decided as the better solution according to the design requirements; this option includes a reliable failsafe, less implementation complexity, and less control responsibilities.

4.2 Methods, Testing and Results

As discussed in the beginning of this chapter, actuation devices are needed on the three major human inputs: acceleration, braking, and steering. This section of the report will discuss the team's mechanical development work, as well as the results of testing the mechanical development. To outfit the vehicle with actuators, the project team created an initial computer aided design (CAD) model in a program called Fusion 360, using measurements of the Baja vehicle. When the team modeled the actuators, they were first 3D printed to test fit on the vehicle. An image of the Fusion 360 model to date is available in Appendix B. The model allowed for a stronger visualization of sensor placement, sizing, as well as overall vehicle balance, center of mass and weight as the project continues to develop. It was also used as the primary basis from which the actuated parts were designed, as well as test placements and actuation. Most components were bolted-on to allow for serviceability, future modifications, and reversibility. Prior to assembling the physical components of the Baja vehicle, the team unit tested the functionality of individual components. The team wired and tested the actuators prior to installation.

To address acceleration, the team uninstalled the two sealed deep cycle batteries that were non-functional and wired two marine deep cycle batteries into the circuit for stationary testing. Loose connections were then secured, a blown fuse for the emergency-stop circuit was replaced, and then used to test functionality of the system. The accelerator pedal would make the wheels spin if both emergency stop switches were not pushed in. It can also make the wheels spin either forwards or backwards depending on a switch labeled as the reverse switch. It was discovered that there was a single switch that can turn the motor controller on or off.

To actuate the acceleration system autonomously, two circuits were required. One circuit was used to translate a pulse width modulated signal from an Arduino to a DC signal used to control the acceleration of the vehicle. The second circuit allowed the acceleration to be controlled by either the mechanical pedal, or the electrical system. Section 5.2 contains explanations of the circuits and their functionality.

To address braking, the team obtained a linear actuator capable of exerting 100lb force for compressing the brakes. Placement of this actuator needed to be parallel to the travel of the two brake cylinders. This meant that the reservoirs and associated mounting bracket had to be removed. The reservoirs were relocated by designing a clamp that secured the reservoirs to tubes. The team then mocked the linear actuator up directly above the brake cylinders and found that a method was required to secure the base of the linear actuator to the vehicle frame and to secure the piston of the linear actuator to the brake mechanism.

To secure the linear actuator to the frame of the vehicle, the team designed a small metal frame that was bolted to vehicle and bolted to the linear actuator. This design was completed in CAD software, Autodesk Inventor. The engineering drawings of this design can be found in Appendix C: Brakes: Linear Actuator Mount Drawings. The design was based primarily on space constraints and manufacturability considerations. Because of the large size of the linear actuator mount, it did not make sense to machine the mount out of stock because of the wasted material. Therefore, it was logical to design the mount out of mild steel bar and angle stock. Because the mount would be constructed out of mild steel, the different pieces that make up the assembly could be joined together by welding.

Prior to manufacturing, the design needed to be validated to ensure that it would perform adequately and not fail under the maximum force the linear actuator can generate (100 lb-force). To validate the design, finite element analysis (FEA) was conducted, where 100 lb-force was applied to the linear actuator bolt holes and the attachment points were fixed. The FEA report can be found in Appendix D: Brakes: Linear Actuator Mount Finite Element Analysis (FEA). The FEA report showed that there was minimal deflection and more than adequate safety factor for the design. It should be noted that the joints were treated as bonded rather than welded because the tensile strength of the weld will equal or exceed that of the bar and angle stock. The bar and angle stock are A36 steel, with a tensile strength of 36,300 psi [33]. The tensile strength of the stainless steel 308L filler rod material once welded is above 75,000 psi [34].

Because the metal stock is relatively thin (⅛-inch) and the joints are small, tungsten inert gas (TIG) welding would likely produce the cleanest weld joints. To weld ⅛-inch mild steel, a 2% thoriated tungsten electrode shaped to a fine point needed to be used in conjunction with argon gas. The welding machine needed to be set to direct current electrode negative (DCEN) and the argon gas needs to be set at about 20 psi. The weld settings that the team used were based off a paper version of the Miller TIG calculator, as shown in Figure 12.

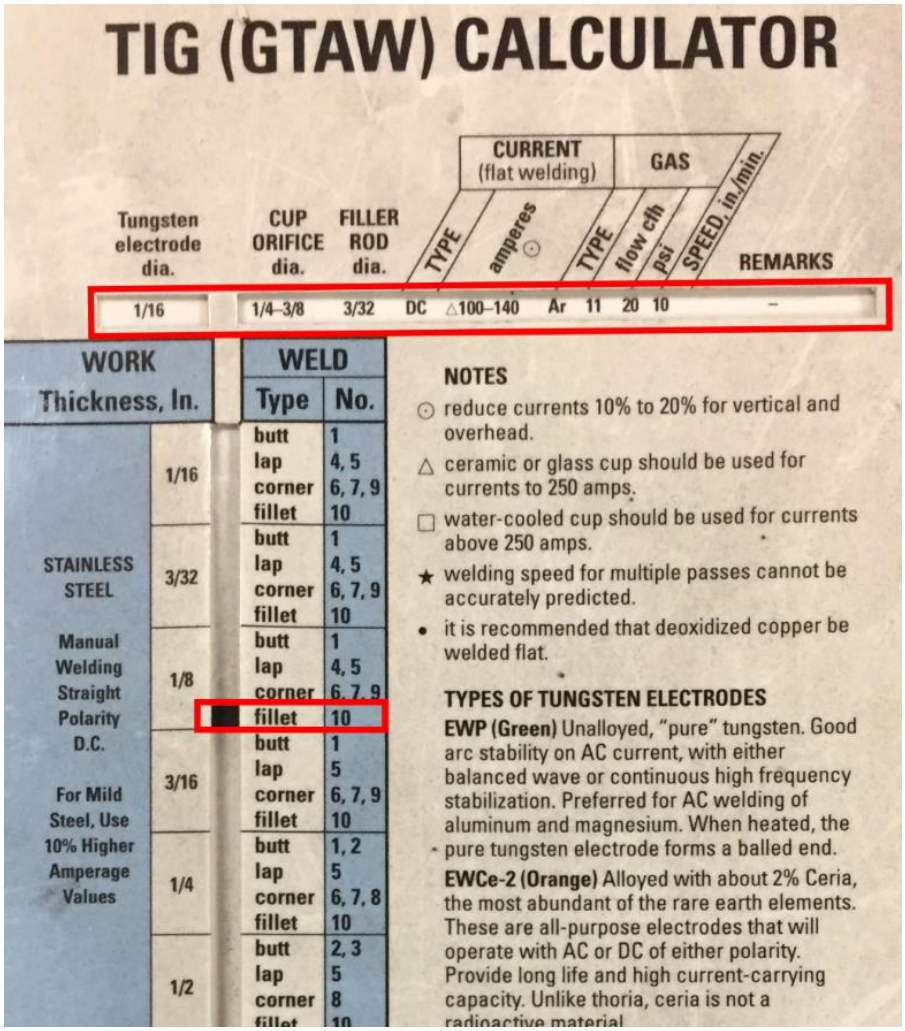


Figure 12. Paper Miller TIG Calculator for Fillet Welds of ⅛-inch Steel

The settings recommended for each weld type for 1/8-inch steel are summarized in Table 3. Generally, the team followed the recommendations with fine tuning for additional weld quality. The welder controls where the settings are adjusted is shown in Figure 13.

Table 3. Weld Setting Recommended for ⅛-inch Steel

Weld Type	Electrode Diameter (in)	Amperage	Gas (psi)	Speed (in/min)
Butt	1/16	93.5-131	20	12
Lap	1/16	110-151	20	10
Corner	1/16	93.5-131	20	12
Fillet	1/16	110-154	20	10

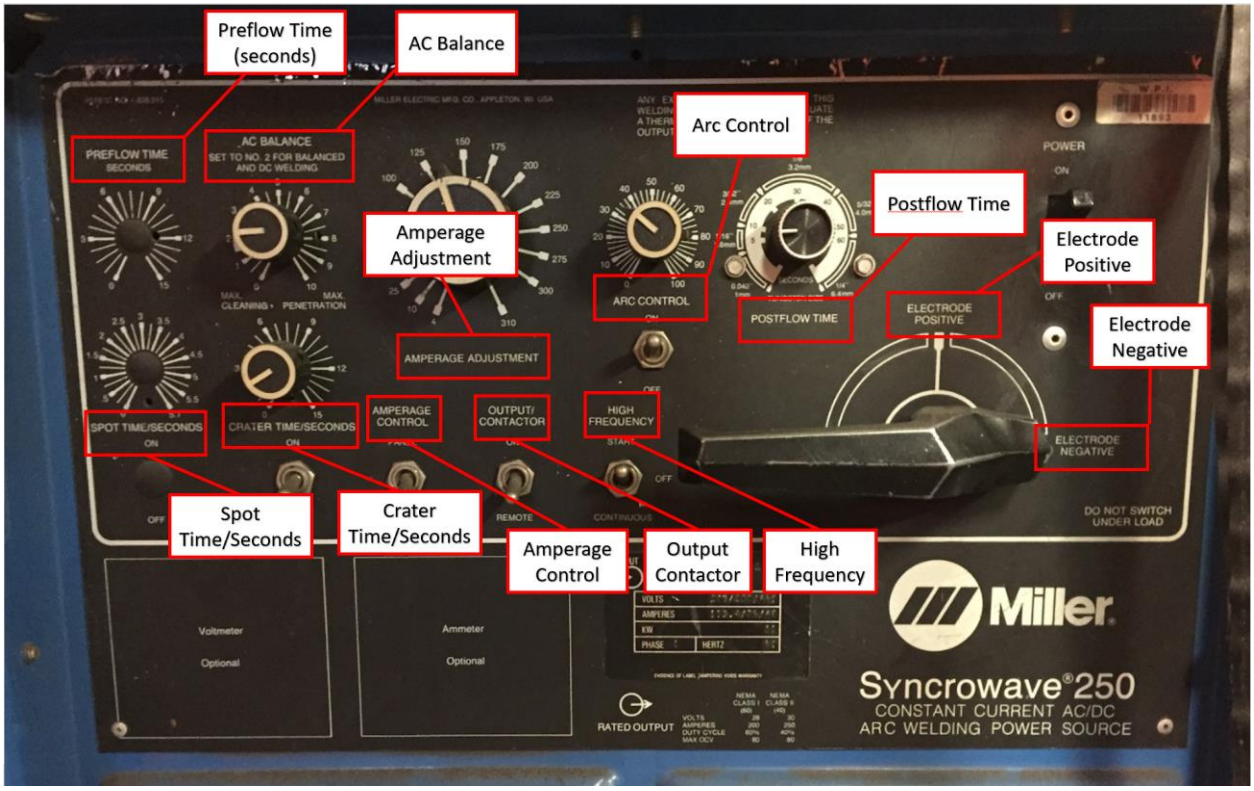


Figure 13. Miller TIG Welder Controls for Adjusting Weld Settings

The finished product fit as designed. Figure 14 through Figure 17 show the installed system from several different views. The team used Grade 8 or better hardware in the assembly.

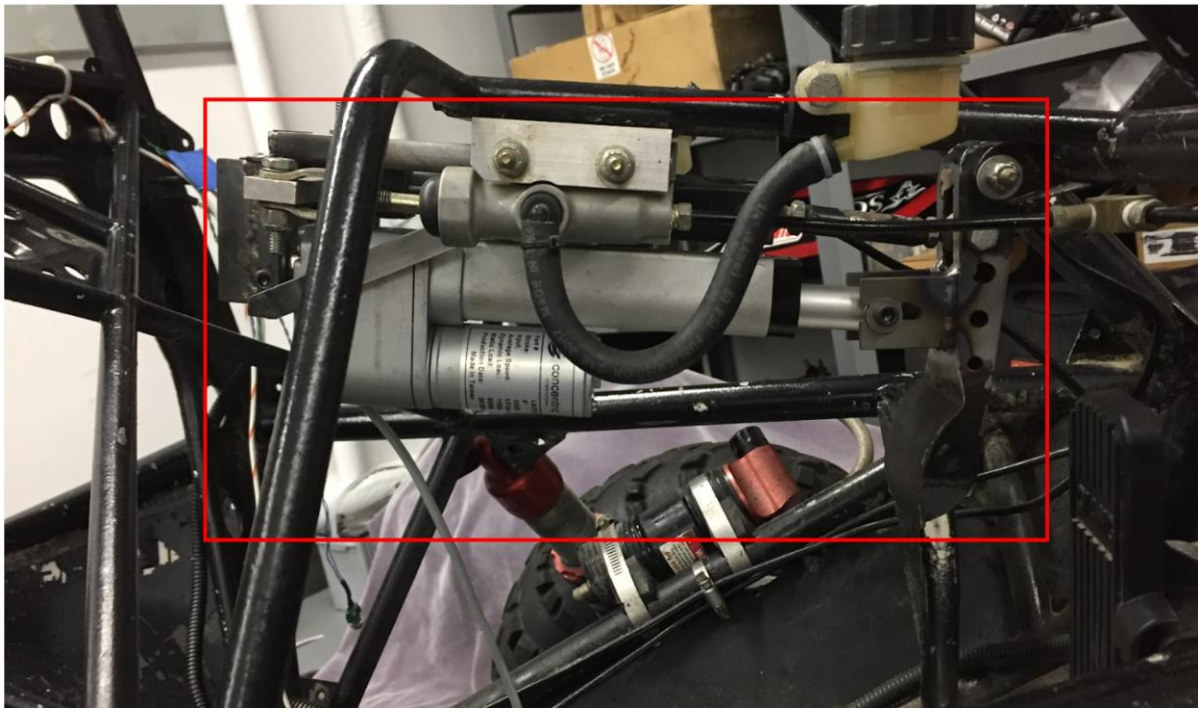


Figure 14. Linear Actuator Mount for Brake Actuation as Shown from the Right Side

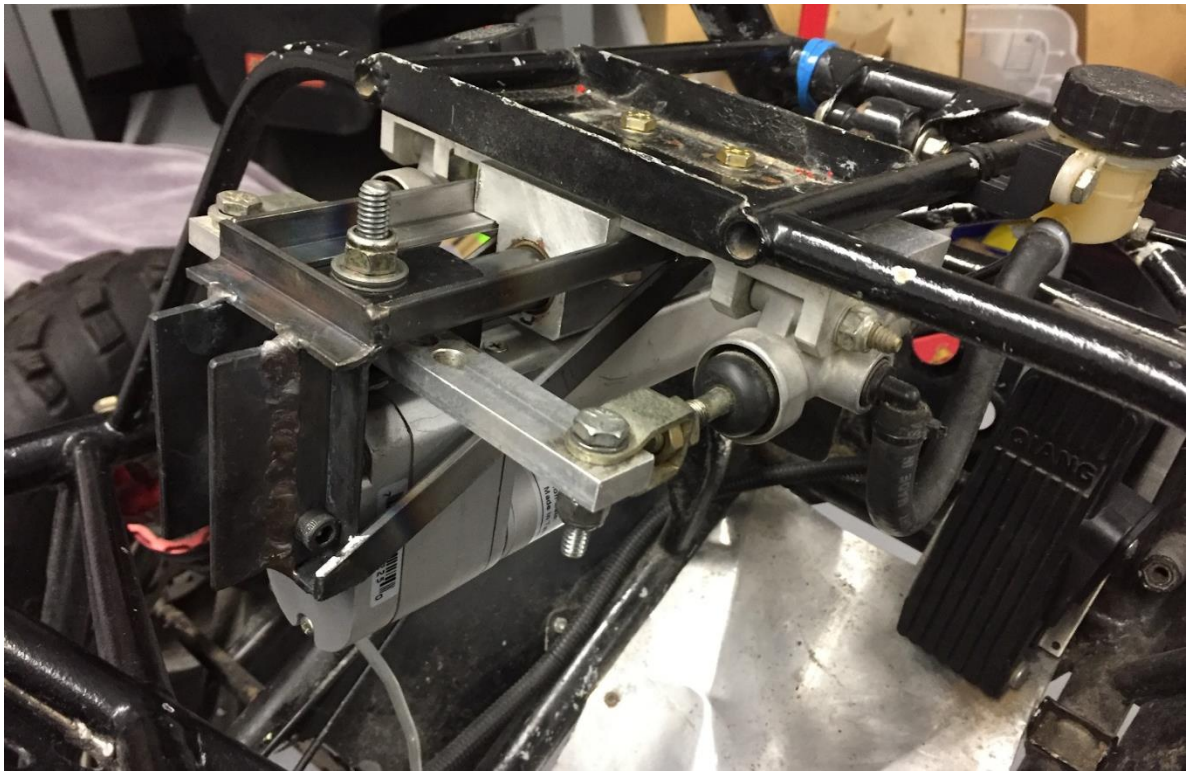


Figure 15. Linear Actuator Mounted to the Vehicle with Welded Steel Frame

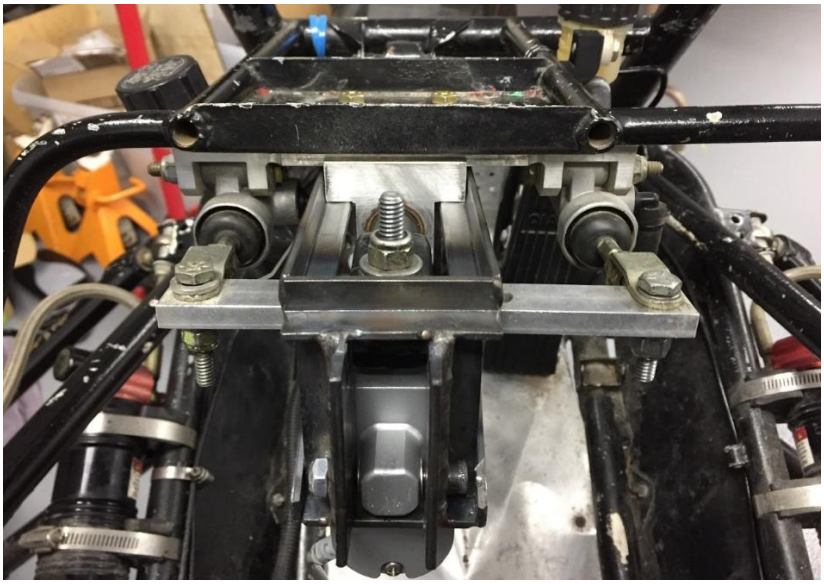


Figure 16. View of the Linear Actuator Mount from the Driver's Seat

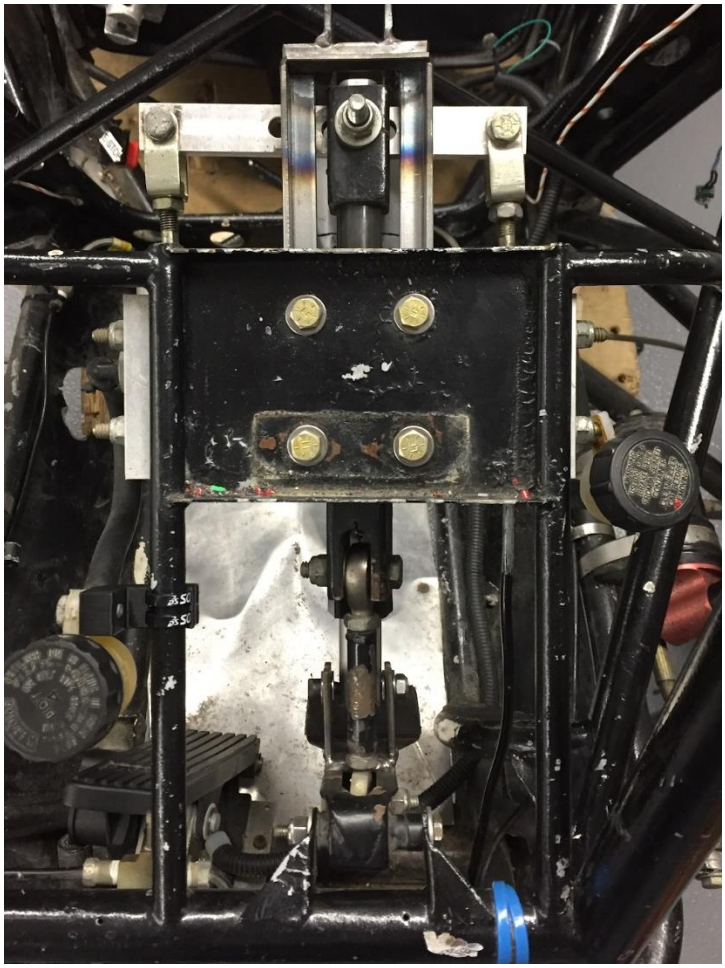


Figure 17. Linear Actuator Mounted to the Vehicle as Shown from Above

One problem with the design was that the retaining bolt would fall out of the travel slot during manual braking. The retaining bolt allows the linear actuators piston to travel in the slot attached to the brake pedal, and so it is not supposed to fall out. To remedy this problem, the team made the travel slot longer by welding additional metal and then grinding the surface smooth. This proved to be an effective solution. See Figure 18 for the unmodified travel slot and see Figure 19 and Figure 20 for the lengthened travel slot.

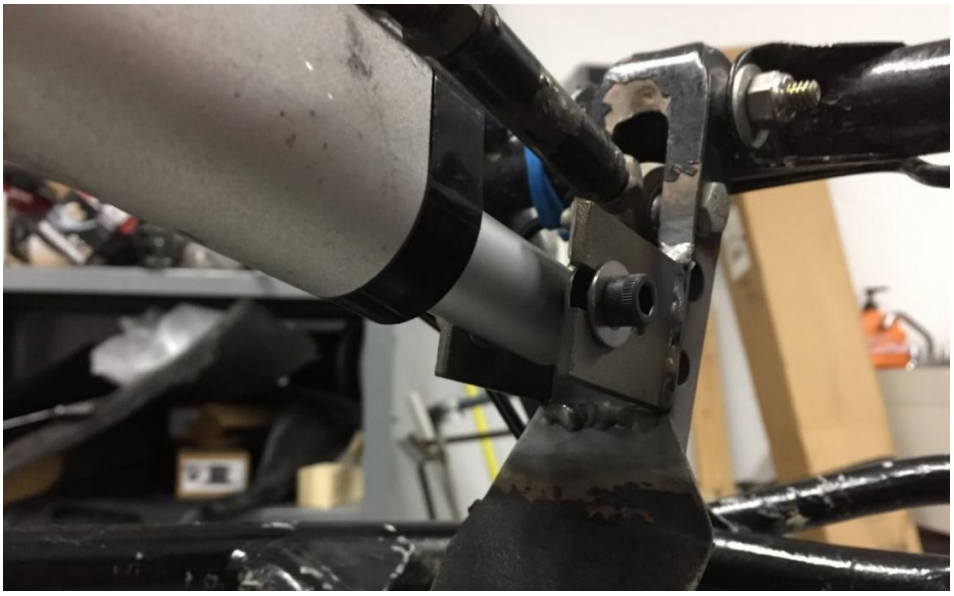


Figure 18. Unmodified, Initial Design for Piston-to-Brake Pedal Interface

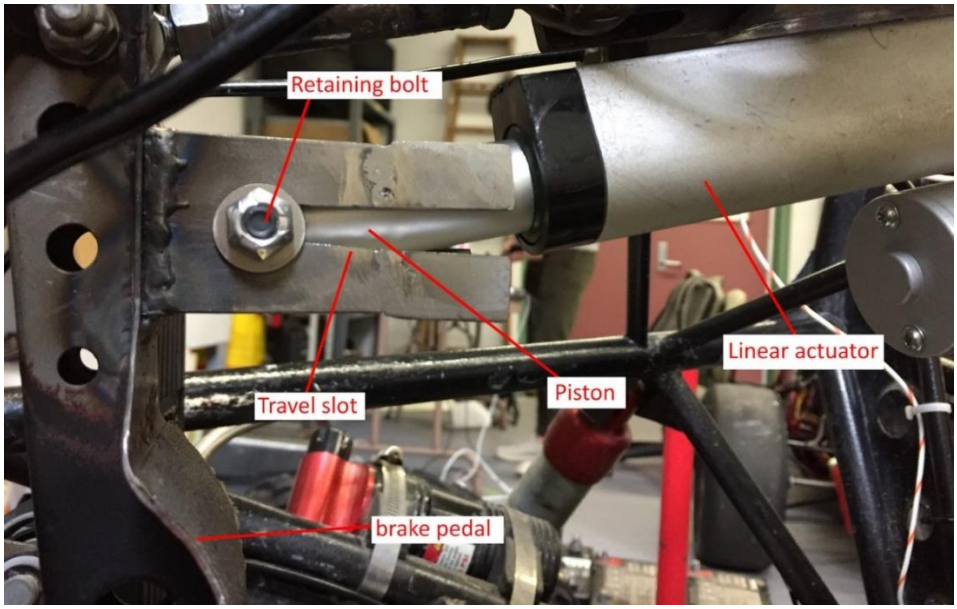


Figure 19. Travel Slot Lengthened to Keep Retaining Nut in Place Under Manual Braking

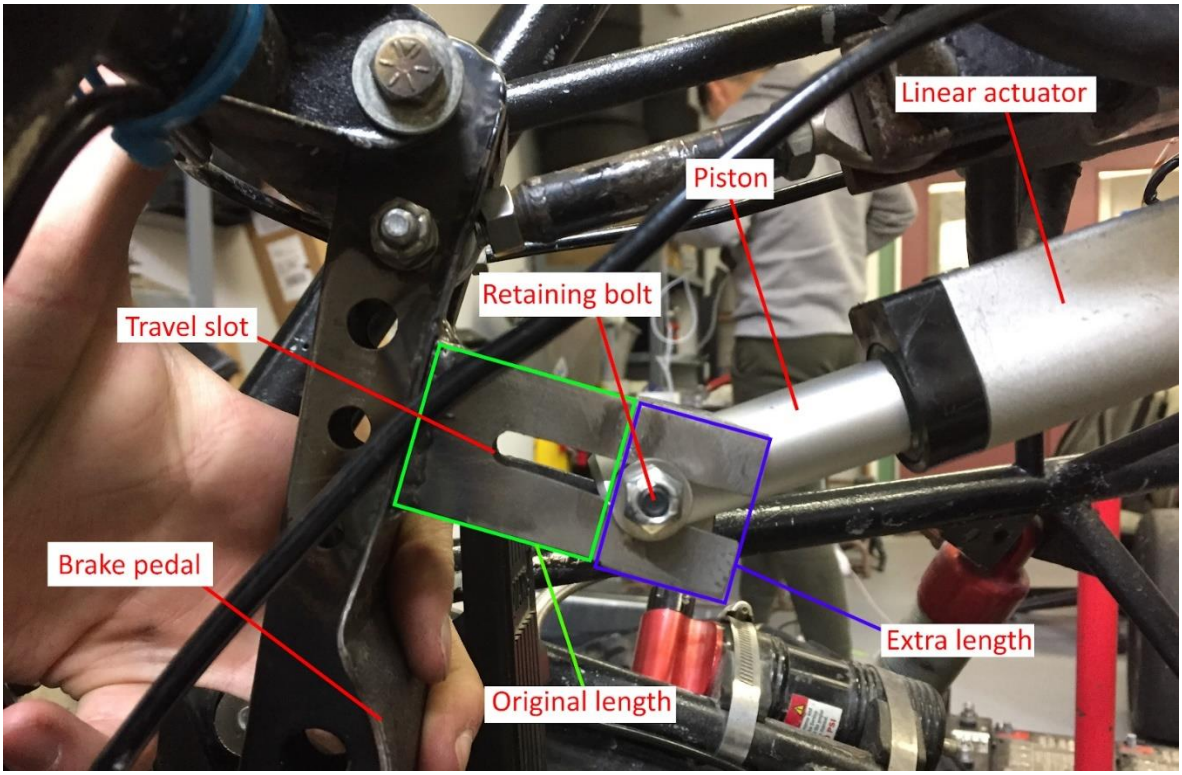


Figure 20. Demonstration that Under Manual Braking the Retaining Bolt does not Fall Out

The linear actuator for the braking system was hooked up to a motor controller, that could handle a range from 5V to 24V, and could output up to 10A continuously. The motor controller was wired to a microcontroller, through a breadboard, to the motor. A feedback loop was written that will continuously monitor how far the motors potentiometer is turned and allows users to set the motor to any location that is needed or desired. A formula was created that expressed the actuators position not as a set of potentiometer values, but rather as a percentage of full extension. If the motor was to extend to 50% of its maximum extension capability, the following line of code would be run:

```
linActControl(50);
```

The motor, when active, only pulls around 0.7A, which is far below the 10A maximum of the motor controller. The motor controller that is controlling the braking system can also control the steering system, as the motor controller is dual-channel, meaning it can control two motors independently.

For the steering system, the team used an ACI82460 window motor to actuate the steering shaft. Due to the unusual shape of the motor, the team needed to design a custom mount

in Fusion 360 to house the motor. In addition to the motor, the team designed and manufactured a 60-tooth custom gear to interface the steering shaft and the window motor. The window motor can be seen in Figure 21.

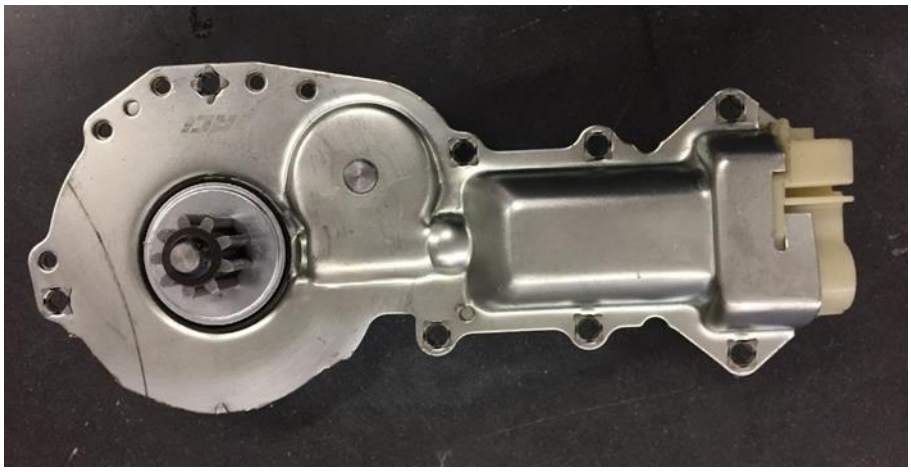


Figure 21. Window Motor for Steering System

To actuate the steering, the team need to transmit torque from an electric motor to the input shaft of the steering rack. The transmission options include: belt drive, chain drive, and gear drive. Since human-operated steering needed to be retained, it made sense to use a gear drive system with a four-bar linkage to engage and disengage the teeth of the two gears. This allows the operator of the vehicle to manually put the vehicle in one mode or the other.

The steering transmission needs to be able to supply the input of the steering rack with 30 lb. ft. of torque. The electric window motor that the team sourced can output in excess of 10 lb. ft of torque. However, it is ideal to run the motor at significantly less load to reduce current draw and increase controllability. Specifically, the team wanted to target a 5 lb. ft. peak torque, which means a reduction ratio of 1:6 is necessary. Since the motor has 9 teeth, it would be ideal if a 54-tooth gear was driven by the motor to satisfy the desired ratio. Unfortunately, this size gear resulted in interference between the actuator and the steering input shaft as discovered in CAD modeling. A 60-tooth gear resulted in enough clearance between the electric motor and the input to the steering rack. The resulting reduction ratio was 1:6.67, which was acceptably close to the desired reduction ratio.

The actuator was placed between the base of the steering shaft and the conjunction of the vehicle that holds the steering wheel in place. A custom cradle needed to be manufactured so that the motor could be held in place in the vehicle. To keep the motor in place as it interfaces with

the shaft gear, a 4-bar is used. The 4-bar was intentionally designed to reach its toggle point and lock in place so that the gear would not slip when it interfaced with the vehicle. A lever was then put in place in conjunction with a spring so that the 4-bar would be able to toggle out of its locked position and disengage the motor from the drive shaft. This lever is controlled via a clutch placed near the car.

Part of this design included a gear that would interface with the gear already attached to the electric motor. The team took key measurements of the existing gear, shown in Table 4. Figure 22 is a visual representation of some gear nomenclature used in the tables and calculations that follow.

Table 4. Measurements for Actuator Gear

Measurement	**Abbreviation**	**Equation**	**Value**	**Description**
Teeth	N	none	9 teeth	The number of teeth on the gear
Pitch Diameter	D	D = N/P	0.75 inches	The diameter of the pitch circle
Pressure Angle	PA	none	20°	The angle force is always applied when gear teeth mesh

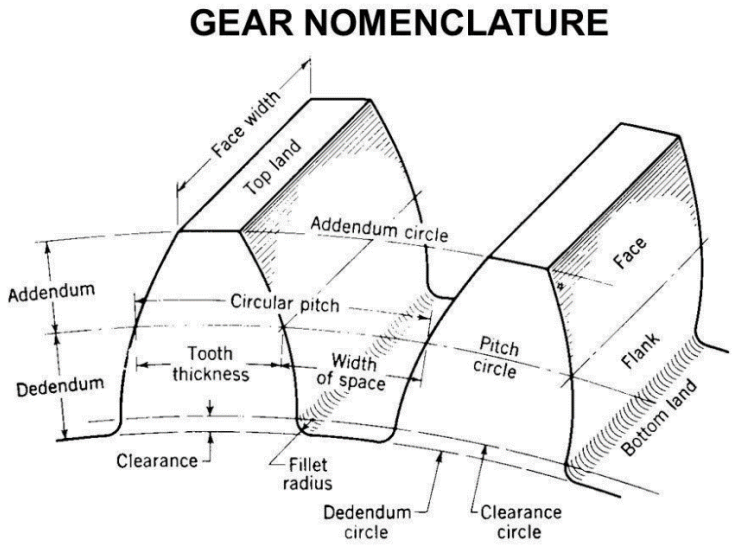


Figure 22. Gear Nomenclature (Reprinted from [35])

The measurements from the above table allowed the team to derive the useful properties as shown in Table 5. These properties are used in the parametric equations that represent the involute curve of the gear teeth. Two gears that are meant to interface must share some of these properties, including the diametral pitch and pressure angle. Using a true involute curve for representing the curvature of the face of the gear ensures that the torque from one gear is transmitted smoothly and continuously to the other.

Table 5. Derivations from Measurements for Actuator Gear

Measurement	**Abbreviation**	**Equation**	**Value**	**Description**
Diametral Pitch	P	P = N/D	12 teeth/inch	The number of teeth per inch of diameter of the pitch circle
Addendum	A	A = 1/P	1/12 inches	The radial distance between the pitch circle and the addendum circle
Dedendum	B	B = 1.25/P	B = 0.1042 inches	The radial distance between the pitch circle and the dedendum circle
Addendum circle	da	da = (N + 2)/P	0.9167 inches	The diameter of the addendum circle
Dedendum circle	df	df = -2*(1.25/P) + D	0.5417 inches	The diameter of the dedendum circle
Base circle	db	Db = D*cos(PA*pi/180)	0.7048 inches	The diameter of the base circle

By changing the number of teeth, the team could calculate new values for the gear properties. The gear properties are necessary for designing the gear. In Table 6, there is a

summary of the different properties calculated for three different gears: the actuator gear, the 54-tooth driven gear, and the 60-tooth driven gear.

Table 6. Summary of Gear Properties

Gear	**N**	**PA**	**P**	**D**	**da**	**db**	**df**
Actuator	9	20	12	0.75	0.917	0.705	0.542
Driven I	54	20	12	4.5	4.667	4.229	4.292
Driven II	60	20	12	5	5.167	4.699	4.792

Two gears that are coming together must be designed to interface at a specific distance. To find the distance from the center of one gear to the center of the other gear, the team followed the calculation below. The results of this calculation are shown in Table 7 for two different gearing options.

$$Center\ distance = (D1 + D2)/2$$

Table 7. Center Distance Calculations

Driving Gear	**Driven Gear**	**Center Distance**
Actuator	Driven I	2.625
Actuator	Driven II	2.875

The fillet radius of the gear is dependent primarily on the shape and path of the interfacing gears teeth as well as our means of manufacturing. The smallest end mill that the team had access to is a 1/8-inch diameter end mill. This means that the smallest achievable fillet radius the team could select is 1/16-inch. This was a suitable fillet radius because two 1/16-inch fillets could fit between two gears without overlapping each other. The involute curve that defines the profile of the gear teeth is based upon the parametric equations, where “D” is pitch diameter and “PA” is pressure angle:

$$xt = 0.5*D*\cos(PA*pi/180)*(\cos(t) + t*\sin(t))$$
$$yt = 0.5*D*\cos(PA*pi/180)*(\sin(t) - t*\cos(t))$$

The team designed the 54-tooth driven gear and 60 tooth driven gear in SolidWorks and then imported them to Fusion360 to make a CAM profile. Prior to manufacturing, the team both laser cut and 3D printed the 60-tooth gear to test fitment with the 9-tooth actuator gear. Since the test fitment was adequate, the team machined the 60-tooth gear out of mild steel.

The gear that is driven by the actuator is bolted in-line to the input shaft of the steering rack. Having the gear attach with bolts allows the team to change the gear out for different one if the gear becomes damaged or worn or a different reduction ratio is desired. The safety factor can be calculated using different grade bolts and optimize weight by choosing smaller fastening hardware. Since the gear is sandwiched between two ¼-inch steel plates, the bolts used to attach the gear are placed in double shear. The team will regularly be subjecting the gear to 30 lb. ft. of torque. Given that torque = force * distance, and the four bolts are placed at a distance of 0.75-inches from the center of the input to the steering shaft, the force that all four bolts will be collectively subjected to is 480 pound-force. Under ideal circumstances that will divide to 120 pounds of shear force per bolt.

To calculate the safety factor of the fasteners chosen, the team needed to know the tensile strength of the bolts. The tensile strength for several different bolt options are listed in Table 8, in addition to the estimated shear strength of those bolts. Generally, the sheer strength is 60% of the tensile strength [36]. The equation for calculating the shear stress (psi) based off force (lbs) and cross section area of the bolt (in^2):

$$T = V/(2A)$$

The equation for calculating the minor cross section area of the bolt is:

$$A = (pi/4)*(dminor)^2$$

The minor cross section area of the bolt is derived from the minor diameter of the bolt (dminor). This diameter is calculated from the normal diameter of the bolt (dnorm) and the number of threads per inch (TPI). Using the formula below, the minor diameter for the four 8-32

bolts selected for attaching the driven steering gear was 0.123 inches. For perspective, the normal diameter of the bolts is 0.164 inches:

$$dminor = dnorm - 1.299038/TPI$$

To calculate the safety factor of the bolts selected, one can calculate the shear stress for a given application of torque and then divide the rated shear stress of the bolt by the calculated shear stress of the applied torque. The safety factor an indicator of how much margin is given in the design. Table 8 shows the values that the team calculated for this project.

Table 8. Tensile Strength and Shear Strength of Bolts

Bolt	**Tensile Strength (psi)**	**Shear Strength (psi)**	**Safety Factor for 30 lb. ft. of torque**	**Safety Factor for 150 lb. ft. of torque**
318 or 8-18 (Grade 2)	70,000	42,000	8.4	1.7
Grade 8	150,000	90,000	18	3.6
Mil. Spec. Alloy Steel	180,000	108,000	21.6	4.3

To be failsafe, the steering system required stability, strength, and flexibility. The overall design uses a four-bar toggle to manually engage or disengage the steering system, inspired by a vice grip. As discussed previously, when the toggle is engaged, the four-bar system is locked into its toggle point, and the four-bar will not allow further input motion from the driving link. This position sees the driving link in the four-bar become parallel with the coupler, effectively changing the direction of the force vector on the coupler and driving link, locking it in place.

Before moving forward, the team needed to conduct some four-bar calculations. The team assumes that there exists a static point in time where the steering mechanism is in static equilibrium, while engaged with the steering shaft gear. For the gear motor to remain engaged, the total torque of the system must remain zero. The sum of total moments in the system is as follows:

$$3.521A_1Sin(\theta_{A1}) + 3.521A_1Cos(\theta_{A1}) + 3.521A_2Sin(\theta_{A2}) + 3.521A_2Cos(\theta_{A2}) + 4.543C_1Sin(\theta_{C1}) + 4.543C_1Cos(\theta_{C1}) + 4.543C_2Sin(\theta_{C2}) + 4.543C_2Cos(\theta_{C2}) = 0$$

In the above equation, A_1 and A_2 are the forces applied upon the motor by the steering gear and C_1represents the supporting forces from the 3rd and 4th bar in the four-bar mechanism. Once the four-bar is engaged, the forces on the mechanism become counteracting forces and the tensile or compression forces of the 4th and 3rd bar become equal and opposite forces opposing the moments of the steering gear applied to the motor, rotating around point B. This can be achieved by locking the four-bar into the toggle position. A toggle position is defined as a state where the four-bar will not allow further input motion in one direction from one of the rockers [37]. See Figure 23 for the team's hand-drawn analysis.

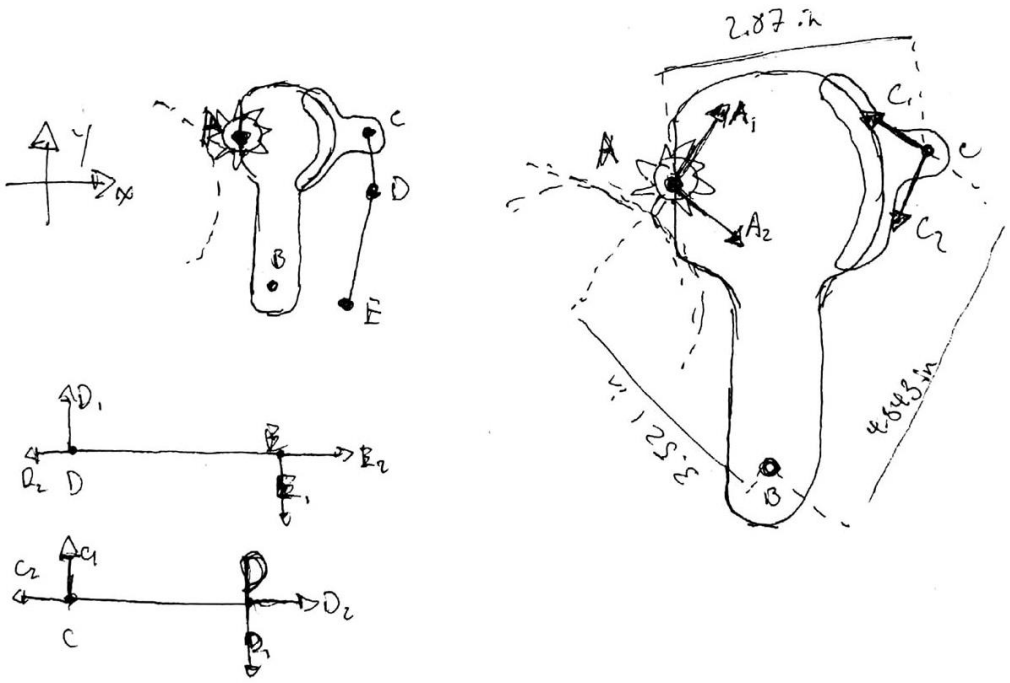


Figure 23. Hand-drawn Four-bar Analysis

The four-bar mechanism is a double rocker mechanism according to Grashof's law [38], where the length of the shortest link "s" and the length of the greatest link "l" is greater than the length of the other two links combined (p + q). In this project's four-bar, s = 2 in, l = 4.543 in, p = 3 in, q = 1.658 in.

$$s + l > p + q$$

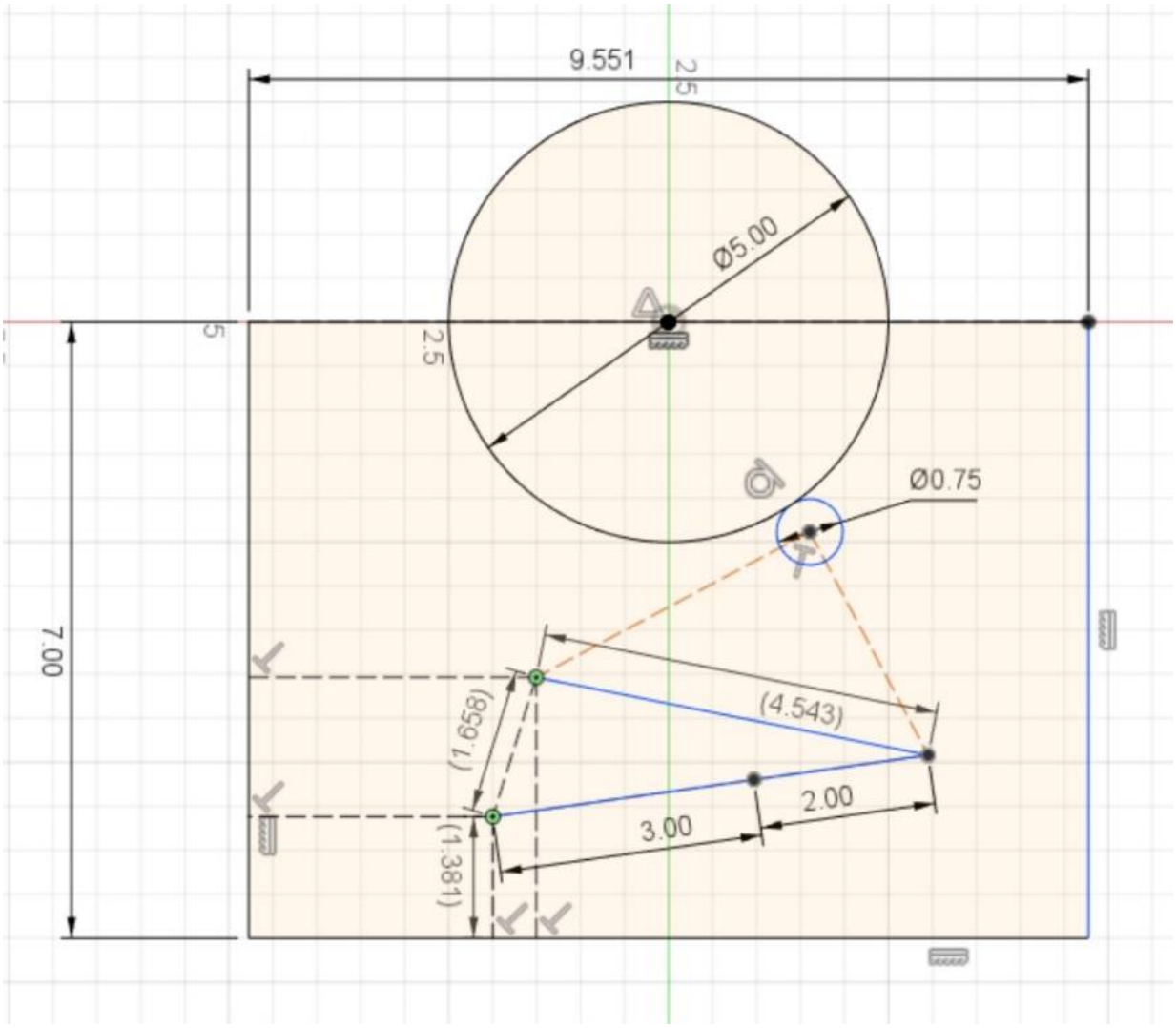


Figure 24. Four-bar Simulation Sketch

Through simulation analysis, the team was able to determine the toggle positions of the designed four-bar and selected the toggle point seen in Figure 24 to satisfy the static equilibrium state, allowing the motor to be locked and engaged with the larger steering gear.

The custom mounting plate that the team designed was intended to combine the motor interface and the four-bar toggle system and secure them to the vehicle. The team designed the mounting plate while considering its space efficiency, stability, and machinability. For the team to properly machine the mounting plate, the entire piece had to be broken down into a sub-assembly of 6 separate pieces that attach to form the mounting plate. The mounting plate is shown in Figure 25.

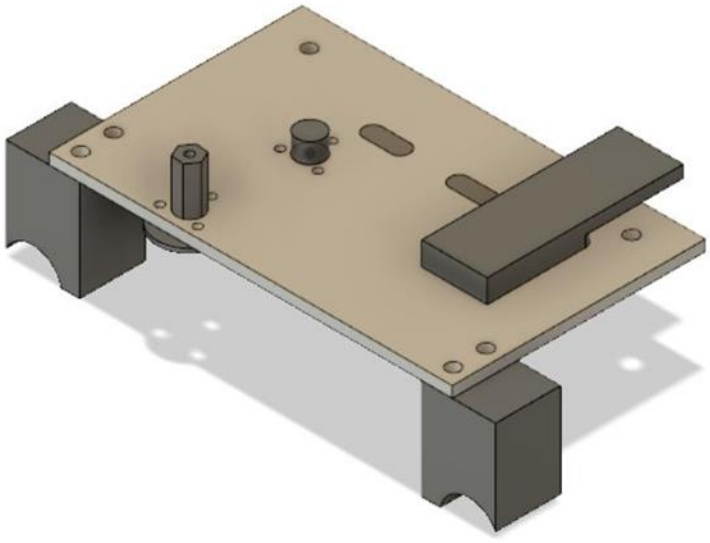


Figure 25. Four-bar and Window Motor Mounting Plate

The machines that the team used to cut the parts are: a Haas Automation VM2 and a Haas MiniMill. The team purchased aluminum stock of appropriate sizes to machine the motor interface and the mounting plate, as well as steel for the 60-tooth gear and its interface pieces. The team modeled and compiled all CAD designs and CAM G-code using Fusion 360. The completed design is shown in Figure 26. The machined parts are shown in Figure 27. The machined parts put together on the vehicle are shown in Figure 28.

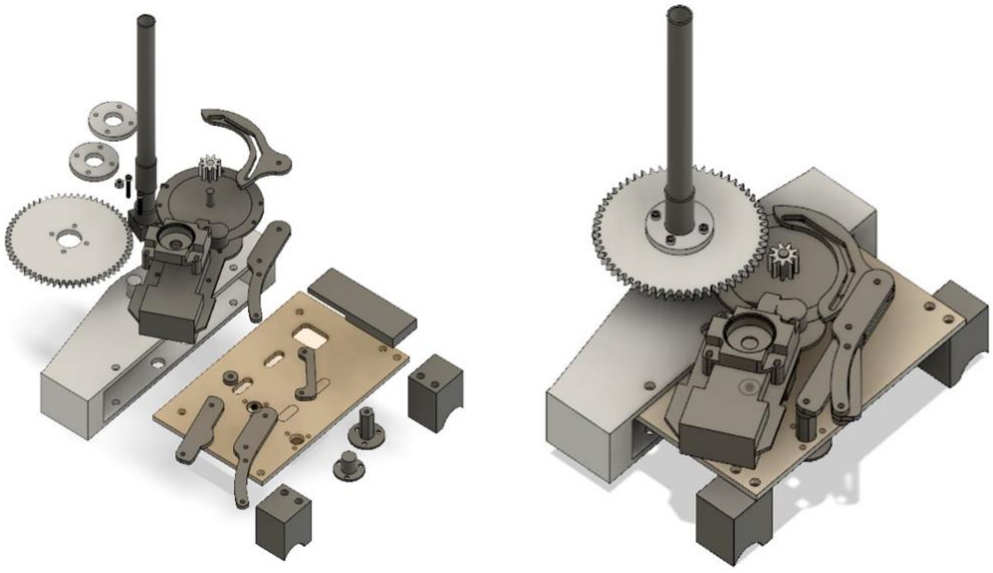


Figure 26. Exploded Steering System (Left) and Completed Steering System (Right)

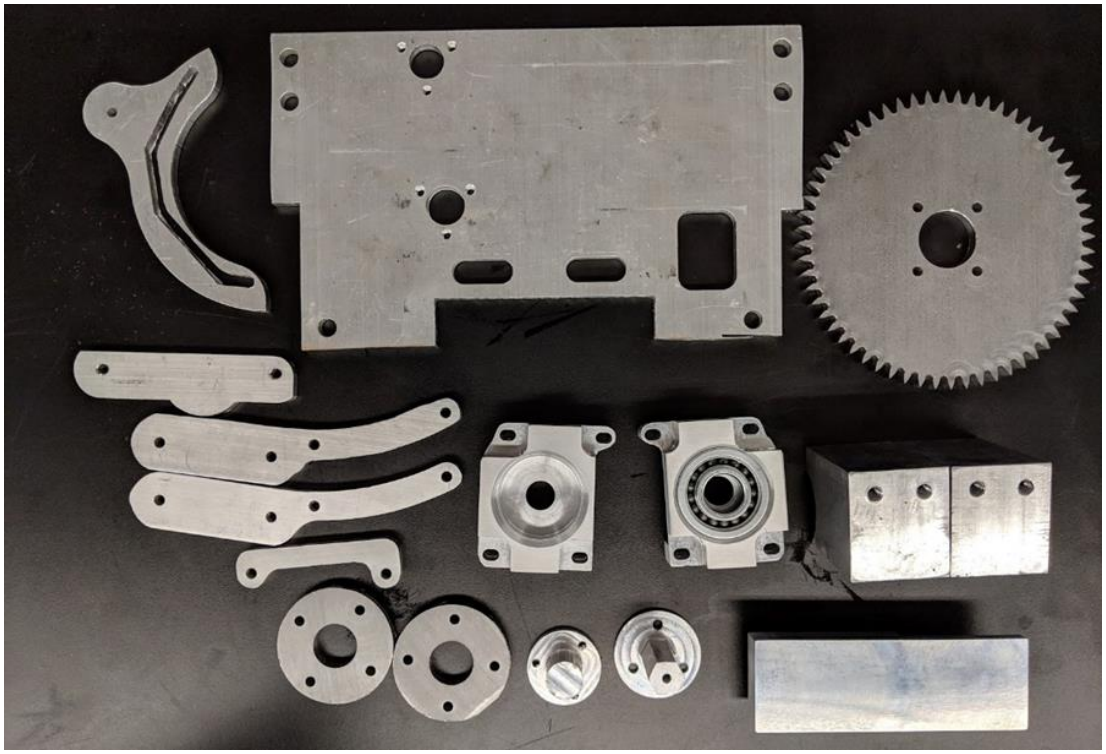


Figure 27. Steering System Machined Parts

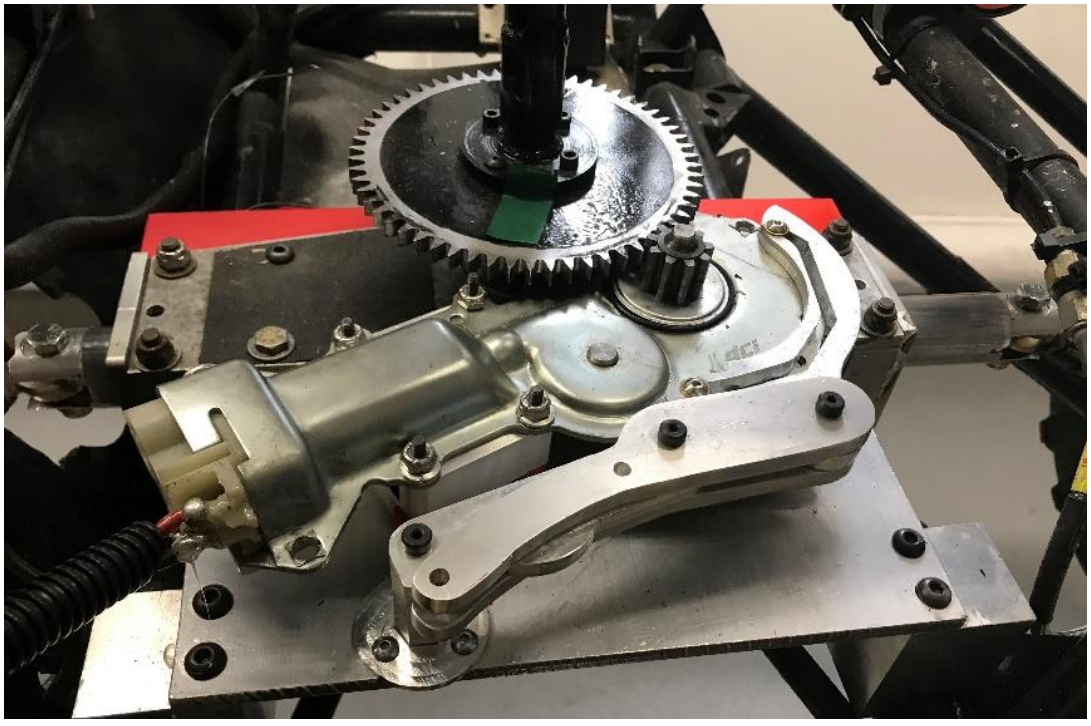


Figure 28. Steering System Machined Parts Assembled

The steering rack is composed of a gear and a sliding rack. As the input shaft of the steering rack turns, the gear turns, converting rotational motion into linear motion. To measure the steering angle in a reliable way, the team decided to put linear potentiometers on the steering rack to measure the position of the sliding component of the steering rack. The team selected a 100 mm slide potentiometer, (part number: PTF01-152A-103B2), that had two readouts for the single finger. By attaching two of the sensors to the steering rack, that will give four individual potentiometer readings to average and compare. Having four readings allows for the ability to detect if one of the readings is faulty or if one sensor unit fails entirely. The potentiometers are mounted in the front of the steering rack. They are housed in a 3D printed enclosure as shown in Figure 29 through Figure 31. The enclosures were printed in PLA with a 20% infill on a Lulzbot TAZ 6 3D printer. The potentiometers are secured to the enclosure with 3-millimeter screws. The finger of the potentiometer is unobstructed in its full range of travel. The potentiometer mount attaches to the steering rack with a single 1/4-20 bolt and geometry that prevents movement.

Figure 29. Top of Potentiometer Mount as Installed in Steering Rack

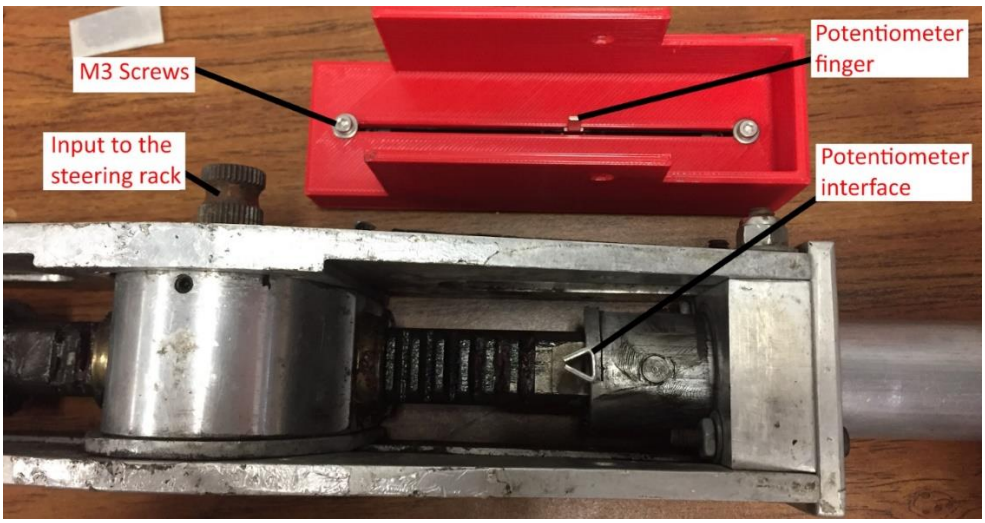


Figure 30. Bottom of Potentiometer Mount Next to Steering Rack

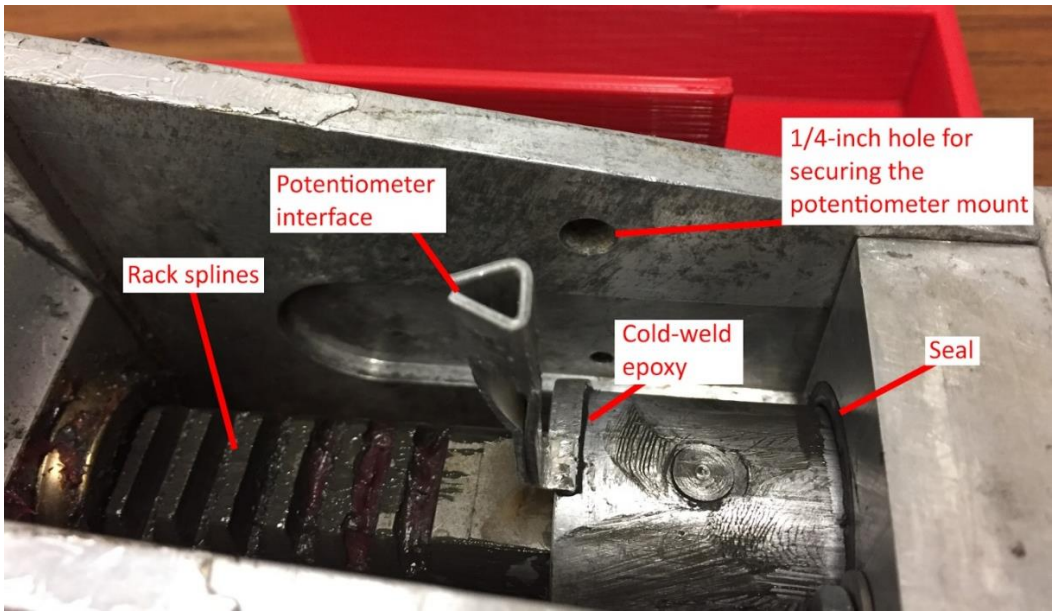


Figure 31. Potentiometer Interface

The finger of the potentiometer sits into an interface that is epoxied to the sliding component of the steering rack. The interface is made from 18 gage aluminum sheet metal. The interface piece went through several iterations to improve rigidity, function, and interference. The final version shown in the following figures does not interfere with the outer seals or the limiters. The sheet metal parts are manufactured by hand.

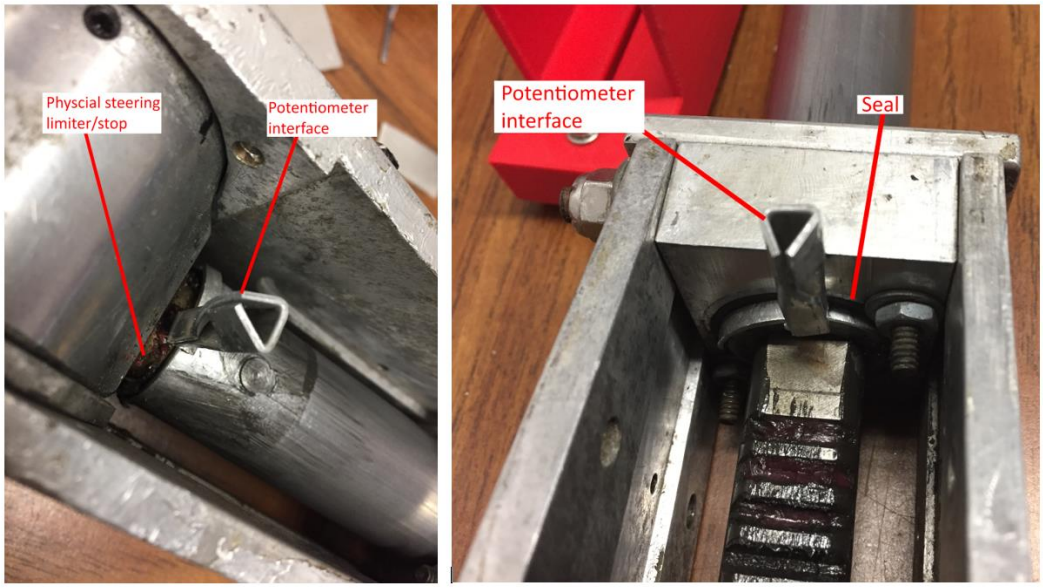


Figure 32. Potentiometer Interface Does Not Interfere with Steering Stop (Left) or Seal (Right)

Although this linear potentiometer worked well for autonomous driving, when switching to manual operation, the “jerky” nature of human operation knocks the finger of the potentiometer off the steering rack. This would cause setbacks for the team, and it was clear that the system had to be re-designed. To remedy this problem, the team designed a steering system that mounts to the shaft of the steering wheel. Part of the shaft rotates, and the other stays in place. Placing a gear on the rotating shaft, and a potentiometer on the static piece allowed for feedback about the position of the steering wheel and would be more difficult to knock loose even with very harsh turns of the wheel.

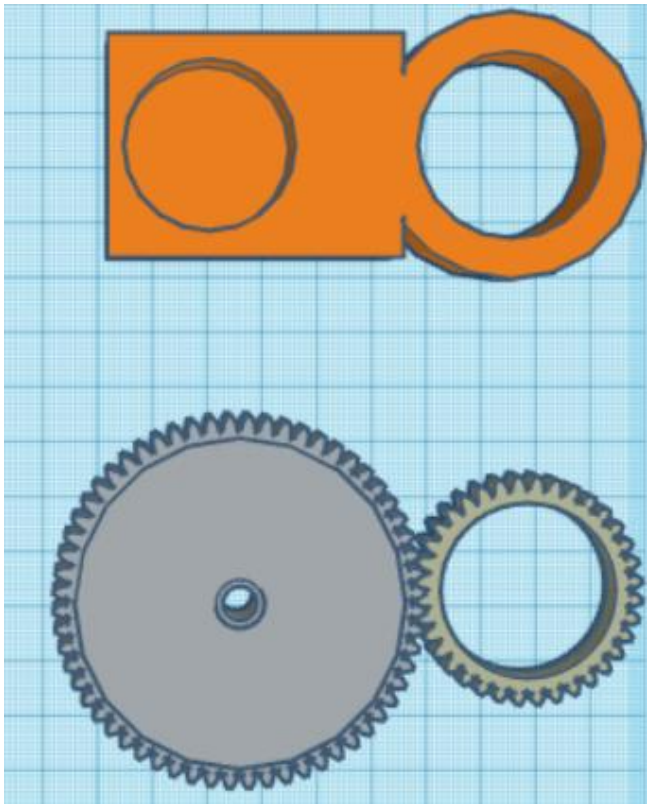


Figure 33. Potentiometer Cradle and Gears 3D Design

Figure 33 shows how the system comes together. The top component cradles the potentiometer, and the larger gear is attached to the potentiometer itself. The smaller, hollow gear is placed on the rotating shaft, while the cradle is placed on the part of the shaft which remains static. The printed version of this, mounted to the vehicle with the potentiometer is shown in Figure 34.

Figure 34. Potentiometer Cradle and Gears Printed and Mounted

4.3 Summary

A vehicle has three core driver controls: acceleration, braking, and steering. Since the Baja vehicle initially only had human driver inputs, electro-mechanical actuators needed to be added to the vehicle to enable a computer to operate the vehicle controls with electrical signals. Acceleration is controlled by interrupting and replacing the signal going to the electronic accelerator pedal. Braking is controlled by a linear actuator that compresses the hydraulic brake cylinders. Steering is actuated by a high-torque electric motor that can be disconnected with a simple clutch. These systems allow autonomous control of the physical vehicle controls with adequate failsafe measures to retain human operability in the case of a system malfunction. The steps to moving forward included developing the designs for actuation, physical implementation, unit testing, and system integration.

In development of custom actuation, braking, and steering control systems, certain design constraints had to be observed. Size, fail safety, and strength were the team's primary considerations for any design that required a man-made mechanical system. The acceleration system was primarily electronic and did not require a mechanical system to be custom built for it. The braking and steering systems were not primarily electronic and did require a mechanical system to be custom built in order to actuate them. When manufacturing components for these systems, manufacturing constraints required various changes in design.

The machining time frame that the team had initially planned for was significantly underestimated. Washburn Shops, the WPI building where the manufacturing machines were held, was only open after 5 PM if there was a lab monitor present making access to the machine shops difficult for team members. It was not guaranteed that a lab monitor would be present in order to keep the lab open after 5 PM. Team members personal schedules had to be coordinated with the lab monitors personal schedules to ensure access to Washburn Shops. For future manufacturing needs, team members are recommended to use the manufacturing lab in the basement of Higgins Lab, because that shop is supposedly easier to access.

Since the vehicle model that the team designed is not 100% accurate to the real vehicle, there were some physical changes and debugging that was necessary after parts were manufactured. Some of these inconsistencies were difficult to measure and were simplified for the sake of producing the vehicle chassis model. Due to the inconsistencies in the manufactured pieces, which were not intended to be structural load bearing devices, they were cut and removed from the vehicle to provide more space for necessary vehicle control systems.

The team also needed to account for parts breaking, specifically the potentiometer fingers. Welding the fingers to the potentiometer system would have required the entire steering system be taken apart again, which was not ideal. The team needed to redesign and print a new method for detecting the current steering position of the vehicle. This was done using a rotary potentiometer.

5: Electrical System Design

Once the team actuated the Baja vehicle, sensors were required to obtain accurate short range and long-range information along with backup data. The team examined several types of sensors, including a stereo vision camera as the method for short-range data acquisition, LiDAR as a source for long-range information, several short-range ultrasonic sensors as a form of acquiring information close to the vehicle, a global positioning system for localization, and a motion sensor for location and velocity detection. Chapter 5 discusses how the team decided on the specific sensors used in this project, as well how they were used in this project and the results that were seen from using them.

5.1 Research and Proposed Methodology

Cameras

Cameras are optical sensors used to capture images. For the vehicle to adequately analyze its environment, image capturing devices were necessary. After the image is captured, image processing must be implemented, enabling object detection, recognition, and perception. The most useful application of cameras is stereo vision, which can be created using two 2D cameras and merging the images together. Stereo vision can detect how far away an object is, as well as the size and depth of the object. “Stereo vision is the process of extracting 3D information from multiple 2D views of a scene. Stereo vision is used in applications such as advanced driver assistance systems (ADAS) and robot navigation” [39]. This is critical for obtaining an accurate model of the surrounding area while navigating around and over obstacles.

Without knowing what the objects are, the Baja vehicle would not know how to traverse the obstacles. A stereo vision system was an effective solution to this problem. Two standard implementations that the team explored consisted of:

- One proprietary stereo vision implementation from two separate cameras
- One established stereo vision unit that contains two cameras and software

According to [40], the core metrics for evaluating the performance of cameras and stereo vision systems include how quickly images are acquired, range of vision, resolution, accuracy of

depth perception, and field of view. Faster image acquisition allows for higher frame rate, which allows the system to stay more up-to-date. Longer range image sensors enable an earlier reaction to environmental obstacles. The further away objects can be evaluated, the more time is available to make decisions. Higher resolution allows for clearer images, thus increasing the accuracy of image processing. Similarly, more accurate depth perception allows for better decision making. Finally, a larger field of view allows for more information of the environment to be captured.

Depth perception was necessary in the front of the vehicle to perform effective imaging. The rear of the vehicle needs less detailed object detection because the vehicle will only be traveling in the forward direction, so ultrasonic sensors were used instead of a vision system such as a camera. Ultrasonic sensors are discussed in more detail later in this Chapter.

Table 9. Quantitative Decision Matrix for Choosing a Stereo Vision System

Design Considerations	**Weight**	**Tara - USB 3.0 Stereo Vision Camera**	**BlackBird 1 3D FPV Camera**	**Intel® RealSense™ Depth Camera D400-Series**	**2 x FL2G-13S2M/C**
Convenience	**7**	8	6	10	5
Documentation	**8**	6	5	10	7
Easy Learning Curve	**4**	6	5	8	4
Price	**6**	7	8	7	3
Range of Vision	**9**	4	0	7	0
FPS	**9**	10	10	10	10
Software	**8**	5	3	10	3
Stereo Vision	**7**	10	9	10	5
Total	**580**	406	327	527	274
Percentage		70%	56%	91%	47%

Table 9 contains a decision matrix analyzing the potential options for a Stereo Vision System. Convenience of implementation and existence of appropriate documentation allow for efficient implementation of the system. The Intel RealSense ranked highest in this field because it is programmed with stereo vision prior to purchasing and documentation is available for using this camera with autonomous vehicles. Intel RealSense also ranked highest in easy learning

curve due to the level of documentation. Price was an important factor due to a limited project budget. Low-cost cameras are necessary to save budget for other sensors and project incidentals. The range of vision and the frames per second of the cameras was ranked as the highest criterion because of the importance for image detection accuracy and quick reaction time. The Intel RealSense camera had the longest range of vision, around 10 m [41]. The software of the cameras was ranked high because of usability; ease of integration and coding is necessary to be efficient. Each camera system was examined for whether it had stereo vision. All camera systems did well in this category except for the two FL2G-13S2M/C cameras [41]. Overall, the Intel RealSense camera ranked the highest for use in this project and was used by the team.

Range Sensors

Based on the chosen Intel RealSense camera, the stereo vision is accurate to a range of 32.8 to 49.2 ft (10 to 15 m) [41], only allowing about 2 to 3 seconds of reaction time based on the top speed of the vehicle, which is 25 mph (11.2 meters per second). Therefore, a form of long-range detection must be implemented for long range path planning. And cameras, like the human eye, cannot see through poor weather conditions. In the event where the cameras cannot see their surroundings, the Baja vehicle would need an alternate detection method for obstacles in its path. Long range sensors were needed to detect such obstacles and their distance from the vehicle, thus allowing for a short reaction time. The two options for standard long-range distance sensing were light detection and ranging (LiDAR) and range finding radar.

Radar uses electromagnetic waves, projected forward within a cone area, to detect obstacles [42]. Electromagnetic waves bounce off obstacles and return to the radar receiver. The radar detects the range and movement of the objects by measuring the time and the frequency of the returning signal [42].

LiDAR is an active sensor that calculates the distance to an object by illuminating it with pulsed laser lights and then measuring the return time of the reflected pulses [43]. LiDAR can be separated into 2D and 3D LiDARs. 2D LiDAR usually contains a single laser emitter and a servo powered rotating reflection platform, to reflect the laser beams to all angles within the detection range. 2D LiDAR can construct a 2D map that represents a horizontal cross section image of the environment at the LiDAR mounting height level. 3D LiDAR uses multi-layered laser beams to draw several 2D cross sectional areas to construct a 3D representation of the environment.

Although radar and LiDAR can both provide long range detection, LiDAR has significant advantages over radar in the field of autonomous driving. Single-beamed laser light can illuminate objects much more accurately than radar waves. Different from radar sensors' wide cone shaped wave front, a LiDAR beam only focuses on a small area in the laser beam's direction. Additionally, LiDAR signals contains less noise than radar signals. Radar can sometimes generate noise due to unwanted reflection, however LiDAR has a lower tendency of reflecting. Lastly, LiDARs use active laser signals and do not depend on outside light sources to operate.

2D LiDARs can only gather distance information within a 2D plane, which cannot detect or measure any obstacles that are higher or lower than that plane. In the case of this project, the automatic decision would be for the Baja vehicle to travel around the object, because with only LiDAR there is no way to identify the object in the path of the vehicle. LiDAR sensors rely on inner reflective platform's rotation to emit to different angles, they do have a low scanning frequency that is limited by the spinning speed of the servo. LiDARs can generate noise and inaccuracy when encountering transparent or reflective objects such as windows and mirrors, however an off-road vehicle does not have to worry about these concerns.

Radar and LiDAR was determined to be too expensive to purchase, prompting the team to examine a 2-Dimensional SICK LMS291 LiDAR that was donated by WPI to the team. The SICK LMS291 LiDAR, worth approximately $6,000.00, was tested by the team in the spring of 2018. The team would not have been able to afford this device if it was not donated. This LiDAR is a long range 2D LiDAR with high accuracy. However, this LiDAR device is from 2002 and due to the age, the software and connection protocol is outdated.

In the Fall of 2018, the team gained access to a SICK 3D LiDAR, specifically the LD-MRS800001S01 (LD-MRS), for the sensor suite via a generous donation from SICK, a sponsor of the project [44]. This LiDAR is much smaller when compared to the 2D SICK LiDAR originally intended for this project, and it can generate point clouds. Point clouds are 3D arrays of points that depict a view of the LiDARs surroundings, up to 328 ft (100 m) away in real time. A point cloud will show where objects are on multiple vertical planes of view [23]. This differs from 2D LiDARs that can only see points on the same plane that the LiDAR is mounted on. Having high-resolution point cloud data will aid in object detection, environment perception, and risk mitigation. The LiDAR detects objects using laser beams in four stacked planes, measuring

the distance and direction of the object compared to the LiDAR. The LD-MRS can also track objects without additional hardware or software required; it can output the position, speed, size, direction of movement and age (scans, time) of the object in its field of view. The LiDAR has the power to track up to 128 objects at the same time which is then processed in real-time [45].

Figure 35 shows the process that occurs when the LiDAR is detecting an object. A laser pulse (2) is transmitted from the LD-MRS (4) to the object (1), and then the laser pulse (3) is reflected to the LD-MRS which collects the reflection and processes the information. The information, along with any other data processed by the LD-MRS, is always sent over Ethernet [45].

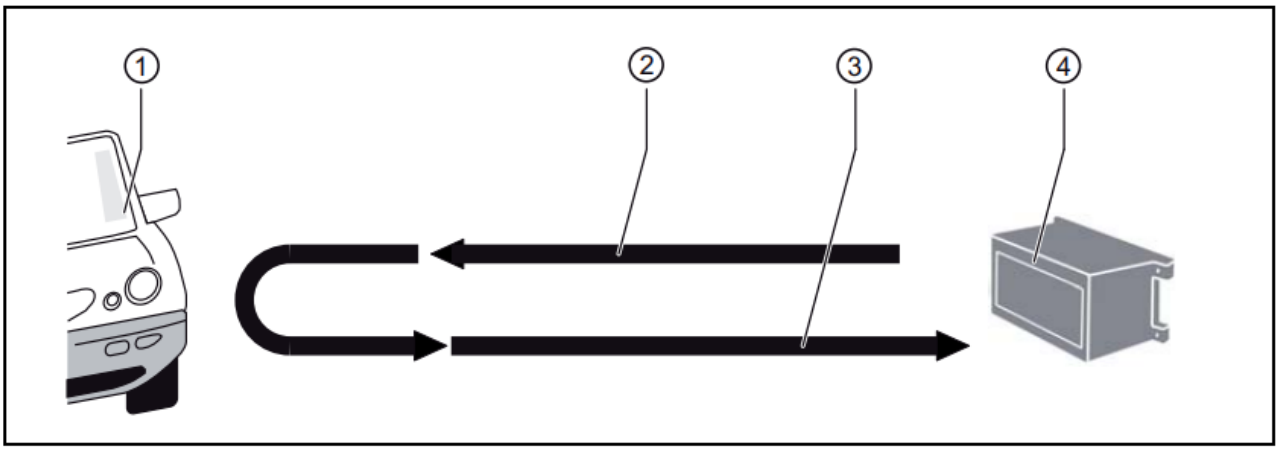


Figure 35. LD-MRS Time-of-Flight Measurement (Reprinted from [45])

As mentioned previously, the LD-MRS has multi-layer technology. The four scan planes, with various vertical angles, allow for pitch angle compensation. This means that if the LiDAR is attached to a vehicle (such as a Baja vehicle), it can detect objects accurately even if the vehicle is braking or accelerating [45]. Figure 36 shows the four scan planes in relation to the LD-MRS. (1) in the figure refers to the vertical aperture angle, (2) refers to one of the four scan planes, and (3) refers to the LD-MRS, or the LiDAR.

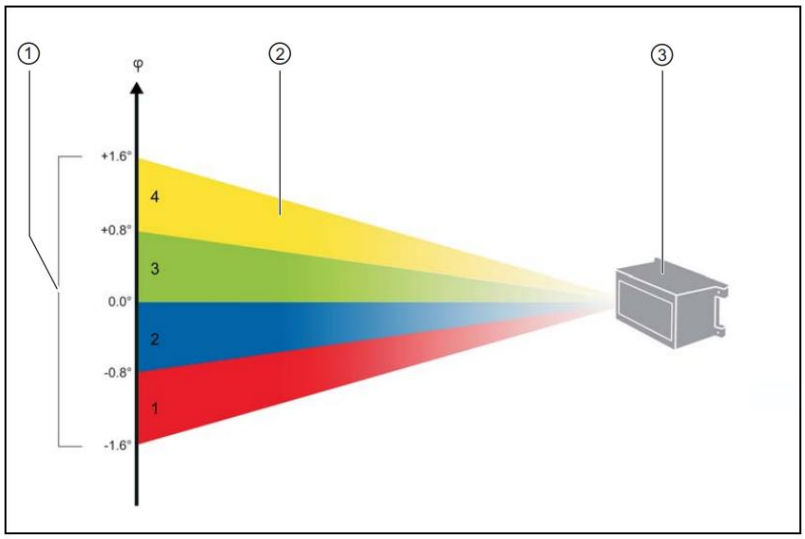


Figure 36. LD-MRS Four Scan Planes (Reprinted from [45])

The LD-MRS has a horizontal aperture angle of 85 degrees, but the scanning range can be extended to a total of 110 degrees. It has a vertical aperture angle of 4.2-6.4 degrees and operates at a scanning frequency of 12.5 to 50 Hz; object tracking occurs at 12.5 Hz. This LiDAR has a working range of up to 984 ft (300 m) at 100% remission and can detect objects of almost any shape. It also can function normally between -40 degrees Celsius and +70 degrees Celsius. Figure 37 visually shows the scanning range of the LiDAR. (1) in the figure, or the green area, is the central working range. (2) in the figure, or the light green area, is the lateral scanning range. (3) in the figure is the LD-MRS [45].

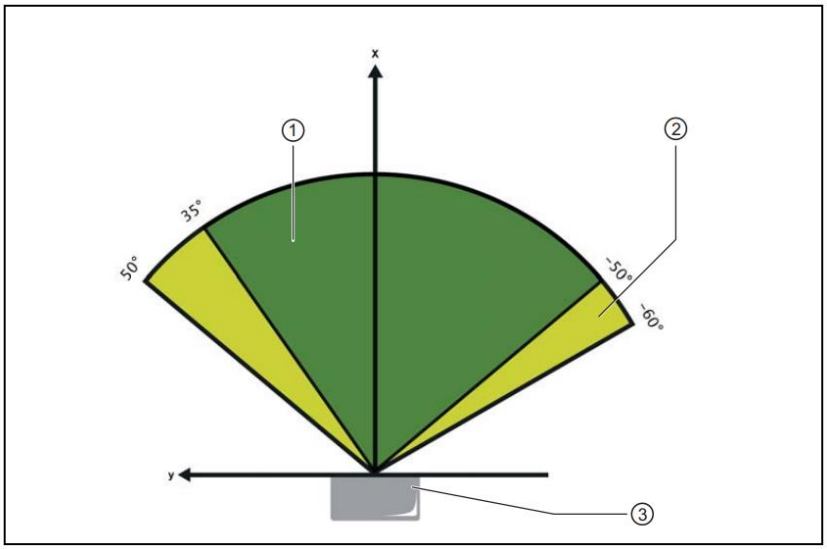


Figure 37. LD-MRS Scanning Range (Reprinted from [45])

The LD-MRS also has a noise filter that can reduce the effect of weather on its vision, such as rain [45]. This is especially useful for the LiDARs application in this project. The availability of a 3D LiDAR led to the discontinued use and testing of the 2D LiDAR. Testing showing data from the 2D LiDAR is demonstrated in Section 5.2 Methods, Testing, and Results.

Ultrasonic Sensors

The main detection system for the autonomous vehicle is made up of a camera and a LiDAR. However additional sensors, specifically ultrasonic sensors, are required to be able to sense obstacles that might appear on the sides of the vehicle, outside of the field of view of the camera or LiDAR. These side sensors do not need to display as much information as the cameras or LiDAR, and several can be used for the robot to have a general sense of the fields around it.

Ultrasonic sensors were chosen as side sensors mainly because they are relatively inexpensive, easy to buy in bulk, and provide a simple proximity distance from the vehicle to any objects. An ultrasonic sensor uses sound in order measure the distance to an object. The sensor sends out ultrasonic sound waves at specific frequencies, measures the time it takes for that specific frequency to hit an object and get back to the sensor, then calculates the distance. The function used to calculate is: $D = (v_{sound} * time) / 2$.

However, since the sensors send out sound waves, they are affected by the ambient temperature, which affects the speed of sound. Having multiple ultrasonic sensors could cause signals to collide into one another changing their waveform and invalidating the collected data. Another issue that could occur is material absorbing the sound wave. In the case of any signals being absorbed or lost, multiple readings will be taken and averaged to obtain a more accurate answer. Any values outside a certain threshold can also be filtered out. Since ultrasonic sensors are not the primary location sensor on the vehicle, it is not crucial they always return extremely accurate data, only an approximation of the area around the side and back of the car. To manage multiple sensors, the detection area of the chosen sensor will be examined, and the sensors will be placed to mitigate interference. Another option would be to have the sensors each ping distance during separate time intervals, so there never exist multiple waves at once. Table 10 showcases the options and design consideration weighting when choosing a specific ultrasonic sensor.

Table 10. Quantitative Decision Matrix for Ultrasonic Sensors

Design Considerations	Weight	LV-MaxSonar® - EZ™ Series	Ultrasonic Ranging Module HC - SR04	Murata MA300D1-1
Distance Range	**8**	8	6	0
Cone	**9**	9	6	4
Sensitivity	**6**	6	8	7
Cost	**6**	6	9	7
Documentation	**9**	9	8	5
Size	**2**	10	9	10
Zero Range	**4**	6	9	9
Total		342	330	221
Percentage		78%	75%	50%

The range of the sensor determines the maximum distance at which the sensor can detect any object. However, the purpose of the ultrasonic sensors is to detect objects that may collide from the side. The cone of the sensor determines the initial size of the waveform as well as the final shape and size of the waveform once it propagates to its maximum distance. The sensitivity of the sensor is defined by the maximum range divided by the resolution of the sensor. For this projects purpose, a highly sensitive system is not necessary as the objects to be sensed are primarily quite large. Increments of an inch of sensitivity is more than enough. Documentation of the sensor is important for ease of integration and data fusion. *Data fusion* is the process of combining information from multiple sources, typically sensors, to establish an accurate representation of the surrounding area [46]. Size of the sensor is nominal as the smaller the sensors the more flexible they are. The zero range of the sensor is at what point the sensor would not be able to accurately output distance data. At this distance the sensor would still be able to detect the presence of an object. This data can still be used as an indicator for the vehicle to make corrections. The team originally chose to use the LV-MaxSonar® -EZ™ Series [47].

In the Fall of 2018, SICK Sensor Intelligence suggested that in addition to a 3D LiDAR, the team select an ultrasonic model from their company to use for this project. SICK has a large selection of ultrasonic sensor families for various purposes, such as recognizing smaller objects versus larger objects [48]. When examining the sensors, the team looked for a sensor with a large

sensing cone and a weatherproof design. SICK offers two sensors that can see further than 16.4042 ft (five m) in distance, the UM30 and the UC30. The UM offered a more cylindrical shape while the UC offered a rectangular shape. Both sensors were resistant to dirty, dusty, humid and foggy conditions, however the team eventually decided on the UC30 [49] [50]. The size and structure of the sensor meant that incorporating the sensor on the vehicle would be easier when compared to the UM30, and the sensor fulfilled the necessary requirements of the ultrasonic sensors needed for the project discussed in Table 10.

The specific sensor offered to the team was the UC30-21416B, which has a five-meter range of vision, a 0.18mm resolution, and a 180 ms response time [51]. The team was sent five ultrasonic sensors, power cables, an IO-Link Master for the sensors as well as a coupling cable to link the IO-Master with the UC30s.

The Sick Ultrasonic sensors featured several applications over the MaxBotix sensors. They have several different switching modes such as distance to object, switching window, and object between sensor and background. Figure 38 shows the range cone of the sensors. “The dark blue area shown in these switch panels shows an example of the sensor’s working area if a round pipe is detected. The light blue area shows the maximum detection range which can be achieved under ideal conditions for easily detectable objects, such as the aligned plate given here” [52]. In the image, (1) stands for the sensing range dependent on reflection properties, size and orientation of the object. (2) stands for the limiting range and (3) stands for the working range. (4) represents an example object: an aligned plate 500 mm x 500 mm. Lastly (5) also represents an example object: a pipe with 27 mm diameter [52].

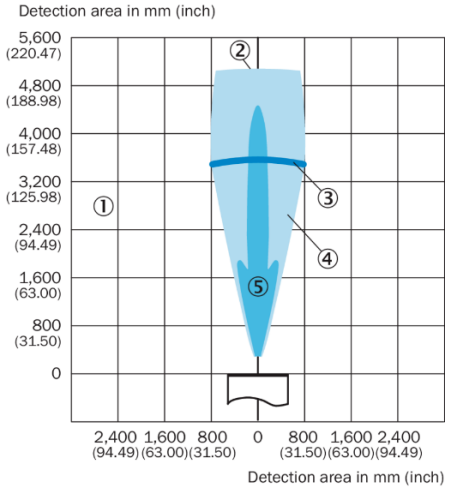


Figure 38. UC30-214 Detection Area (Reprinted from [51])

Figure 39 shows the sensing range of the UC30. The UC30 sensor, along with other SICK range sensors, sense objects using sonic pulses. The UC30s emit a sonic pulse that is reflected by the object that the ultrasonic is detecting. The distance between the UC30 and the object is calculated using the following equation:

$$Distance = speed\ of\ sound * \frac{total\ sonic\ time\ of\ flight}{2}.$$

These sonic pulses can be adjusted. By adjusting the sensitivity of the UC30, the sound cone of the sensors and its detection range can be altered. The SICK UC30 is the only ultrasonic sensor in the market with this feature [52].

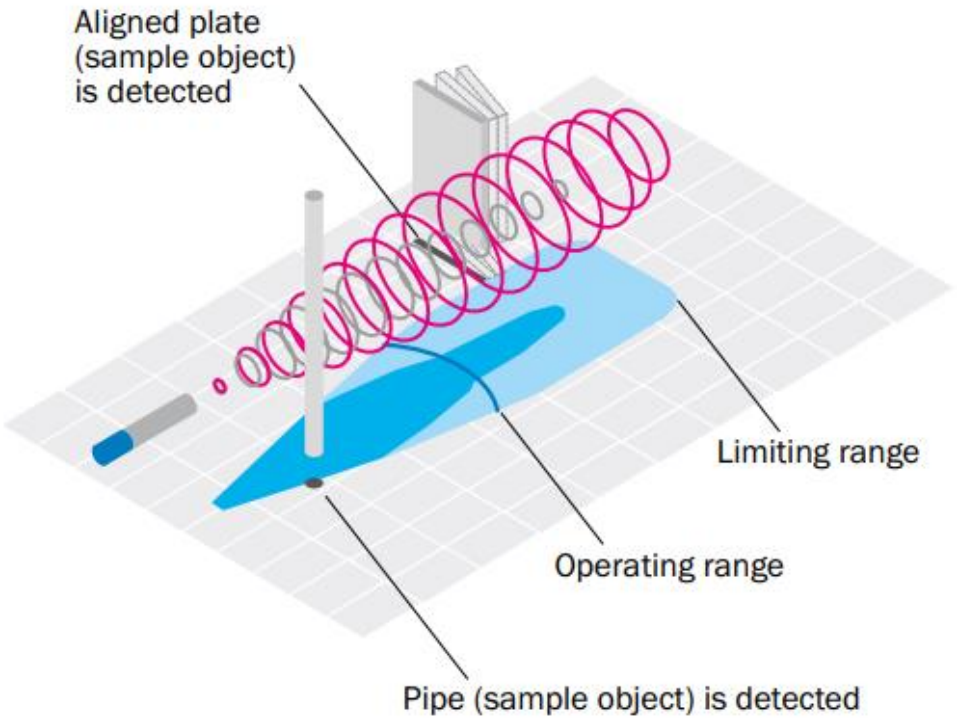


Figure 39. UC30 Sensing Range (Reprinted from [52])

The sensors were designed to communicate together meaning signal interference would not be a concern for the team. Up to fifty sensors can be combined without cross talk, and up to ten sensors can be synchronized together. The sensors and software also have a built-in filter that can be adjusted to the user's needs.

Since these sensors will be used simply for blind spot detection on the vehicle, the team will not be taking advantage of the extra features this device has to offer. The vehicle is equipped with five of these ultrasonic sensors, that are responsible for creating binary representations of

obstacles in the way of the vehicle during turns or backing up. This means the vehicle will know if an object is preventing it from turning, but it will not know *what* the object is that is. Results on these sensors are discussed in Section 5.2 Methods, Testing, and Results.

Global Position Systems

Vehicle perception using the RealSense, the SICK LiDAR, and the SICK ultrasonics is important, but it is not enough to create a level three autonomous vehicle; localization is another very important aspect of this project. To localize the vehicle in its environment, the team chose to use a GPS system. According to [53] "GPS is a constellation of satellites that provides a user with an accurate position on the surface of the earth. […] GPS can provide precise position and time information to a user anywhere in the world."

The team examined multiple GPS modules, looking for systems that would be easy to assemble and integrate with the entire communication system. The team researched popular modules on the market and created various design requirements with respective weights of importance in order to analyze different modules. Two GPS were examined in the decision matrix shown in Table 11.

Table 11. Quantitative Decision Matrix for GPS Modules

Design Requirements	Weight	Adafruit Ultimate GPS Breakout	Gowoops Ublox NEO-6M
Cost	**10**	8	10
Satellite Connections	**7**	9	3
Ease of use	**7**	10	7
Documentation	**5**	10	10
Update rate	**6**	8	5
Total	**1**	311	271
Percent total	**1**	89%	77%

The Adafruit Ultimate GPS Breakout [54], priced at $39.95, was designed to fulfill human-beings "every desire" [55]. It has a -165 dBm sensitivity, 10 Hz updates, and 66

channels. It has a 5V friendly design and only a 20 mA current draw. It is also breadboard friendly with two mounting holes. It can track up to 22 satellites on 66 different channels with a built-in antenna. It can update its location up to 10 times per second and requires very little current to power itself. The GPS module has a built-in microcontroller with empty FLASH memory that allows users to send commands to do internal logging and can be implemented with a simple "Start Logging" command. "The time, date, longitude, latitude, and height are logged every 15 seconds and only when there is a fix. The internal FLASH can store about 16 hours of data, it will automatically append data, so you don't have to worry about accidentally losing data if power is lost" [55]. Adafruit, the company that sells the device, also provides a software library for GPS usage [55].

The Gowoops Ublox NEO-6M, priced at $20.99, is cost-effective [56]. The module has a -161 dBm sensitivity, 5 Hz updates and 50 channels. It takes in a 3.6V input voltage and has a current draw of 10 mA [57]. While the data sheet does not document exactly how many satellites the Gowoops Ublox can track, reviewers on Amazon state that it can track approximately 3-5, compared to the 22 of the Adafruit [58]. The device does have an eye-catching feature: AssistNow Autonomous. This feature "uses previous broadcast satellite ephemeris data downloaded to and stored by the GPS receiver to automatically generate accurate satellite orbital data that is usable for future GPS position fixes" [57].

Overall, while slightly more expensive, the Adafruit Ultimate GPS Breakout proved to fulfill more of the design requirements and was chosen as the GPS module for this project.

Motion Sensors

Driving in off-road environments means that the GPS signal may cut-out intermittently. If the Baja vehicle was to rely solely on a GPS module, and it cut out, the vehicle is left with no method to detect its location. The team decided that the solution to this problem could be the addition of a motion sensor, or an inertial measurement unit (IMU). An IMU is a sensor that gives a 6-dimensional pose comprising of location (3-dimensions) and orientation (3-dimensions). This is accomplished by combining the readings from a gyroscope and accelerometer.

Motion sensors are useful for measuring acceleration, velocity, and position in both translation and rotation. The main challenge of using an IMU is velocity and position drift, hence

why an IMU is needed *in addition to* a GPS. The positional drift is caused by the single or double integrations needed to calculate velocity and position. Any small errors or drift in the acceleration measurements are amplified after the integrations. IMUs excel at calculating poses quickly and are useful as an interoceptive sensor to incorporate into feedback loops.

The 2010 WPI MQP team discussed in Section 2.2, Prometheus, decided not to use an IMU because of the DC bias in the calculated velocity which manifests as IMU drift. Prometheus did comment that the rotational data was relatively accurate and stable. This project used an InterSense NavChip IMU, which is a 6-axis motion sensor that does not have the same on-board drift correction ability as a 9-axis motion sensor [19].

The alternative to using a 9-axis motion sensor that does sensor fusion onboard, is to pair an IMU with a different sensor, such as a camera, and manually fuse pose data. This process typically uses an Extended Kalman filter (EKF) to fuse camera pose data and IMU pose data. Disadvantages to using an EKF include complexity of implementation and vision processing, which is a computationally slow and expensive operation. One area that increases the complexity of implementation is the need to know the exact time offset and exact physical pose offset to accurately fuse the camera pose data with the IMU pose data. This method was documented in reference [59] with relatively good performance using an $89 camera sensor and a $1700 IMU. This option is not suited to this project given the level of complexity to achieve a reliable pose correction and the fact that simple to use consumer-grade 9-axis motion sensors have good performance for a low cost. Therefore, IMU and camera sensor fusion for pose data was not considered.

Based on [59], there are two major classifications of motion sensors: consumer grade and industrial grade. Various design requirements were created and weighted based off the document and were examined in the decision matrix shown in Table 12.

Table 12. Quantitative Decision Matrix for Motion Sensor

Design Requirements	Weight	Consumer Grade Motion Sensor	Industrial Grade Motion Sensor
Cost	**10**	10	1
Durability	**3**	5	10
Static Error	**6**	7	9
Dynamic Error	**8**	6	8
Ease of use	**7**	10	3
Documentation	**5**	10	7
Update rate	**5**	6	8
Total	**1**	355	254
Percent total	**1**	81%	58%

Optimizing performance per dollar is crucial to keeping the entire system feasible. The industrial grade motion sensors typically come in self-contained cases that are protective in various ways while many consumer-grade motion sensors have the electronics exposed and are not packaged for protection. The reason durability is of lower importance is because the team can always design an enclosure if necessary and the intent is to test in fair weather conditions when outdoors. Performance metrics such as static error and dynamic error or more relevant as they indicate how useful and reliable the data will be. Refresh rate impacts how much time the system must respond to changes in the sensor data. This standard was obtained by looking at specification sheets for various sensors. The ease of use is based on the available resources, simplicity of the sensor interface and communication protocol, personal experience, and level of manual sensor fusion. Based on the decision matrix, it was found that the consumer grade motion sensor is more applicable for this project's design considerations and led the team to do more research into exactly which consumer grade motion sensor should be purchased.

Microelectromechanical Systems (MEMS) are designed to measure pose through electrical and mechanical methods in a small package. According to [60], the trend in IMU sensors has been to fuse the data from several separate MEMS sensors together to negate the disadvantages of each individual sensor. [60] compared orientation accuracy of three popular consumer-grade Attitude Heading Reference System (AHRS) motion sensors versus an

industrial-grade AHRS motion sensor. An AHRS combines the data from a three-axis gyroscope, a three-axis accelerometer, and a three-axis magnetometer.

Reference [60] examined two AHRS motion sensors: the MTi-300 and the BNO055. It concluded that the MTi-300 industrial sensor had the lowest static and dynamic error in orientation accuracy, and that the BNO055 Bosch sensor had the best dynamic performance of the three consumer-grade AHRS motion sensors. The MTi-300 costs roughly $3500, while the BNO055 costs about $35. Table 13 and Table 14 contain a performance comparison between the NTi-300 and BNO055 [60].

Table 13. Static Orientation Accuracy Performance (from [60])

Sensor	**Pitch (degrees)**	**Roll (degrees)**	**Yaw (degrees)**
MTi-300	0.03	0.02	0.05
BNO055	0.08	0.03	0.71

Table 14: Dynamic Orientation Accuracy Performance (from [60])

Sensor	**Pitch (degrees)**	**Roll (degrees)**	**Yaw (degrees)**
MTi-300	0.40	0.40	0.55
BNO055	1.40	0.80	4.60

The Bosch sensor performed the best of the consumer-grade motion sensors, making it a top contender. The Bosch sensor can analyze dynamically moving vehicles, demonstrated when it was used on a wheelchair controlled by the BNO055 motion sensor, which was mounted to the occupant's head. The wheelchair used a second BNO055 motion sensor connected to the wheelchair for slope correction. The project was documented in reference [61]. The sensor does data processing, sensor fusion, and time considerations on its own processor [62]. It communicates using I^2C or UART (serial communication channels) with a 10 Hz pose update rate. The BNO055 motion sensor [63] was purchased by the team for this project for the IMU module.

Main Computing Unit

As the project involves data from a multitude of sensors, data fusion is necessary. Unfortunately, data fusion requires a substantial amount of computing resources, especially when referring to computer vision. The team chose to use a Main Computing Unit (MCU) to handle larger tasks, and smaller embedded process to handle smaller tasks, dictated by the MCU. To help with processing speed and power management, any lower level functions can be off loaded from an MCU onto smaller embedded processors running set functions. Off-loading actuation and smaller tasks will give an MCU one less task to perform.

The MCU is the brain of the system and it can acquire real world feedback and make computations on sensor data. Autonomous projects such as WPI Professor Xinming Huang's autonomous RC Car MQP [64], MIT's RACECAR project [65], and Sweden's "Real-time, GPU-based Pose Estimation of a UAV for Autonomous Takeoff and Landing" project [66], used NVIDIA's Jetson platform because of its powerful graphics processing unit (GPU). CUDA cores are individually programmable cores that run processes in parallel, as opposed to the central processing unit (CPU), which runs commands sequentially. Intense computational processes such as Neural Networks, Computer Vision, and AI are pushed onto the GPU [66].

Intel has an UP board, a direct competitor to the Jetson. Intel's board is cheaper; however, it is not as intuitive to program. The Raspberry Pi platform was another consideration, however there are no projects where the Raspberry Pi is the CPU for a self-driving vehicle moving faster than 5 miles an hour.

From a lower level perspective, the Arduino microcontroller platform has revolutionized the project space by allowing for quick prototyping. This was done by having the integrated development environment (IDE) address the registers directly, instead of searching through the boards documentation to find the correct registers to address. This makes it easier to quickly put together motor control and sensor reading circuitry. Additionally, these microcontrollers are low power, something that the previously aforementioned microprocessors cannot claim.

To make educated decision-making in choosing an MCU, microcontrollers and microprocessors need to be compared. A microcontroller and a microprocessor both have a CPU and peripherals. Microcontrollers can be found in devices such as: digital cameras, washing machines, and microwaves. Microcontrollers are low on memory relative to processors;

therefore, their functionality is limited, however due to their limited memory and low performance they are extremely power efficient.

Microprocessors are comparable in computing power to standard computers. They typically run an operating system, have more memory, and can quickly handle heavy computing tasks. They can also run multiple applications, whereas typically a microcontroller can just run one program repeatedly. This makes microprocessors better at running simulation programs while processing data simultaneously.

When performing computer vision, data-fusion, and running simulation software, like Gazebo and rViz, a microprocessor is the preferred approach. However, for actuating the motors and less intensive processes, microcontrollers are more power cautious and an efficient choice. When choosing the microprocessor, a quantitative decision matrix was made to justify each microprocessor as seen in Table 15.

Table 15: MCU Decision Matrix

Design Considerations	**Weight**	**Up Board 2**	**Jetson TX2**	**Jetson TK1**	**Raspberry Pi 3**
Support	**8**	7	9	9	10
Documentation	**7**	6	10	8	10
Price	**6**	6	6	8	10
Easy Learning Curve	**4**	5	6	6	8
Software	**8**	10	10	10	5
Performance	**8**	9	10	5	4
Total		306	362	320	314
Percentage		75%	88%	78%	77%

Based on this table, the team chose to use the Jetson TX2 as the microprocessor for this project. The Jetson TX2 has a 64-bit multi-process CPU complex and a 128-bit memory controller. The Jetson TX2 also has an Inter-Chip Communication (I^2C) Controller, which supports serial device communications to a multitude of various devices. It supports a 7-bit slave address as well as master and slave modes of operation, at speeds of up to 1Mbit/s [67]. This feature is important for the communications bus and is discussed in more detail in Chapter 6.

Affordable and easily programmable microcontrollers such as the Arduino allow for quick prototyping and easy integration into larger projects such as this one. Microcontrollers are effective at controlling peripherals such as motors and actuators. Therefore, a microcontroller can control the steering, braking, and acceleration, while the MCU performs the data processing and visualization. The Arduino can also be used to turn on and off the autonomous capability of the vehicle.

The Arduino can implement Proportional Integral Derivative (PID) control, a feedback loop commonly used in robotic systems. The error between the desired sensor value and the current sensor value is constantly fed into this control loop, to reach the desired value quickly, smoothly, and accurately. The feedback loop can be seen in Figure 40 [68].

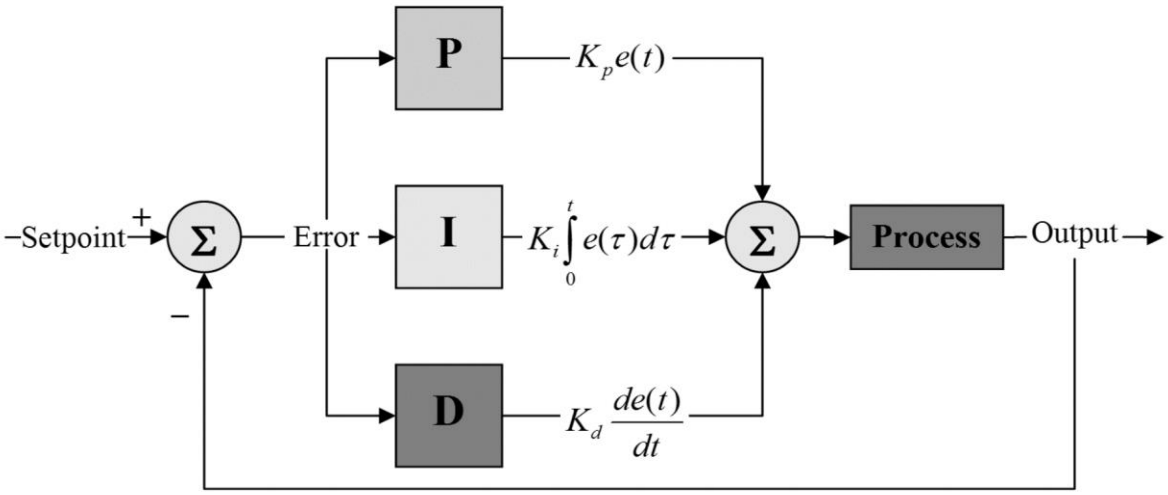


Figure 40: Commonly Used PID Loop Control (Reproduced from [68])

Implementing this motion will make for fluid acceleration, deceleration, steering, and power steering. The easiest way for the MCU to communicate to the Arduino is using messages sent over serial. Using an Arduino as a helper to the MCU means that even if the MCU fails, the Baja vehicle can still be controlled manually, but with the added benefit of power steering. The specific Arduino that will be used is the Arduino ATmega-328, otherwise commonly known as an Arduino Uno. [69]

Final Proposed Sensor Array

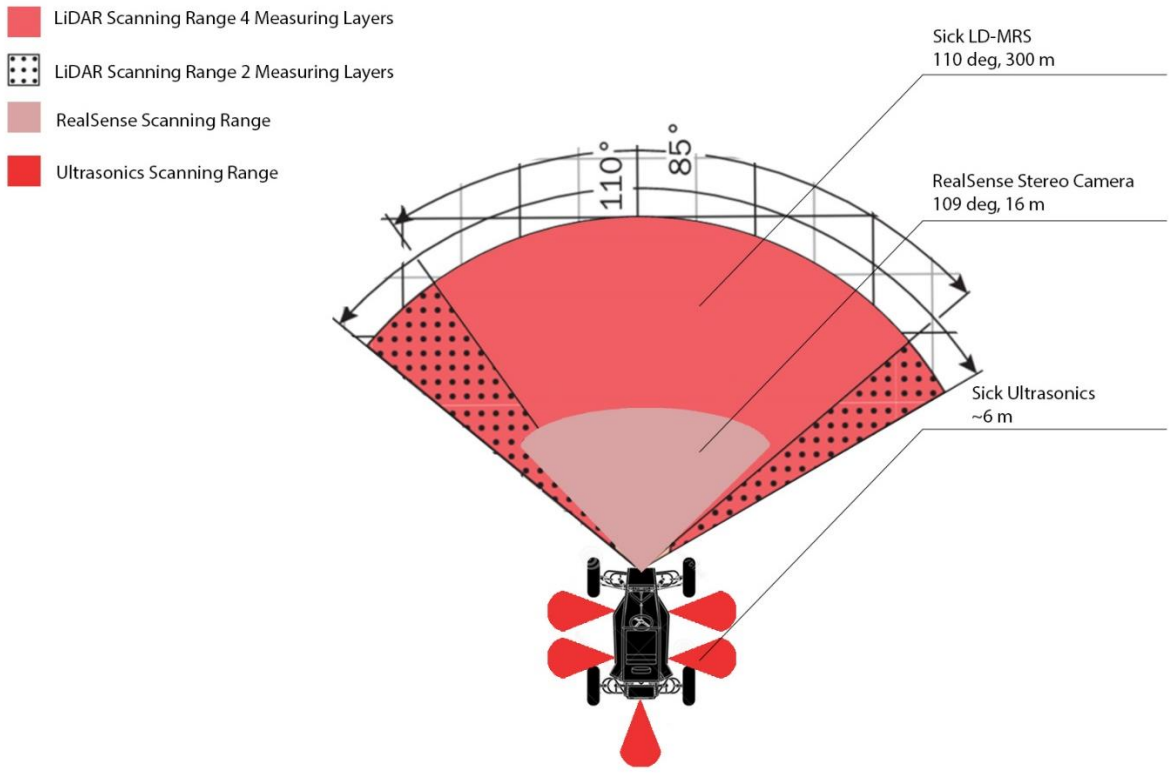


Figure 41. Sensor Breakdown of Baja Vehicle (Reproduced from [44])

Figure 41 shows a bird's eye view of the vehicle outfitted with an array of the proposed sensors to help it "see" its environment. It is important that the vehicle can see and analyze the road in front of it just as would a human. To accomplish this high-level of perception, the team chose to equip the vehicle with:

- A 3D LiDAR (Sick LD-MRS) [44]
- A 3D Stereo Camera (Intel RealSense) [41]
- A GPS [54]
- An IMU [63]
- And five ultrasonic sensors (SICK UC30) [51]

Each of these sensors plays an integral role in the ability of the vehicle to navigate autonomously. The LiDAR sees a slot of its environment in front of it; it can see 110 degrees in front of it horizontally, at a range of 984 ft (300 m), but unfortunately is limited in its vertical

field of view. To help the vehicle see more detailed images at a closer range with a wider vertical field of view, the team chose to equip the vehicle with an Intel RealSense D435 that supplements the LiDARs capabilities and is available for purchase at a more affordable price when compared to a LiDAR. The RealSense also comes equipped with some out-of-the-box 3D software re-construction features and the ability to integrate it with operating systems [41].

As Figure 41 shows, these two sensors combined cannot create an image of the entire 360-degree environment around the vehicle. To see in the LiDAR and RealSenses blind spots, the team chose to equip the vehicle with five ultrasonic sensors that create binary representations of obstacles in the way of the vehicle during turns or backing up. This means that the vehicle will know if an object is preventing it from turning, but it will not know *what* the blocking object is. This sensor suite allows the team to write software that can interpret the data in front of the vehicle, while also keeping in mind any obstacles on its sides.

Vehicle perception is important, but it is not enough to create a level three autonomous vehicle; localization is another very important aspect of this project. To localize the vehicle in its environment, the team chose to use a GPS unit. Although this is adequate in most environments, driving in off-road environments means that the GPS signal may cut-out intermittently. If the vehicle was to rely solely on this technology, and the GPS signal cut out, the vehicle is left with no method to detect its location globally. The solution the team has come up with is using the GPS with an IMU, as well as using a software solution to make up for IMU drifting. If the GPS was to cut out, the IMU or the software could use the last known position to help localize the vehicle for a short period of time until the GPS comes back. Creating a backup for the GPS will have a large software component. The data from the GPS, the IMU, and the software will be combined to accurately pinpoint the vehicle globally and relatively in its environment. The software solution to this is discussed in more detail in Chapter 6. The full electrical system will be controlled by a Jetson TX2 and multiple Arduino Unos.

5.2 Methods, Testing, and Results

This section of the report will discuss the team’s electrical development work, as well as the results of testing the electrical development.

The team examined several sensors when building the sensor array for this project, including a stereo vision camera as the primary method of data acquisition, LiDAR as a secondary source of information, several short-range ultrasonic sensors as a form of acquiring information close to the vehicle, and a motion sensor for location and velocity detection. Ultimately the team decided on the use of the Intel® RealSense™ Depth Camera D400-Series [41], a SICK 3D LiDAR [44], SICK UC30 ultrasonics [51], and a consumer grade GPS [54] and motion sensor [63]. All sensors were to be controlled by the master microcontroller, the Jetson TX2, as well as several Arduino ATmega-328 slaves. For reference, these are the same slaves that are responsible for controlling the acceleration, braking and steering described in Chapter 4.

Refer to Figure 42 through Figure 45 for models of the vehicle frame with the outfitted actuators and sensors. Figure 42 and Figure 44 highlight the placement of the actuators, the braking system, and the steering system as well as the placement of the range sensor or the LiDAR. Figure 43 shows the placement of the ultrasonic sensors on the sides and rear of the vehicle frame. Figure 45 presents the placement and angle of the RealSense camera. These sensors and actuators were obtained and mounted by this MQP team, and each sensor mount and actuating system was specifically designed by this team to interface with the Baja vehicle.

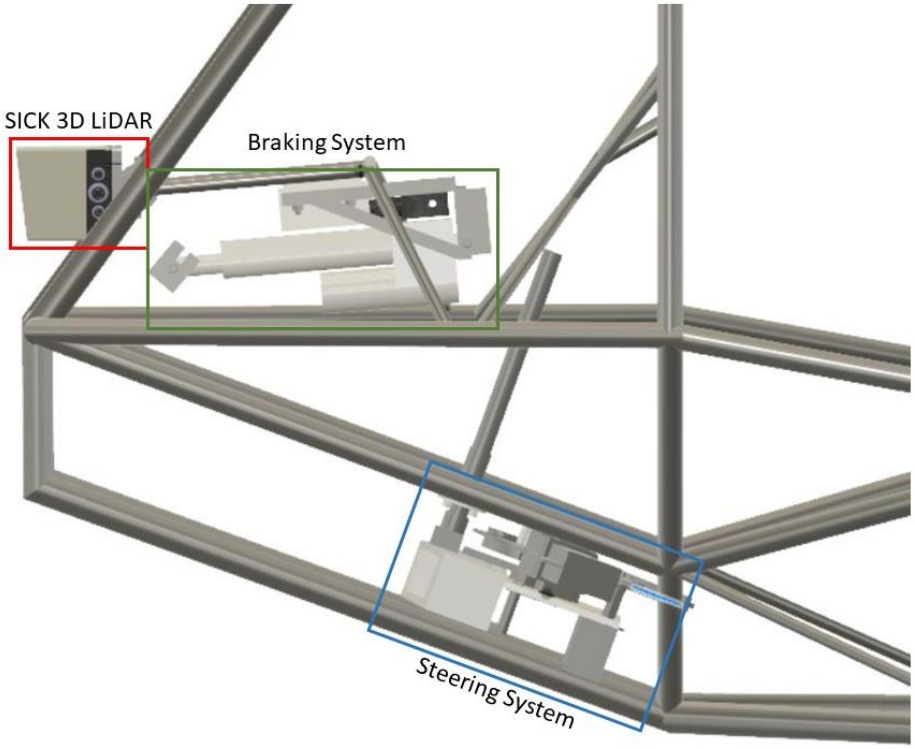


Figure 42. Baja Vehicle Outfitted with Sensors and Actuators from the Right Side

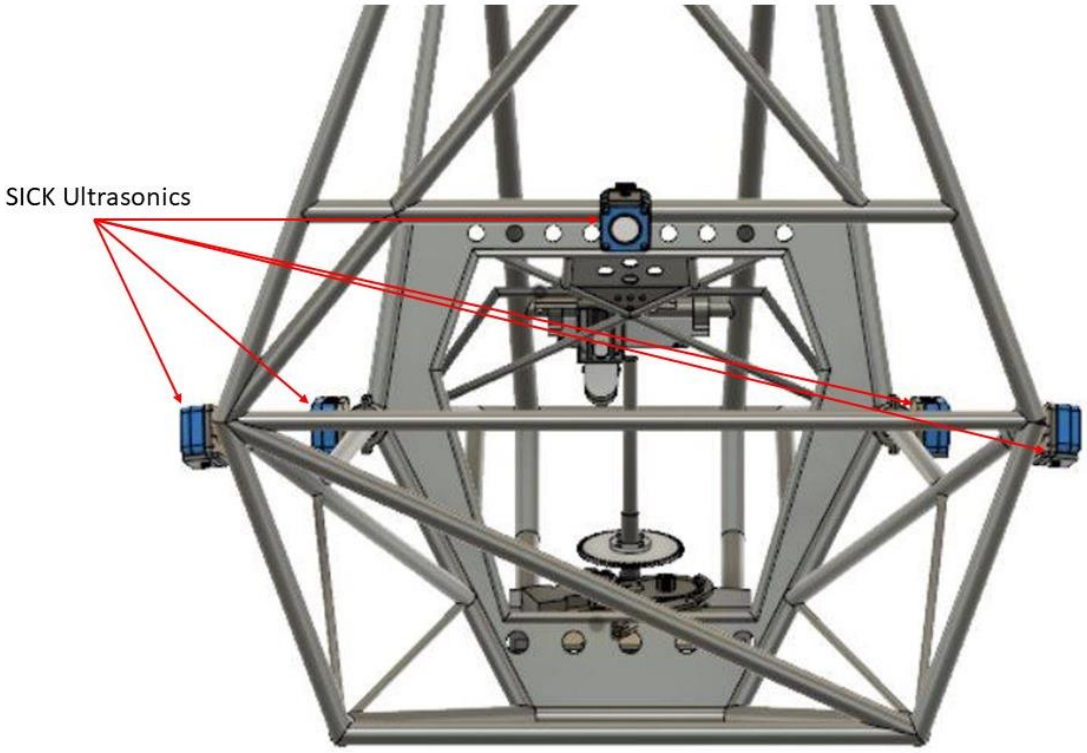


Figure 43. Baja Vehicle Outfitted with Sensors and Actuators from the Rear

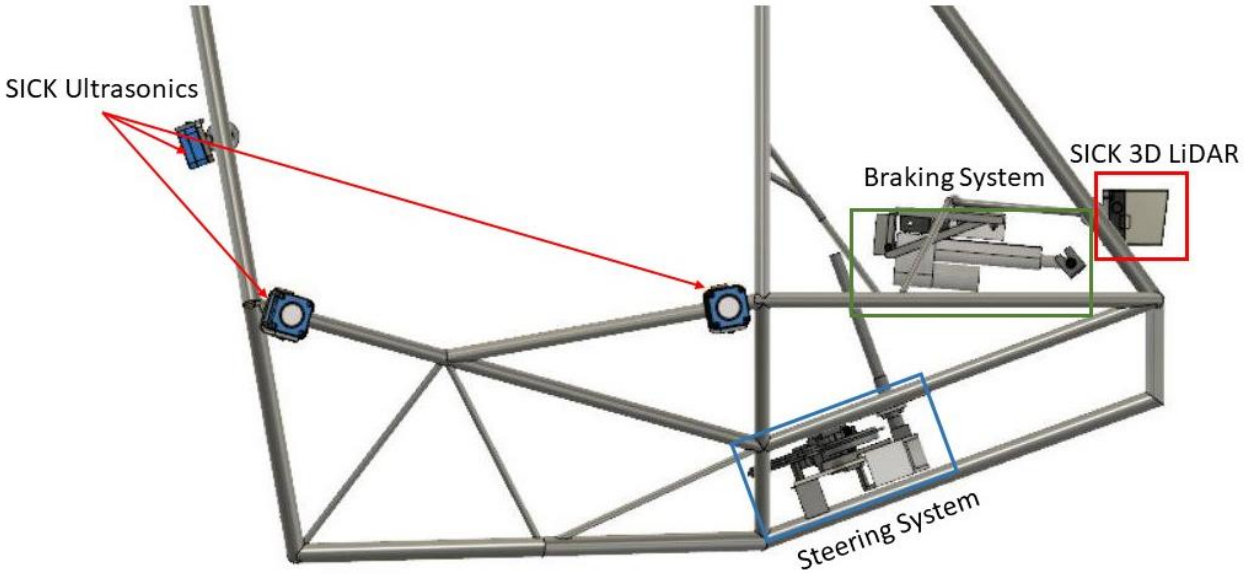


Figure 44. Baja Vehicle Outfitted with Sensors and Actuators from the Left Side

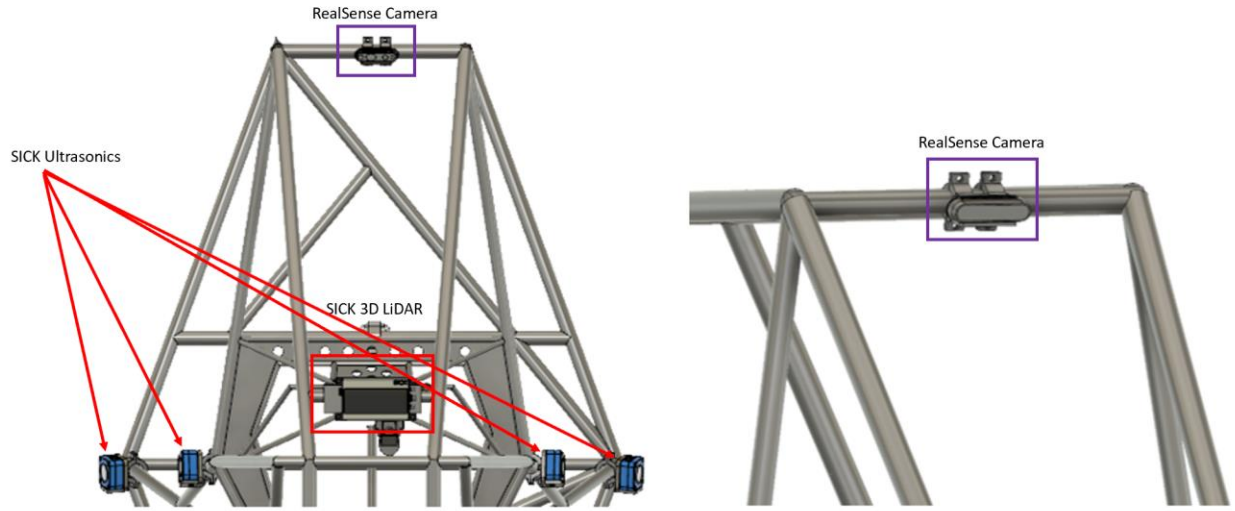


Figure 45. Baja Vehicle Showcasing RealSense Placement

For short range detection, the team obtained a RealSense D435 [41]. This model utilizes stereo and IR dot projection to sense RGB images and depth as well as generate object point cloud in a range up to approximately 29.5 ft (9 m). The camera was calibrated to find the best settings for long range depth sensing and tested in an indoor environment. The results are shown in Figure 46. Warmer colors, such as red, represent objects that are further away in comparison to cooler colors, such as dark blue.

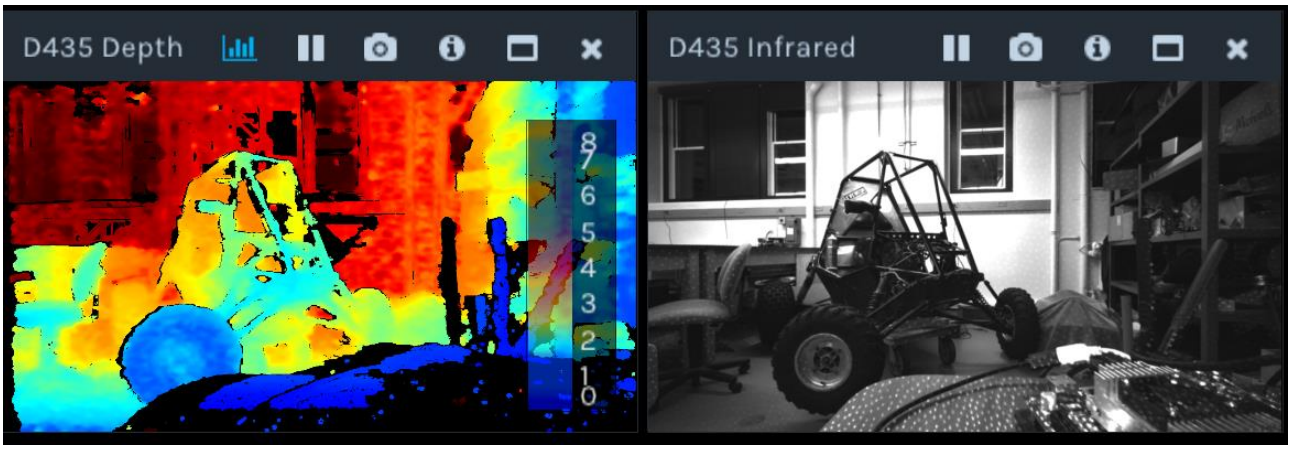


Figure 46. RealSense Depth Data (left) & Infrared Dot Projection on the Scene(right)

The RealSense is adequate for short range (less than or equal to 29.5 ft or 9 m) surface and obstacle sensing. However, the point cloud data generated by the RealSense demonstrates the effects of spraying and vibration. This inconsistency of the point cloud increases the

difficulty to conduct object recognition and plane fitting. Thus, the team is looking into the possibility of using the 3D LiDAR instead of the RealSense camera for accurate point cloud recognition.

For the sensors and microcontrollers to communicate with each other, the team decided to employ Robotic Operating System, or ROS, as the base structure of the system [70]. ROS allows every sensor on the vehicle to publish its real-time data to ROS topics which can then be used for sensor fusion and control algorithms. The team was able to publish the cameras point cloud and depth data to ROS topics and visualize the data in ROS using the built-in tool RVIZ. The point cloud is shown in Figure 47, representing the wall and the bench in the MQP lab located in Higgins Lab 045.

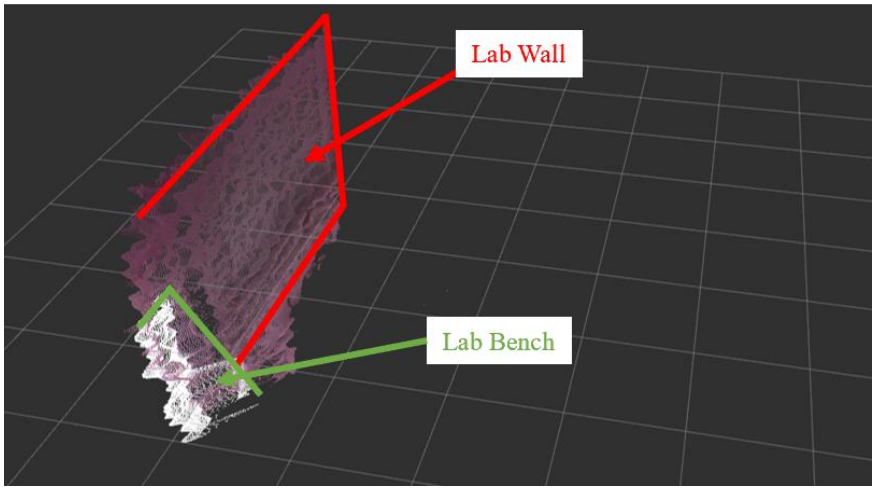


Figure 47. Vehicle Electronics Box Featuring IMU Module

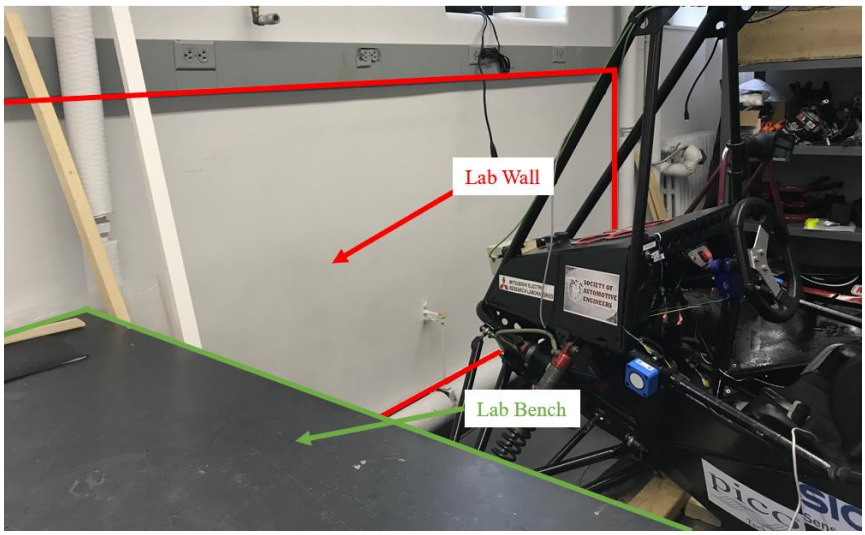


Figure 48. Lab Wall and Bench in Higgins

For a search and rescue vehicle, consistent self-localization and continuous mapping of the environment are crucial for the path planning of the vehicle. As the RealSense camera outputs depth data and RGB streams, feature points can be extracted to track the change in environment along with the movement of the vehicle. The team used the "ORB_SLAM2" package [71] to synchronize all key features to build a continuously expanding and improving 3D map of the environment. This map is useful to assist the team in planning a complete and optimal path to the destination of the vehicle, as well as to keep previous knowledge of the path travelled. This assists in reducing the difficulties when backing-up previously used maps, and when planning for return trips. The SLAM demo is shown in Figure 49.

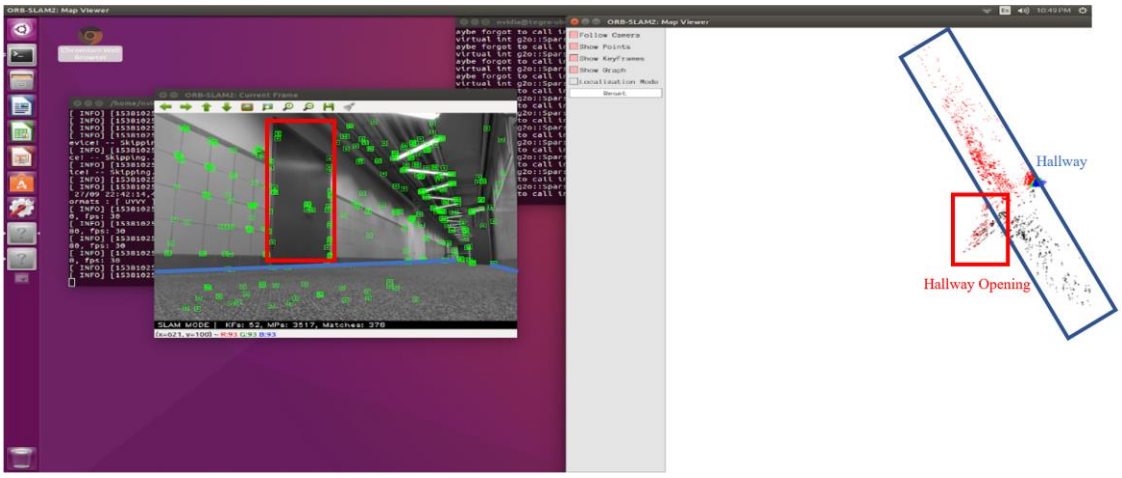


Figure 49. Object Tracking and Partial Hallway Map Generation Using SLAM

Using Fusion 360, the team also designed and 3D printed a mount to place the RealSense on the vehicle. This model is shown in Figure 50.

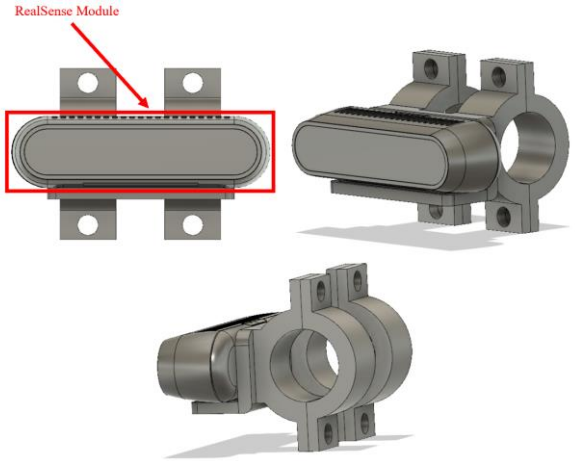


Figure 50. RealSense and Custom Mount Model

During testing in the Fall of 2018, before the 3D LiDAR was donated to this project, the team was able to connect with the 2D LiDAR through RS232 serial connection. The LiDAR uses binary encoding instead of ASCII and transfers data efficiently. Figure 51 and Figure 52 show how the LMS291 LiDAR [72] will outline a cross section boundary of the environment. From the graph origin (0,0), the area that is enclosed by the green outline indicates open areas around the sensor. The images below show the outline cross section of the MQP lab located in Higgins Lab Room 045.

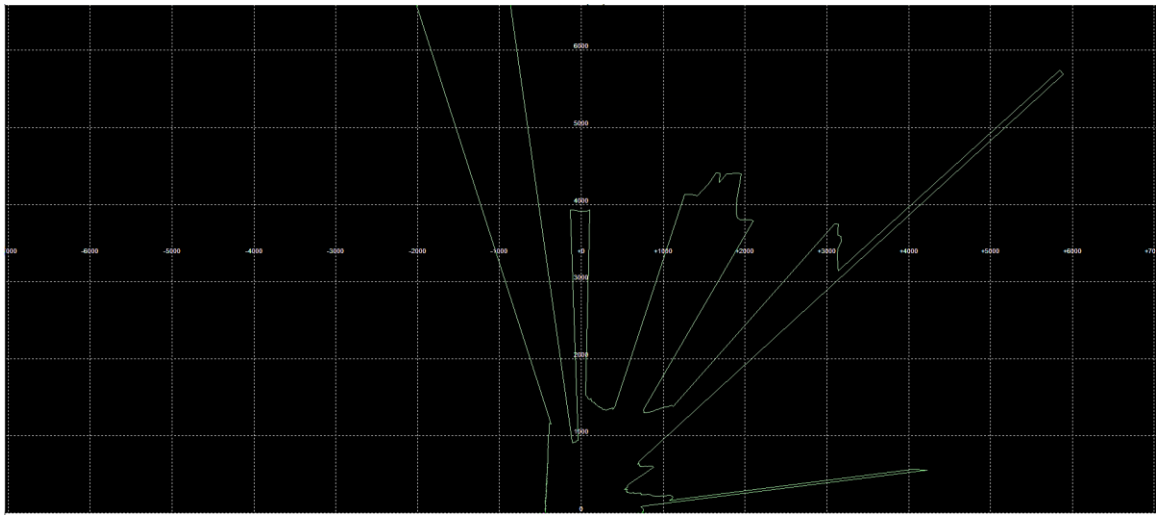


Figure 51: LMS291 Data in Configuration 8m range, 180 deg field, 0.5 deg resolution

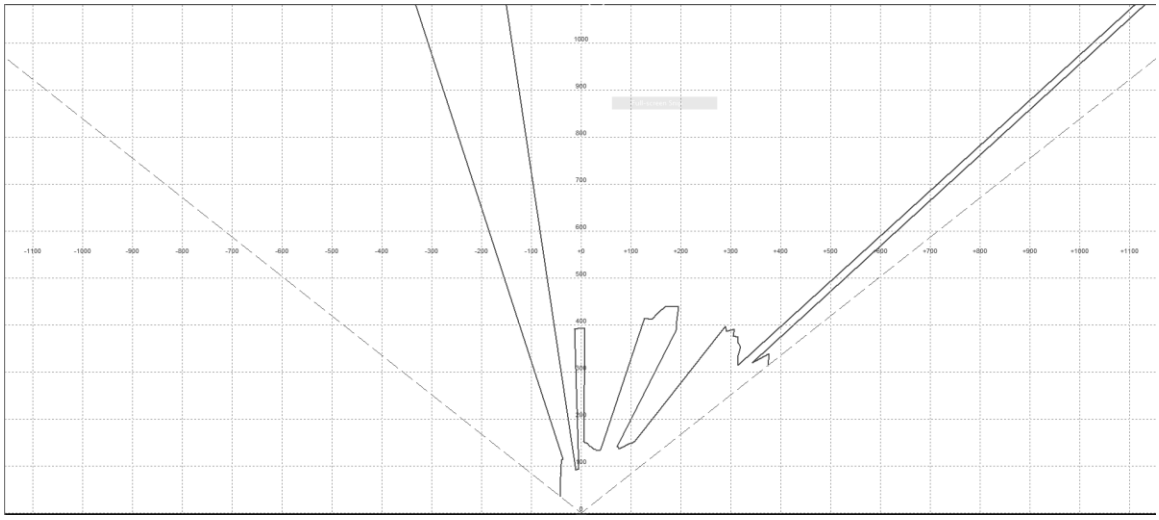


Figure 52: LMS291 Data in Configuration 80m range, 100 deg field, 1 deg resolution

Although the 2D LiDAR demonstrated extremely precise distance data, the low rotating rate of the LiDARs reflection platform caused the scanning frequency to be limited. The team

lowered the angular resolution and scanning angle range to achieve a scanning frequency of only around 2Hz.

In the spring of 2019, SICK Sensor Intelligence donated a 3D LiDAR [44] to the team. The SICK 3D LiDAR comes with its own power source and ROS connection solution, enabling it to be connected to the MCU directly through ethernet connection. The power ideally is sent to the LiDAR through a 48 V to 12 V buck voltage converter. It could also be powered through a 48 V to 24 V buck voltage converter. The LiDAR takes in around 0.3 A to 0.5 A in normal operation and up to 1.5 A when booting up.

The 3D LiDARs point cloud data is sent from the LiDAR to the MCU through a SICK internal UART protocol, and then the data is decoded and filtered with the pre-built SICK ROS package. The software that decodes and filters the data is described in more detail in Chapter 6.

The team also 3D modeled the LiDAR in Fusion 360 to create a mount for it to attach to the vehicle. The LiDAR and a custom mount are modeled in Figure 53. This mount was 3D printed, and the 3D print is used to mount the sensor to the front of the vehicle.

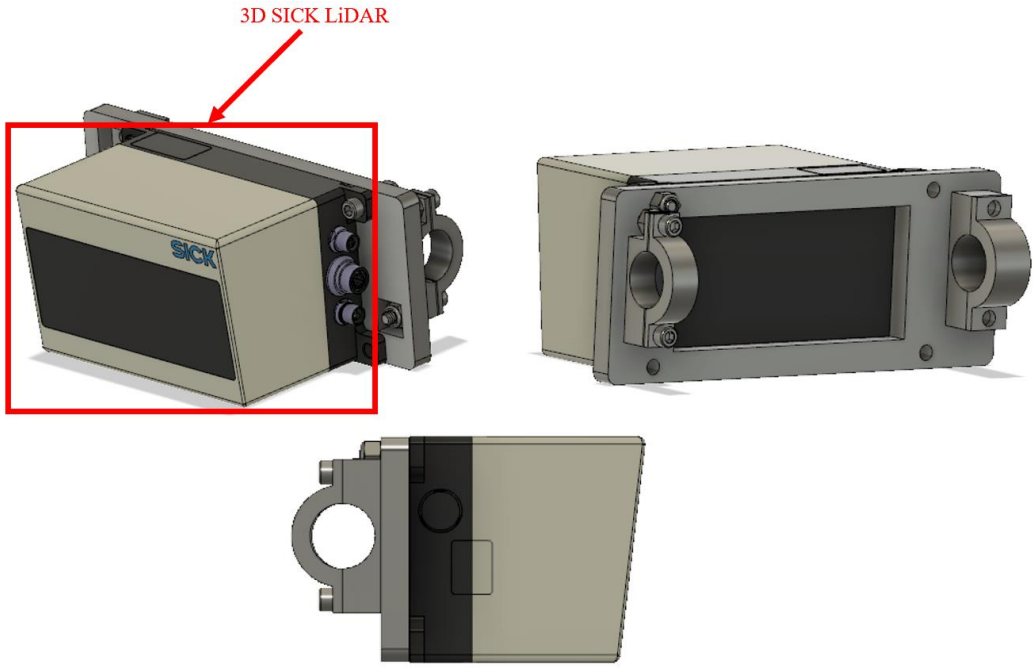


Figure 53. SICK 3D LiDAR and Custom Mount Model

The SICK ultrasonics did not arrive until the end of the holiday break in December 2018, but the team decided to 3D modeled the ultrasonic sensors in Fusion 360 early to create a mount for them to attach to the vehicle. This was done so they could be mounted when they arrived.

The ultrasonics and their custom mounts are modeled in Figure 54. The team designed two different mounts, one for vertical attachment to the vehicle and one for horizontal attachment to the vehicle but ended up only needing the vertical mounts. Four of these mounts were 3D printed and were used to mount the sensors to the vehicle.

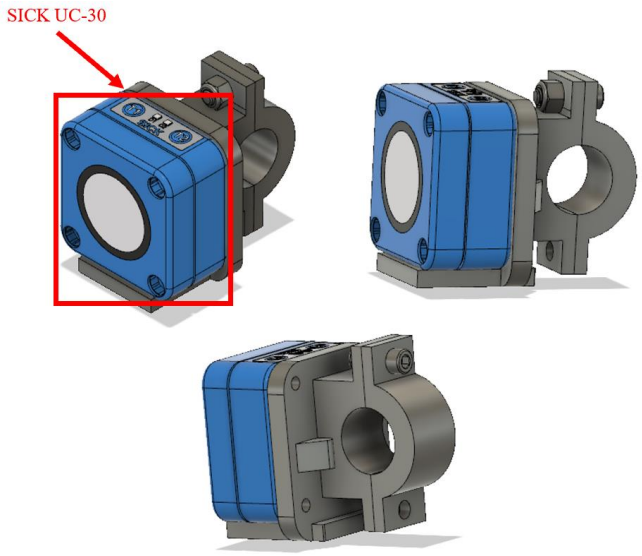


Figure 54. SICK Ultrasonics and Custom Side-Mount Model

The team also designed, and 3D printed a mount for the fifth ultrasonic, which would be placed on the rear of the vehicle. This mount is shown in Figure 55.

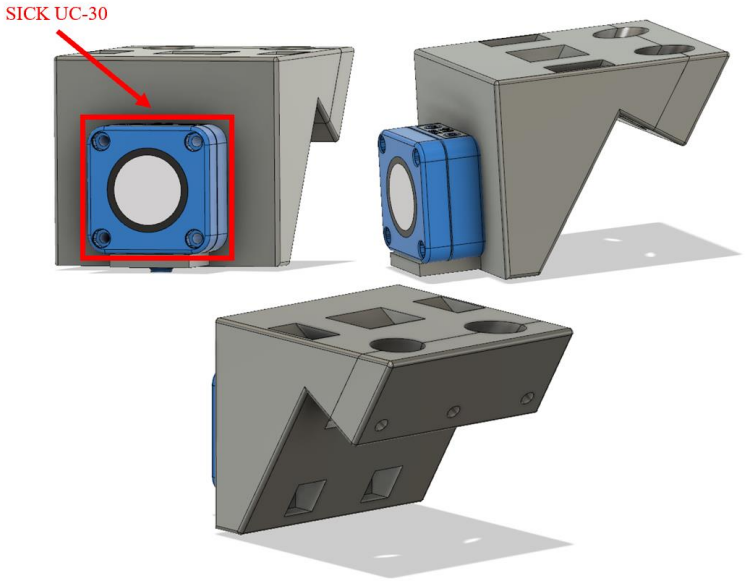


Figure 55. SICK Ultrasonics and Custom Rear-Mount Model

Unfortunately, after weeks of trying to connect the UC30s to any team member's computer, the team determined that the issue was a hardware problem. Moving forward the team decided to not use them but kept them on the vehicle to provide reference for future teams when functional units are available for installation. New ultrasonics are being shipped for the next team to use.

The BNO055 motion sensor [63] that was purchased as the IMU module was originally chosen to assist with the localization of the vehicle and velocity control, but it actually ending up serving a different purpose. As the MCUs resources were starting to run low, the team decided to off-load some processing that was originally going to be on the MCU, to the slave Arduino that is responsible for actuating the throttle. The idea was simple: allow for the slave to determine the velocity of the vehicle and adjust it accordingly, all based on the IMUs data. The team spent some time trying to integrate the IMUs acceleration data to accurately predict the velocity of the vehicle, testing the approach in a team member's car at highway speeds. After extensive testing and researching, the team found that the IMU produces virtually unusable results. The information provided by the IMU quickly became too noisy and unreliable, and also required drastic motions, such as aggressive accelerating or braking, in order to receive usable information.

Although unusable for velocity estimation, the team applied the IMU for gravity compensation. The IMU was used to determine the grade of the slope that the vehicle is on, and assist the vehicle in maintaining a constant speed, regardless of the slope the vehicle is ascending or descending. The average slope measurements are taken, multiplied by some gain, converted into a value that is taken in by a Digital-to-Analog Converter (DAC) made by the team, and then added to the analog values being sent to the motor controller of the vehicle. These values are then interpreted using a Analog-to-Digital Converter (ADC). The IMU is mounted on a perf board and powered by the Master Arduinos 5 V pin, shown in Figure 56.

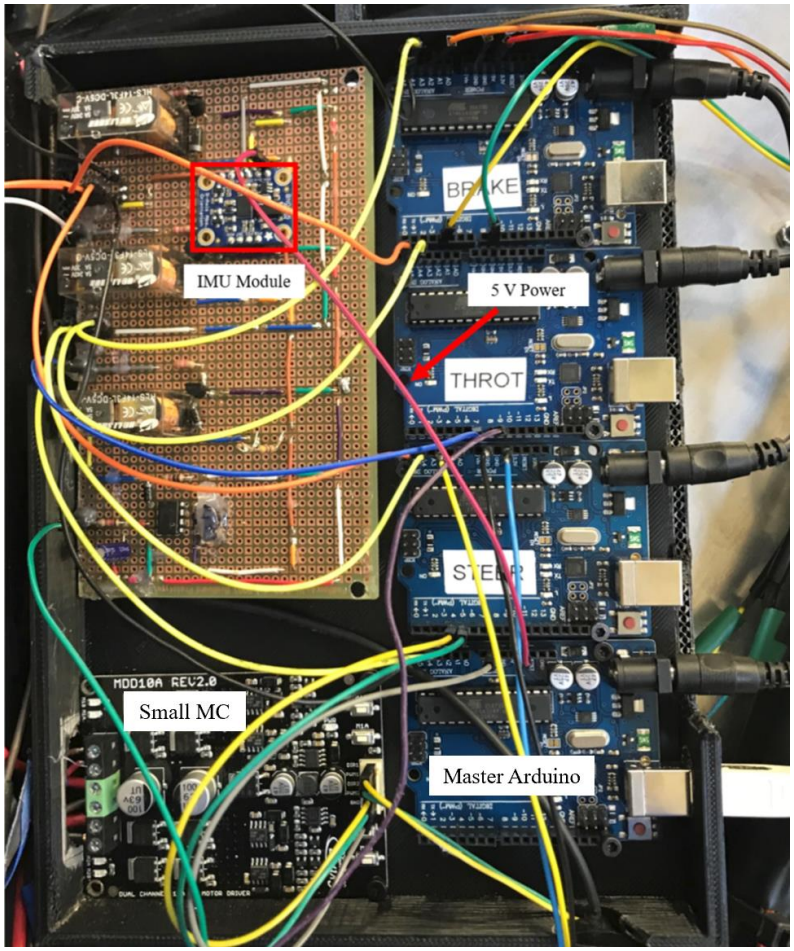


Figure 56. Vehicle Electronics Box Featuring IMU Module

The team still needed an IMU to help localize the vehicle and control its velocity. The BNO055 USB-Stick module [73] was chosen because it had the appropriate drivers necessary to interface with ROS. The breakout board module did not have the appropriate drivers necessary to interface with ROS. This IMU is connected directly to the MCU through UART connection and the information is extracted through a message decoder ROS node. This IMU is used in conjunction with the Adafruit Ultimate GPS Breakout.

Although the IMU module can provide relative position tracking with respect to the vehicle start position, it is still subject to drifting and deviations from the true position due to the accumulation of errors. To solve these problems, the Adafruit Ultimate GPS Breakout [54] is used for the global localization for the vehicle. This GPS module provides the vehicle with the current location with respect to map coordinates. The location of the final goal information is also set and provided with GPS coordinates. This GPS module can track up to 22 satellite fixes

simultaneously and is updated at 10Hz. The breakout board receiver is attached to a perf board and the antenna is mounted on the topside of the vehicle. With the help of the existing Adafruit Arduino library [74], the break board receive can automatically connect to available satellites and print out satellite fix data to the UART serial ports at 10 times a second in real-time.

Challenges arose when the team needed to incorporate the satellite messages into the ROS framework and transfer that into the local map coordinate frame to fuse with the IMU and visual odometry data. Luckily, the Adafruit GPS module outputs NMEA 0183. The team was able to use a universal NMEA message to ROS GPS message convertor to generate the proper location, covariance matrix, and frame ID needed for the ROS interface.

Despite the well-established embedded system library and standardized GPS message format, the team ran into problems with the low quality of the GPS receiver when fusing its data with other position data. This is described in detailed in Chapter 6. In the future, the team recommends using a higher quality GPS module with inner triangulation to combat the discrete GPS position jump problem.

Using Fusion 360 the team also designed and 3D printed a box with a cover to house the GPS and the IMU together, as shown in Figure 57. The box was then mounted to the vehicle as close to its center of gravity that the team could get, which was behind the driver seat. This can be seen in Figure 58. Where the antenna is placed on the vehicle is shown in Figure 59. Figure 60 shows where the 3D printed box and the GPS antenna are located in respect to the Baja vehicle.

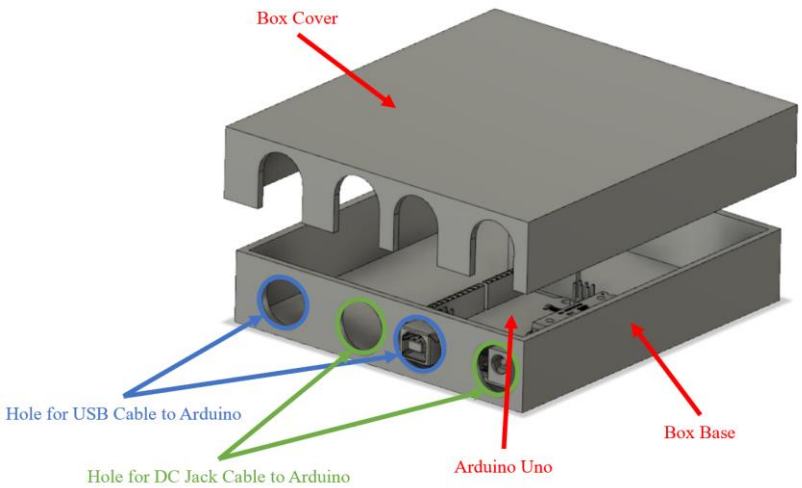


Figure 57. IMU/GPS Box Design

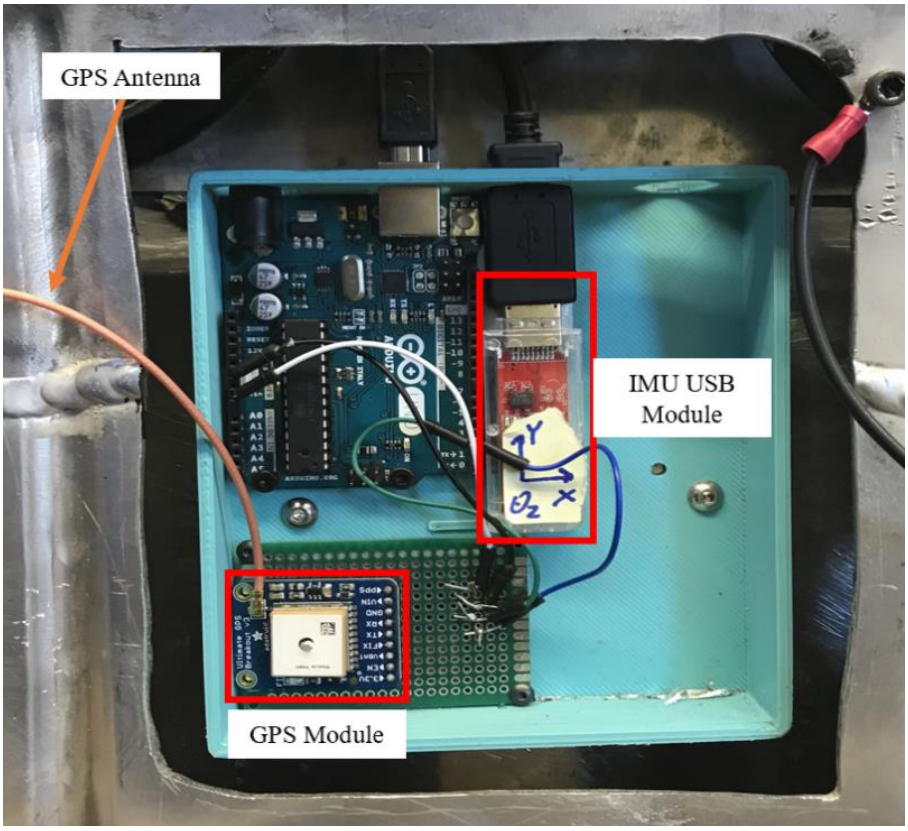


Figure 58. IMU/GPS Box Mounted to Vehicle

Figure 59. GPS Antenna Attached to Vehicle

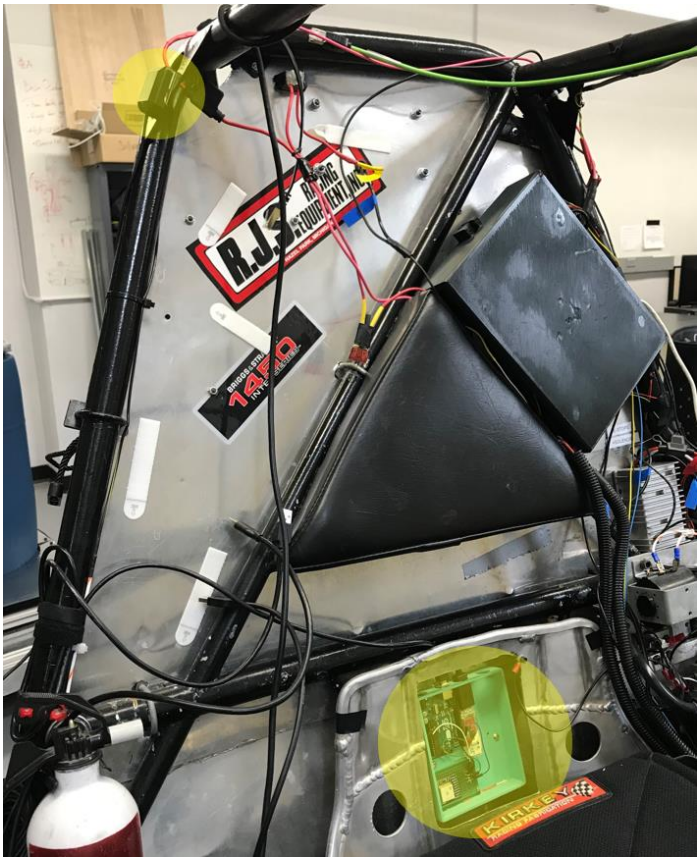


Figure 60. Baja Vehicle Highlighting Placement of GPS/IMU Box and GPS Antenna

In order to fully actuate the vehicle, all mechanical systems needed to be able to respond to electrical signals. Some systems required additional mechanical requirements, such as the steering and breaks, but the acceleration system could be fully actuated electronically.

The first aspect of this that the team worked on was electronically actuating the acceleration. After labeling every switch, the accelerator pedal was wired to an oscilloscope to investigate the electrical signal used to accelerate the vehicle. The team discovered when attempting to disassemble the pedal by removing its plastic enclosure, that the wheels would not spin because a magnet in the cover completed part of the circuit. By tapping into the potentiometer signal line, a relationship was created between voltage and acceleration. The signal coming from the pedal is a DC signal, ranging from 0.8 V to about 5 V, however the maximum speed of the wheels occurs at approximately 3.3 V. The microcontroller that the team decided to use does not have a digital to analog converter (DAC) on board, so the team developed a very simple DAC. The Pulse-Width Modulation (PWM) signal from the microcontroller was attenuated by an active low pass resistor-capacitor (RC) filter. This signal

was passed through an isolation amplifier (also termed a “unity gain buffer”) that isolates the impedance of the pedal and the filter from the impedance of the motor controller. By connecting this signal to the motor controller, the team confirmed that the drive-wheels of the vehicle could be controlled from computer commands.

One of the requirements for the vehicle was the ability to switch between driver operated control and autonomous control. It was recognized that some form of switch would be required to transition between human operation and autonomous driving. The team originally thought that an electrical switch should be used over a mechanical switch, as mechanical systems can degrade. The team hypothesized two types of switches: one that is purely MOSFET based, with the voltage coming from the throttle pedal or the voltage coming from the microcontroller chosen using a P-Channel MOSFET. The source that is not active gets a path to ground via an N-Channel MOSFET. The other idea was to switch using an AD8028 op amp with disable [75], and a MOSFET. When a user needs to use autonomous control, the disable pin on the op amp is enabled, not allowing current to flow to the gas pedal.

Components for both switch circuits were ordered and specked. However, neither set of components were successful in the desired operation. The circuit made from four MOSFETs did not have a high enough threshold voltage. When attempting to separate the electrical and mechanical signals, a lower MOSFET threshold voltage than anticipated decreased the resistance of the circuit and prevented it from operating ideally. The maximum voltage for the MOSFET was also too low for the application, causing crossover distortion between the signal. The second circuit with the AD8028 required that the difference on the input not be greater than 1.8 V. With a negative feedback connecting, the input signal to the Op-Amp and the output signal to the microcontroller had to be within 1.8 V of each other, which was not practical for the required application. Therefore, a third circuit was designed to allow for either autonomous or driver-based control. This circuit featured a diode which allowed current to flow in only one direction. This diode was used to separate the electrical signal from the mechanical signal in one direction, and the high impedance of an op amp to limit the current flow in the other direction. The combination of these two latter circuits should have allowed the team to control the acceleration of the vehicle from the output of the Arduino, while also allowing the output signal to be influenced by the mechanical pedal.

Figure 61 shows the combination of the circuits required to fully integrate the electrical and mechanical systems for acceleration, and to allow the switching between human operation and autonomous operation of the vehicle.

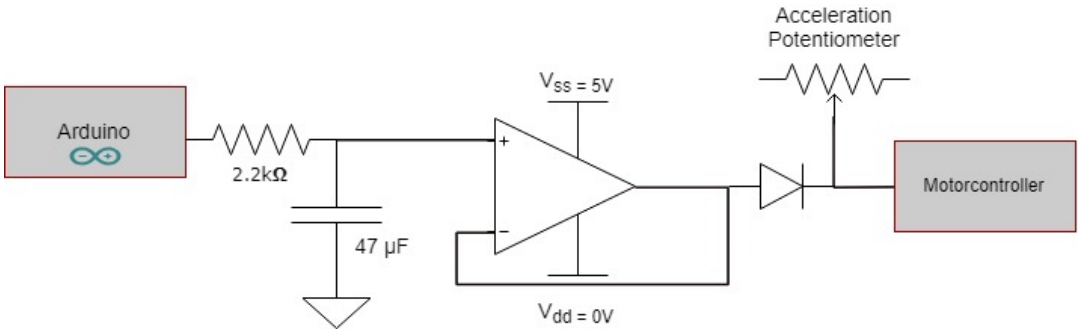


Figure 61. Active Low-pass Resistor-capacitor Filter with Diode

However, Figure 61 did provide some issues with regulating the acceleration circuitry of the vehicle. Any voltage left on the filter output or the potentiometer would have to be regulated and monitored, as the motor controller automatically disconnects if the input voltage drops below 0.7 V. After consideration, the team decided to use a Relay-HLS-14F3D-DC5V-C [76], which is effectively a switch controlled by an inductor. The electrical circuit of the relay is shown in Figure 62. The relay as it appears in real-life and in the circuit board is shown in Figure 63.

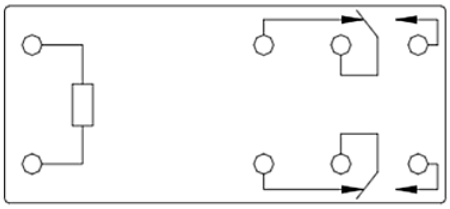


Figure 62. Relay Wiring Diagram Bottom View (Reproduced from [76])

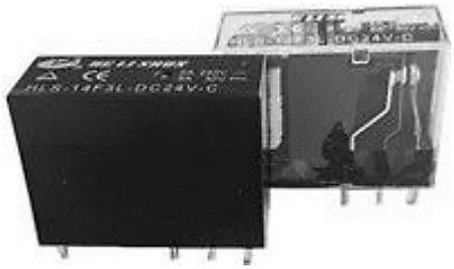


Figure 63. Switching Relay (Reproduced from [76])

Each relay has two switches, with three pins. While the coil has a net zero volts the pin closer to the coil and the middle pin are connected, but when at least five volts are sent across the

coil the pin farther from the coil and the middle pin are connected. The team connected the motor controller input to the middle pin, the pedal to the pin closer to the coil, and the pedal to the pin farther from the coil. This allowed the user to control the acceleration default through the pedal, and switch to autonomous acceleration using an Arduino. For the microcontroller to remain on from the autonomous signal, a default of 0.7 V after the diode was sent.

The team also decided the relay was useful to connect and disconnect the LiDAR from a 48 V to 12 V voltage converter and connected the LiDAR input voltage to a relay. This was intended to allow the LiDAR to be controlled using an Arduino, so it was not constantly receiving power from the voltage converter. This however was not successful due to the inability to solder thick wires to the small pins on the relay. The team decided to directly connect the LiDAR to the voltage converter and simply disconnect the power cable to the LiDAR when it was not in use.

To control the steering and braking system, the team used an off the shelf motor controller (Cytron MDD10A [77]) to send PWM signals at a 12 V scale to both the steering motor and the braking motor. As the motor controller operates off 12 V, another component was needed to utilize the 48-volt system of the vehicle.

The team used a DC voltage regulator to drop the vehicle voltage down from 48 V to 12 V; then the team feeds the output of this regulator to the motor controller. The motor controller is needed because the steering motor and the linear actuator (used for braking) operate off of 12 V. The two Arduinos that control the steering and braking motors both feed their respective signals into the motor controller, which amplifies this signal and puts it on its outputs and is then fed to the actuating motors. The two outputs are independent, so the steering and braking motors can be controlled independently of one another.

The actuators require full ability to be controlled from the MCU, but they also need to provide feedback to the system. The acceleration could be controlled fully electrically, so an external sensor is not needed, but an IMU is used for velocity control, nonetheless. However, the steering mechanism, discussed in detail in Section 4.2, requires additional external sensors for the MCU to know the positions of the actuators on the vehicle. The team decided to outfit the steering mechanism with potentiometers, which could then provide feedback to the MCU and the I^2C bus.

Each side of the steering rack was equipped with a 100 mm linear potentiometer, and each physical potentiometer came equipped with two potentiometers with three pins each. By equipping the mechanisms with four potentiometers total, the values could be averaged for repeatability. Any major difference in the readings of the values could also be examined in the case of broken sensors.

When wiring the sensors to the vehicle the full set up would require four sets of three wires each, power ground, and signal wires that would need to travel seven feet (2.1336 m) back to the MCU. Minimizing the electrical components in the front of the vehicle is desired, so the long wires were required. In order to minimize the wires needed a node was created for the power and ground of the potentiometers. The sensors were wired in parallel, so they all were able to receive five volts from the Steering Arduino. Current draw was not an issue to power the potentiometers as they only serve as voltage dividers. The four signal wires were connected through a pin header and plugged into the Analog inputs of the Steering Arduino. The power and ground wires were connected onto the perf board.

With the first iteration of the vehicle circuitry, the team had almost successfully made a remote-controlled vehicle. There were two issues: manual acceleration was not possible and turning on the Arduinos and the perf board was not safe. The issue with acceleration was that there was not enough power going to the relay on the perf board to switch the system from manual to automatic mode. Powering on and off the Arduinos and perf board was not safe because it required plugging in a live 12 V wire. In the event of an electronics fire, the driver of the vehicle would need to stick their hand into a fire to disconnect the wire.

To fix the manual acceleration problem, the team attempted to re-wire the perf board and Arduinos to take an input of 12 V as opposed to 5 V. The team also introduced a transistor into the circuit so the relay would be able to switch off of a lower voltage. The transistor had its base connected through a resistor to the input of an Arduino. Once the transistor base received more than 0.7 V, it connected its emitter and collector. The collector was connected to the coil on the relay which was then connected to 12 V, and the emitter was connected to ground. Therefore, an Arduino could be used to send a signal to the base and activate the transistor, thus sending a voltage drop across the relay. However, after the team soldered the transistor onto the old perf board, and tried to test the system, it did not seem to work. After adding more components to the perf board, the team was not confident in the functionality of the perf board and decided it would

be worth the time to recreate it. The team recreated the perf board circuit on a breadboard to test the design of the circuit before deciding that the issues were with the soldering of the perf board. The circuit on the breadboard worked successfully so the team decided to make a new and more organized perf board, as shown in Figure 64.

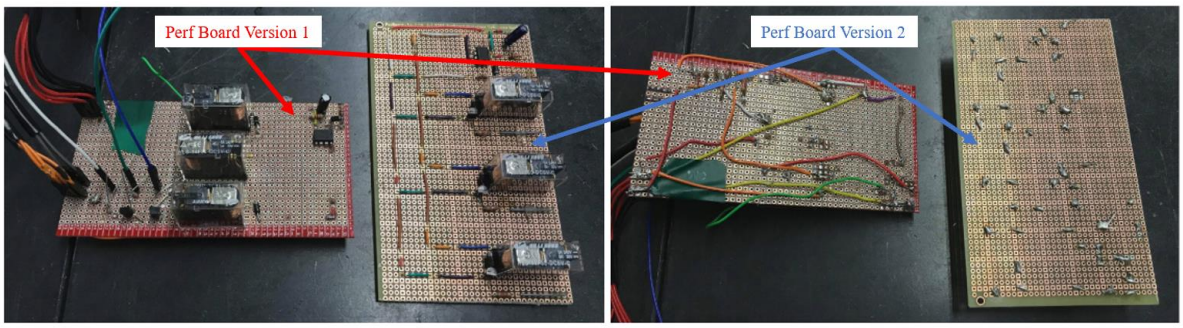


Figure 64. Version 1 vs. Version 2 of Perf Board

To fix the safety problem, the team decided to wire the entire system in parallel with the emergency stop switches. This would mean that pulling out the emergency stops would turn on the solenoid, the small motor controller, the perf board, and the Arduinos all at once. This would also mean that all subsystems could be powered by the same source. This circuit is represented in Figure 65.

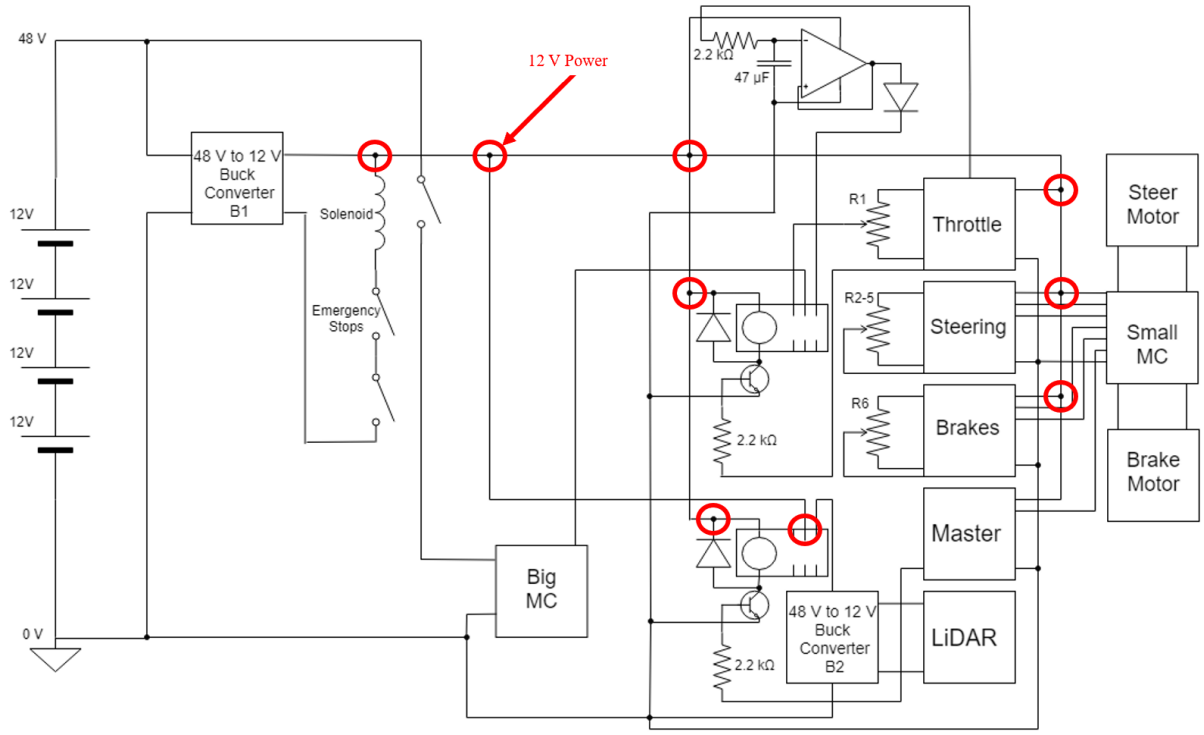


Figure 65. Vehicle Circuit with All Subsystems Wired to Emergency Stops

After implementing the circuit above, the vehicle began experiencing unintended acceleration. For weeks, the team could not figure out why this was occurring. The team first tested if the issue was occurring because the solenoid and the small motor controller were receiving power from the same source. After isolating their power sources using a power supply on a bench, the team determined that was not the issue. One night, the team decided to test a theory: lightly touching, but not pressing, the emergency stops. The team discovered that rubbing the emergency stops would cause the rear wheels of the vehicle to turn at full speed, for the duration of time for which the switches were touched. The team's hypothesis is that the human body making contact with the emergency stop was completing the circuit and sending excess voltage through the system.

The solution to this problem was to return to the previous circuitry, but with a different solution to the prior safety issue. The different solution was to add a physical switch on the output of the voltage converter that was powering the small motor controller, perf board and Arduinos. After making the changes and multiple tests, the team thought that this solution was successful, and a remote-controlled vehicle was created. The system can be represented by Figure 66.

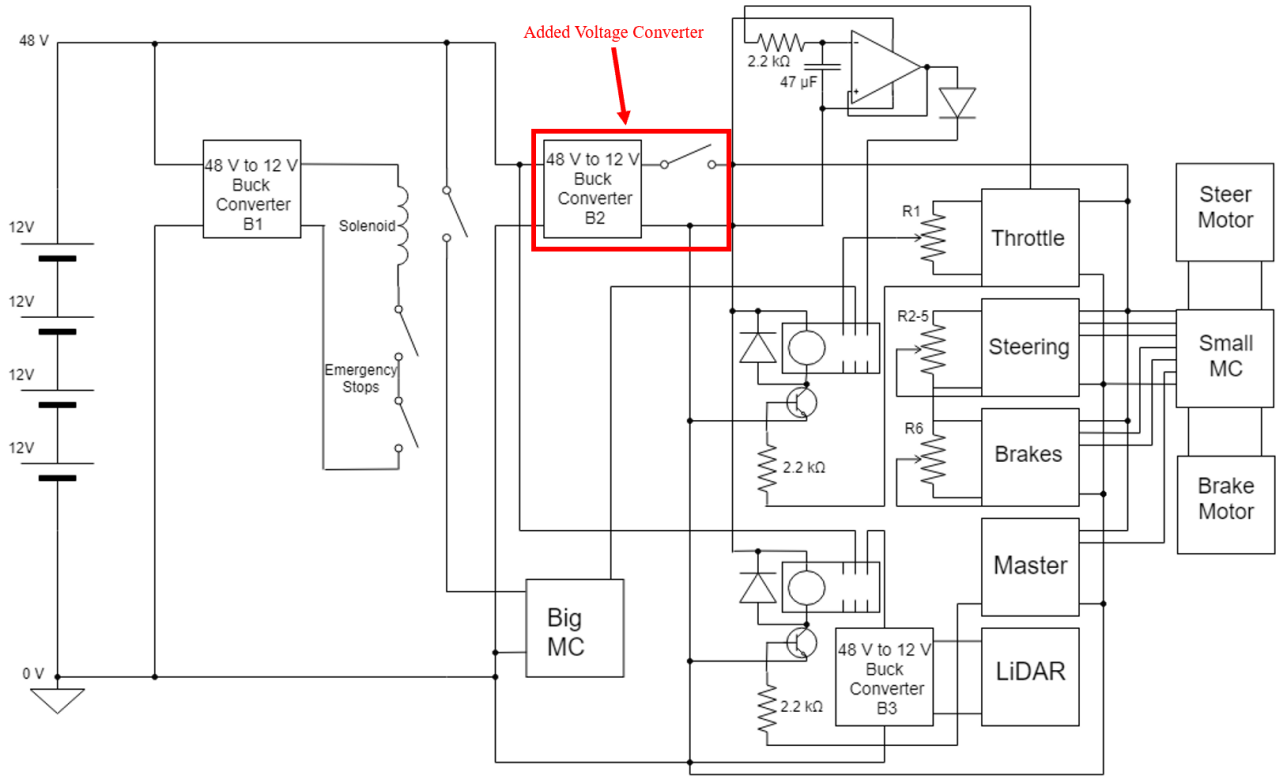


Figure 66. Vehicle Circuit with Actuation System Separated from Emergency Stops

However, this circuit implementation did not fix the unintended acceleration. After a few hours probing the entire vehicle with a multimeter, the team discovered that there was a 0.2 V potential between the ground of the four Arduinos, and the ground of the vehicle. The ground of the vehicle is the ground of the four batteries, which is connected directly to the big motor controller. To make matters worse, when the team made the vehicle steer or brake, either manually on the small motor controller or via MATLAB software, that 0.2 V potential spiked to around 0.3 V. The first attempt at fixing the issue was to directly connect the ground of the vehicle to one of the Arduinos. This lowered the voltage spike from a 0.1 V jump to a 0.06 V jump, so the wheels still accelerated unintentionally but not as quickly or aggressively.

After more probing, the team discovered that when the jumper male-to-male wires soldered to the perf board were moved, the wheels turned. For context, the perf board was grounded to the vehicle ground through a wire that the team soldered to a male-to-male jumper wire on the perf board. Then, four male-to-male jumper wires were soldered to that jumper wire, and those jumper wires were plugged into the ground pins on the Arduinos. Questioning the quality of the jumper wires, the team decided to reconfigure how the Arduinos were powered and grounded.

The team came up with the solution of powering and grounding the Arduinos using their DC power jacks and eliminating the 10 power and ground jumpers on the perf board altogether. The Arduinos were previously powered like they were grounded: via jumper male-to-male wires on the perf board. The Arduinos and perf board are now all powered and grounded directly from the second voltage converter, without the use of any jumper wires. Not only did this solution fix the unintended acceleration problem, but it also made the circuitry cleaner and safer to work with. (See Figure 67). The team was now able to control the acceleration, braking and steering separately of one another.

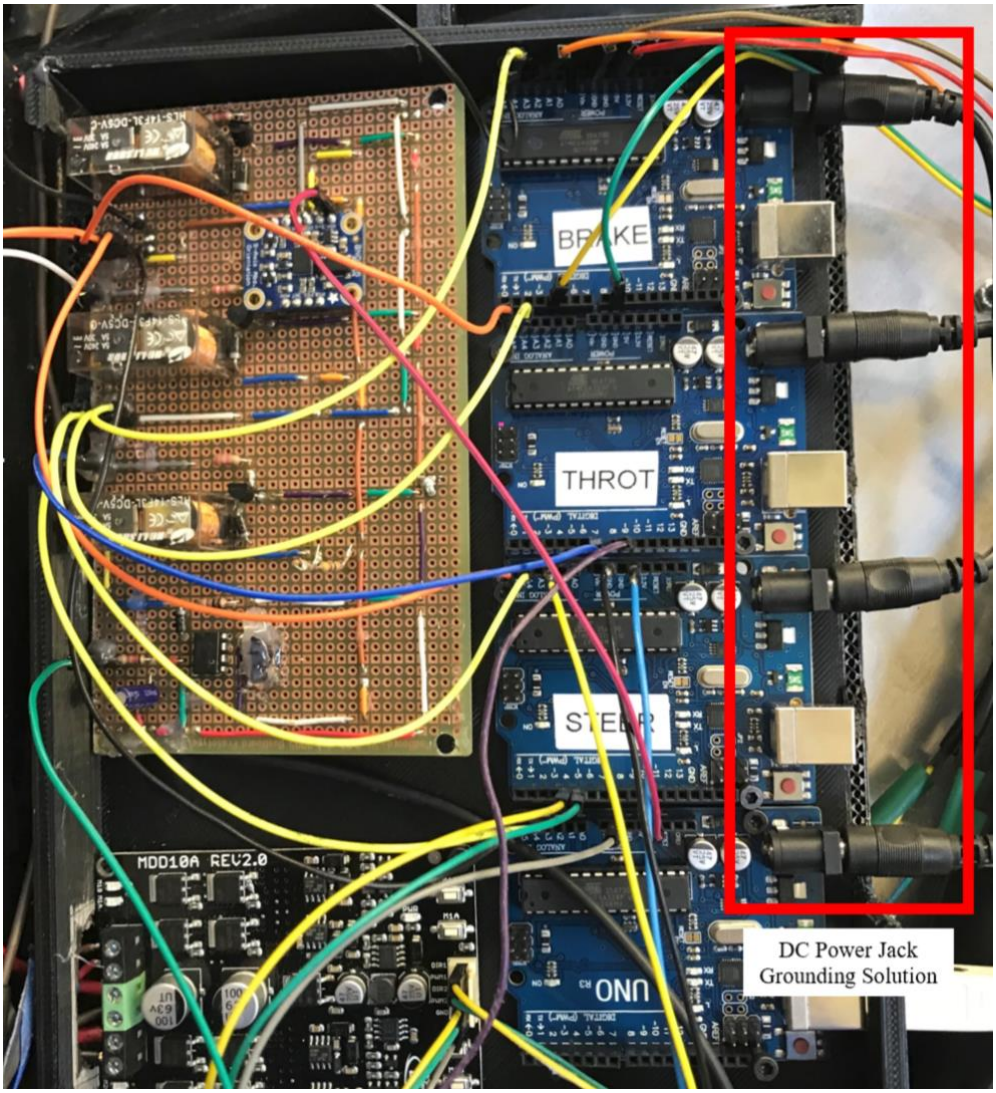


Figure 67. Arduinos Powered Through DC Jacks

The team ran into an unexpected problem: The vehicle could steer and brake at the same time, and it could accelerate, but it could not steer or brake while accelerating. The slave Arduinos were receiving the correct commands at the correct time, so it was not a software issue. After debugging, it was discovered that there was a periodic attenuating oscillation on every voltage wire, as well as on every signal wire of the vehicle. Research showed that all vehicles have what is known as jerk. “Electric Vehicles (EVs) develop high torque at low speeds, resulting in a high rate of acceleration. However, the rapid rise in torque of an electric motor creates undesired torsional oscillations, with vehicle jerk arising as a result” [78]. Figure 68 shows what typical jerk looks like for an EV.

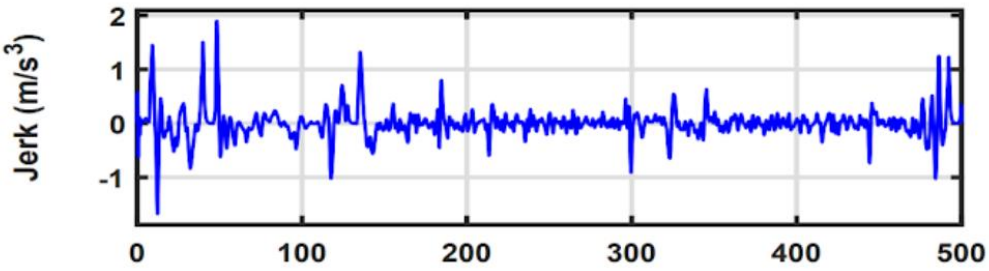


Figure 68. Jerk of an EV (Reproduced from [78])

When driving the vehicle manually, these oscillations do not cause a noticeable discomfort to the driver. The only reason they ended up being an issue for the team is because they were affecting the signal wires that allowed the team to control the various actuators. Figure 69 shows a zoomed-in view of the oscillations that the team saw on an oscilloscope. The red wave is from a voltage line on the vehicle, and the blue wave is from the signal wire connecting the throttle relay to the big motor controller.

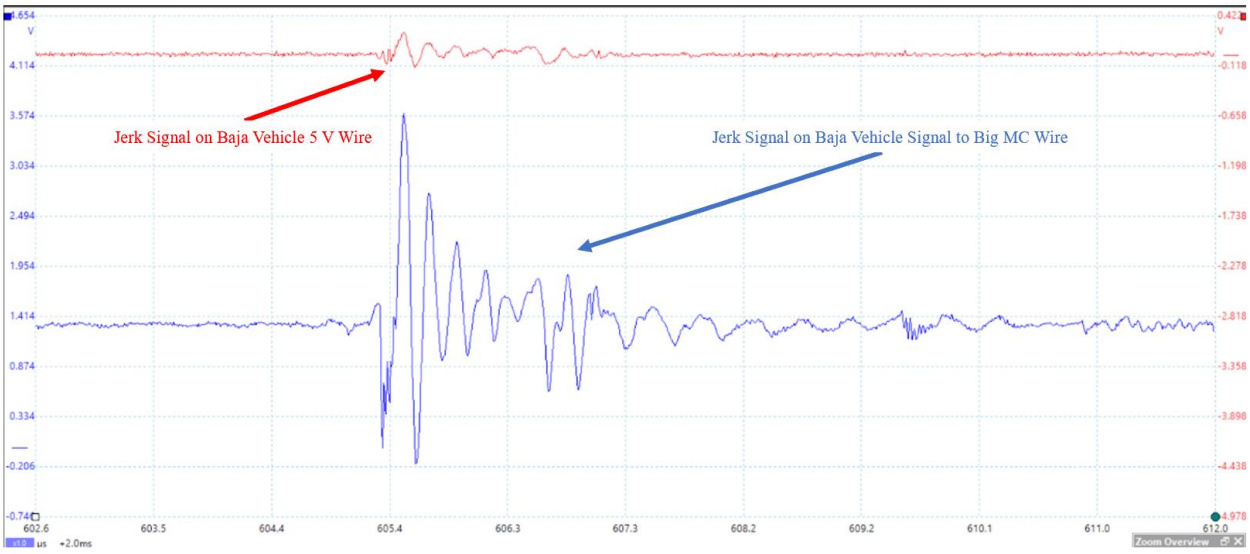


Figure 69. Vehicle Oscillations on Voltage Wires and Signal Wires

When the team tried to accelerate and steer at the same time, or accelerate and brake at the same time, the wheels turned but that was it. The reason for this is because when the vehicle jerked and created an oscillation, the oscillation appeared on all signal wires. This voltage spike on the signal wires made the steering and braking actuators think that they already reached the position they were instructed to go to.

The team's first attempt at resolving the issue was to place a low-pass filter on the signal wire going into the big motor controller. This helped by reducing the amplitude of the jerk signal on the wires, which then allowed the steering and brakes to move slightly. Unfortunately, this solution did not allow the team to move the steering and brakes nearly as far as the team required. This filter is shown in Figure 70.

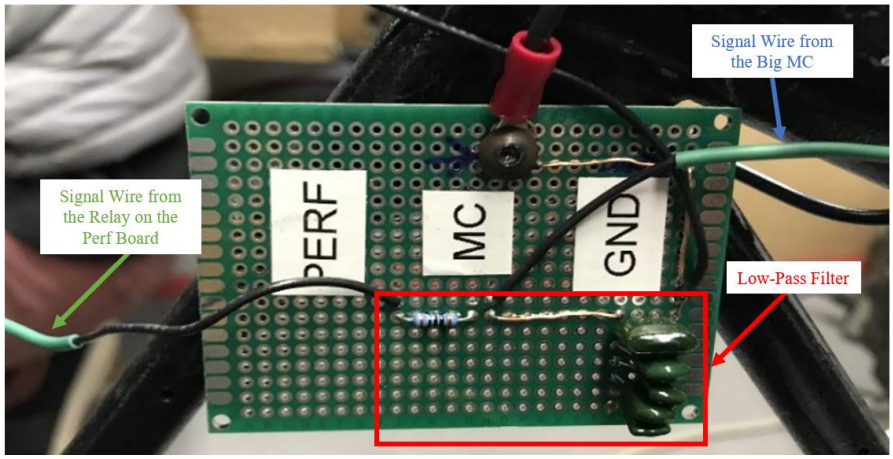


Figure 70. Low-Pass Filter for Signal Wire into Big Motor Controller

What the team needed to figure out, is how and why the oscillations were appearing on the signal wires. It was finally discovered that the vehicle chassis, metal chair and metal back plate, were not grounded and had the same periodic attenuating oscillations on it, with a peak-to-peak height of 800 mV. Grounding the entire vehicle solved the issues, and the team was able to successfully remote-control the vehicle. This can be seen in video here. In this video, the driver is in the vehicle for safety measure, but is not touching the brake pedal, throttle pedal, or the steering wheel. The embedded code on the slave Arduinos and the master Arduino that controls the various actuators is discussed in detail in Section 6.2.

5.3 Summary

This project proposed and used an Intel® RealSense™ Depth Camera, a SICK 3D LiDAR, a GPS module, and two consumer grade motion sensors for data collection. The SICK UC30 ultrasonic sensors were not able to be used by the team due to manufacturing issues. Microcontrollers are used for actuation, and microprocessors are used for data analysis. After various issues with the grounding of the vehicle were solved, the team had created a completely drive-by-wire vehicle with all necessary sensors for autonomy mounted to it. The entire system

can be seen in Figure 71. Future teams are **highly encouraged to check the grounds** if there are any strange problems with the electronics on the vehicle.

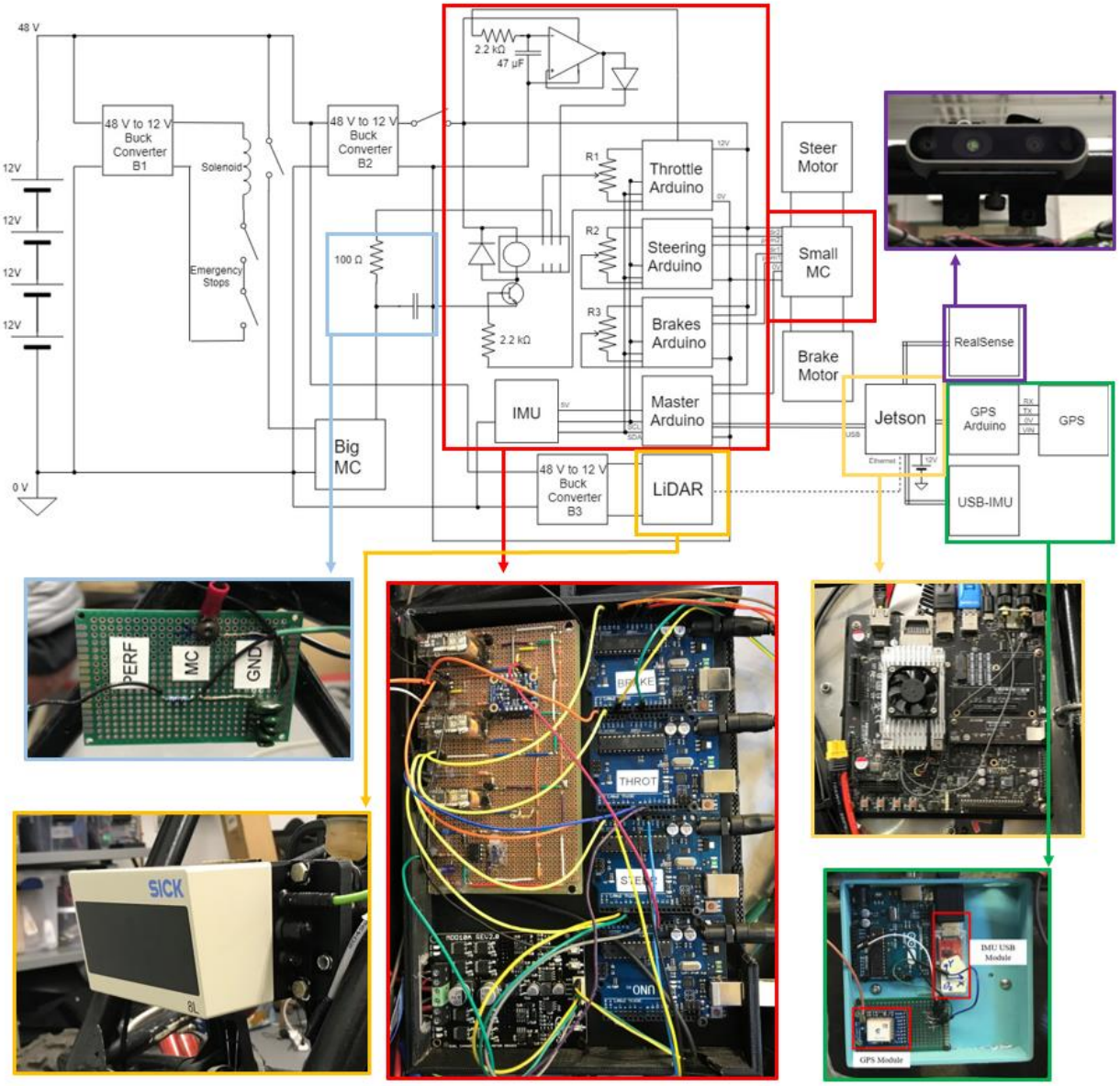


Figure 71. Final Full Vehicle Circuit

The information received by the sensors shown in the circuit diagram needs to be combined and normalized using various data fusion techniques. This fused data will be used to generate maps of the environment around the vehicle and then make path planning decisions. This software is discussed in Chapter 6 in more detail.

6: Software System Design

Fully integrating all necessary sensors and actuators for level three autonomy, which was discussed in Chapter 2, requires the design and implementation of a software architecture that efficiently acquires and interprets all system data to call upon the appropriate actuator. The software system needs to be able to acquire data from multiple mounted sensors and combine it together into a single 3D point cloud. The system needs to be able to use visual odometry and positioning sensors in order to accurately localize the vehicle in real time, and then use the location of the vehicle to project the occupancy grid, such that the vehicle knows where it can or cannot travel. The software system needs to be able to use the occupancy grid to generate a path and communicate the planned trajectory through commands so that the vehicle follows the path and reach a desired end goal.

6.1 Research and Proposed Approach

Hardware Actuation Control and Sensor Data Acquisition

The first area of software development that the team needed to research was the communications bus. “In computer architecture, a bus is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols” [79]. The team investigated three different options for the communications bus:

- A Controller Area Network (CAN) bus
- A Serial Peripheral Interface (SPI) bus
- A two-bus system, consisting of the Robot Operating System (ROS) as well as an Inter-Integrated Circuit (I^2C) bus

The first option was to use a CAN bus, which is a bus standard where all devices are connected to one another and can communicate without a master [80]. Choosing this option means that separate controllers are needed for each sensor, separate drivers are needed to be written to handle the sensor data, and additional parts need to be purchased and incorporated.

This additional work, as well as the high software expenditure caused by having to write original drivers, led the team to forego the use of a CAN bus.

The second option was to use a SPI bus, which is a common method for communication over short distances in embedded systems [81]. Unfortunately, SPI buses require more wires and pins than other bus methods and have no defined error checking protocol. Although SPI buses are much faster than other buses, the level of speed that SPI can provide is not of paramount importance for this project. These downsides led the team to forego the use of an SPI bus.

The third option was to use a two-bus system, consisting of ROS as well as an I^2C bus. An *I^2C* bus is a system that allows for serial communication between multiple slaves and a master. An I^2C bus only requires four different wires to operate between any number of slaves and a master: power, ground, data, and clock. *ROS* acts as a virtual CAN bus that allows the Jetson to handle package management, hardware abstraction, low-level device control as well as pass messages between processes, both internally through the Jetson or externally through wires. As the primary framework of data interpretation within the MCU, ROS uses packages, nodes, topics, messages and services as its data communication architecture. A *node* is an executable program that communicates with other nodes through messages. In this project, the messages consist of the data collected from the sensors as they travel via ports called topics. A node that requires information from a specific topic will become a subscriber to the topic. A *ROS package* is a combination of all related nodes into a software module that can be easily ported to other computing systems which run ROS. ROS has its unique data classes and formats as well as internal synchronization protocols. Thus, ROS can transfer multiple channels of frequent messages among different modules, even in different programming languages seamlessly.

The combination of these two systems requires fewer wires and can support multiple masters (if necessary), would incorporate ACK/NACK [82], as well as allow for the use of pre-built packages that support SLAM, occupancy grids, navigation and system control. Another plus is that the RealSense camera and the LiDAR have ROS integration packages, meaning that the team does not need to create any. One disadvantage is that this system is prone to electrical noise at long distances, but this will not be an issue since the hardware modules will be close to one another and do not need long wires. The other disadvantage is that ROS is not real-time, which essentially means that data cannot be processed quickly enough to provide immediate feedback. The lack of having a real-time operating system will not be an issue because the

project does not require exact real-time communication. This third option was chosen for this project.

Sensor Fusion

The next area of software development that the team needed to research was how the data from each sensor would be combined and interpreted. The team knew that this project would require *multi-sensor data fusion*, a process that allows obtaining comprehensive and robust data from multiple sensors and then combines their data together to make a cohesive and detailed world image. Sensor fusion is more reliable than data fusion, since multiple sensors are collecting data about the same objects through different methods [46]. However, due to the methods in which sensors collect and process information, the resulting data is likely to have a level of uncertainty that needs to be addressed before later use. As a result, accounting for such signal noise is one of the biggest challenges with sensor fusion and must be done before combining the data [83].

Data integration for this project would be completed through multiple microcontrollers and microprocessors, so that sensor fusion exists in a centralized network [46]. This form of sensor fusion is called *object refinement*, which is the ability to identify an object and track its current positions and predict its future positions [84]. This process includes modifying the input data to convert it to a uniform frame of reference and type [84], then continuously using filters to calculate the possibility that an object exists in the given position. This sensor fusion method is used to track the true location of the vehicle as well as to build a map of terrain and obstacles for the vehicle to navigate around.

Simultaneous Localization and Mapping (SLAM) is a method for interpreting data-fused sensor data and estimating geometric features in a global reference frame while using the same geometric features to estimate the position of the robot [85]. The team decided that this project would use this concept in conjunction with the GPS and IMU modules to correct the location "drifting" due to the imperfection of visual data. This process would assist the vehicle with traveling from point A to point B without colliding into real world objects.

For localizing the vehicle in the environment and planning the future movement of the vehicle, the team would implement a form of Bayes filter [86]. "A Bayes filter is an algorithm used in computer science for calculating the probabilities of multiple beliefs to allow a robot to

infer its position and orientation" [86]. The localization of the vehicle must incorporate many sources of sensory data including an IMU, GPS, LiDAR, and a camera. The team would implement a Bayes filter in the vehicle software system to create a statistical model that estimates the exact position of the vehicle. Two of the main type of Bayes filter for position and state estimations are the Kalman filter [87] and the particle filter [88]. The particle filter randomly distributes particles that represent the vehicles location over the entire map and picks the most likely locations of the vehicle by weighing particles. This is done by comparing all particles' theoretical movement against observed vehicle movement. However, the team determined that the Kalman filter is better for this project because it is significantly less computational heavy in comparison to the particle filter.

The Kalman filter consists of two steps - the prediction step and the correction step [87]. The statistical model made by the filter first predicts the location of the vehicle based on its knowledge of the previous location, state, the actuation of the vehicle, and the known dynamic model of the vehicle. The Kalman filter also considers the concept of covariance, a combined uncertainty of all vehicle states. A predicted covariance is also output, based on the previous covariance, the noise of sensors, and the actuation. This represents the uncertainty of the system with respect to the location of the vehicle [87]. Secondly, in the correction step, the algorithm combines all observations of the actual vehicle states and compares them to the prediction results and gives the corrected position and velocity of the vehicle. The filter is built on the Markov assumption that "the probability of the current location is dependent and only dependent on the previous observation" [89], and all observations and probabilities are based on *Gaussian probability*. Thus, in order to correct the prediction location, the current observation is combined with the prediction to form a new estimation of the location of the vehicle. Since the prediction and the current observation are both represented using Gaussian distribution, conveniently, the combined distribution will also become another Gaussian distribution. This new Gaussian distribution will have a mean that is in between the prediction and observation locations, and it will have a smaller variance. This new Gaussian distribution will have a better estimation. This is shown in Figure 72. In this image, the blue line represents the predicted locations and the orange line represents the observation locations. The Gaussian distribution of these two lines creates the gray line, which is a more accurate representation of the location of the vehicle.

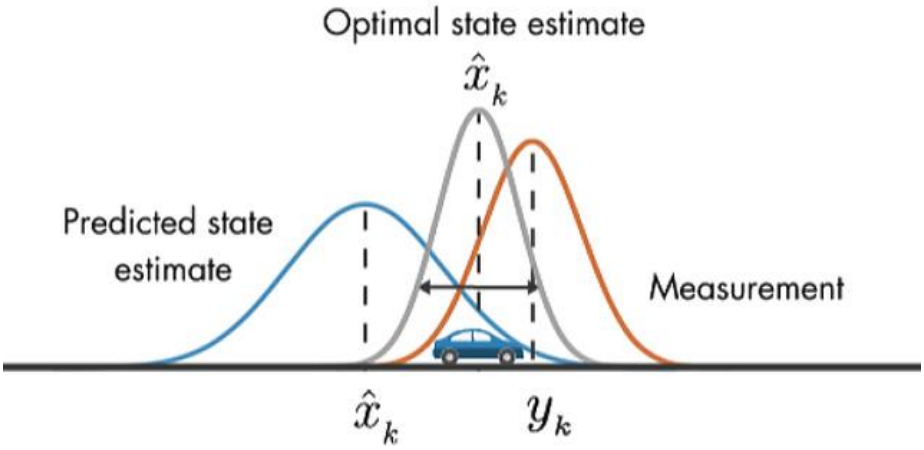


Figure 72. The Kalman Filter Gaussian Process (Reprinted from [89])

The correction step can be tuned with the variable Kalman gain K, shown in the equations below, to achieve better performance in different environments. With the increase of the Kalman gain, the system will take into more consideration on the current observation, making the location estimation more adaptive to changes while ignoring previous estimations.

Prediction Step Equations (Reproduced from [90]):

$$x_t = Fx'_{t-1} + Bu_t$$

$$\sum\nolimits_t = F \sum\nolimits'_{t-1} F^T + Q$$

Correction Step Equations (Reproduced from [90]):

$$K_t = \sum\nolimits_t H^T (H \sum\nolimits_t H^T + R)^{-1}$$

$$x'_t = x_t + K_t(z_t - Hx_t)$$

$$\sum\nolimits'_t = \sum\nolimits_t - K_T H \sum\nolimits_t$$

In this project, the Kalman filter will have to fuse many different types of sensor input to output the most accurate location estimation. First, the GPS module will be used to localize the starting and goal coordinates. However, the GPS localization has an uncertainty of up to 50 feet (15.2 m) radius. Thus, it cannot be a reliable location marker for continuous map building, as the vehicle requires accuracy within half of a meter to safely traverse the environment [91]. The other method to localize the vehicle is by use of an IMU module (a motion sensor). Although the IMU can provide precise vehicle odometry tracking from the measured acceleration data, it

suffers from signal drifting, causing the need for the localization data to be periodically corrected by other tracking methods, specifically SLAM. Second, the perception sensors such as the camera and LiDAR, can provide an efficient method to track relative positions of the vehicle and the environment by comparing and correlating certain image/3D features between frames (visual odometry).

Visual SLAM

During research, the team ultimately decided to use *visual SLAM*, a method that uses visual feature tracking to localize a cameras position and then uses these visual features to build a 3D map of the surrounding environment [92]. Visual SLAM can be accomplished using mono cameras, RGB-D cameras or stereo cameras. However, for most accurate tracking, cameras with 3D capabilities are recommended. The Intel RealSense D435 camera chosen for this project has dual camera set-up, as well as infrared radiation (IR) projection for more accurate 3D construction within low ambient IR environments. Thus, the team can compare two 3D construction methods and choose a more robust method for outdoor environments [93].

Visual SLAM consists of four steps: feature extraction, world point reconstruction, feature comparison and location solving [93]. Using stereo visual SLAM as an example, in the first step of the algorithm the program extracts robust visual features from both the left and right cameras. This is done for each frame of the video feed. These features are typically translation, rotation, and scale invariant image descriptors such as SIFT, SURF and ORB [94]. With two sets of relatively similar features extracted from both images, each of these features will have different image coordinates on the left and right image. The program can use the image coordinate differences to solve the reverse image projection function for the unique world points that coordinate with each of these feature points. The program repeats the first two steps for the next frame. If the vehicle/camera has moved, there will be two different sets of world points coordinates generated [93].

In the third step, the program must correlate these two sets of world points. Usually, methods such as Random Sampling Consensus (RANSAC) are used to randomly sample the next set of world coordinates against a set of randomly chosen points, usually 3 randomly chosen points, within the previous set of points [94]. If these random points can be transformed from the previous set and match to points within the new set, the entire previous set is matched to the next

set using the transformation matrix gained from the randomly chosen points. A score is then calculated based on a sum of the least squared error between two sets after transformation. This random selection process will be repeated for multiple iterations until a satisfying score is generated and then that reticular transformation matrix is recorded [93].

In the final step of visual SLAM, with the transformation matrix from the old-world coordinates to the new world coordinates known, the new camera orientation and position can be calculated using that matrix. Also, all the feature world coordinates in each frame can be fixed with respect to the world origin, thus stitching together a map of world points of the environment. Theoretically, transformation matrix can be generated from 3 non-coplanar world coordinates. However, due to the presence of non-static environmental objects such as other vehicles and humans, as well as the fact that some of the feature points extracted from the previous frame will fall off the edge in the next frame, thousands of image features are normally extracted from each frame for robustness [93].

Terrain Mapping

Autonomous vehicles require efficient and accurate terrain mapping in order to get from point A to point B. The team decided that this MQP project would use 3D probabilistic occupancy grids to create paths around obstacles, using raw data and a ROS package. These occupancy grids will be integrated with algorithms for path planning, enabling our vehicle to traverse a course safely around obstacles to a final destination.

Occupancy Grid Mapping is a method widely used within the robotics field and consists of a collection of algorithms that perceive real world scenes. The world map, or current sensing area, is represented by a grid of boxes, also known as an *occupancy grid*. The occupancy grid simplifies the world and helps solve the problem of navigating a robot or vehicle through a maze of obstacles. The grid boxes are populated either by binary values or by conditional probability values based on the chances that obstacles exist within the corresponding boxes. The size of the boxes reflects the obstacle information accuracy required and the processing power that the on-board computer can provide. A box is marked as occupied if the sensor and data fusion algorithm determines with a high probability that an obstacle(s) exists within that box. The vehicle itself is represented by a predefined volume of grids plus some amount of buffer for error margins. The error margins or “padding” provide tolerance for sensor uncertainties, localization uncertainties,

and additional room for when the vehicle is making a sharp turn. With both the vehicle and the environment represented in the occupancy grid, a path can be calculated while the vehicle is navigating through obstacles.

Occupancy grids maybe be either 2D or 3D. 2D occupancy grids function as a slice of the 3D environment; essentially, they are a single, flat plane. 3D occupancy grids consist of stacked 2D planes, representing a 3D environment [95]. 2D occupancy grids can be represented by either a binary occupancy grid or a probability occupancy grid. A binary occupancy grid uses `true` and `false` values that refer to the obstacles and free space respectively. Binary occupancy grids are commonly used if the project user does not have much memory to work with. Probability occupancy grids use probability values to create maps. This method involves giving each box in the grid a value, where values closer to zero represent free space, and values closer to one represent an obstacle. Since the MCU has enough memory, the team chose to use probability occupancy grids to create 3D occupancy grids. The probability that a box in the grid is occupied is a function of the number of point cloud detections within that cell, in addition to some Gaussian noise due to uncertainties of the sensor [95]. The occupancy grid is updated every time the sensor receives new data. Thus, if a grid cell truly contains obstacles, the probability number of that particular cell will increase upon the confirmation of receiving obstacle points consistently within multiple frames. This assists with passing obstacles such as animals and humans. Occupancy grids are widely used in many robotics navigation tasks in industry settings. These include factory automated guided vehicles (AGVs), drones, and service robots.

Path Planning

As this MQP is a project with search and navigation elements, the system would need an adequate algorithm to choose the optimal path for travel after terrain mapping is completed. Common search algorithms include:

- The Bug Algorithm
- Breadth First Search
- Depth First Search
- Greedy Best First Search (GBFS)
- Dijkstra’s Algorithm
- A*

One of most straightforward movement algorithms is the *Bug Algorithm* [96]. When operating under the Bug Algorithm, the robot will go directly towards its goal. Then if it encounters an obstacle, it follows it along one of its edges until it can refocus on the target. This approach is best applied when the time and distance to the goal are not important. It takes a long time to reach the goal destination with this approach therefore the bug algorithm is not acceptable for this project, which is designed to be used for efficient search and rescue missions. *GBFS* is a simpler algorithm that only looks at the future nodes (marked locations) that can be accessed in a map and chooses the node that is closest to the goal destination [97]. *Dijkstra's algorithm* is similar to GBFS, but instead of always choosing the best neighboring node, it explores all the possibilities based on a cost. Dijkstra's algorithm considers the cost to go from the root (starting location) to the node nearest the destination [98]. For example, instead of always going to the nearest city on a road trip, one first looks at a map, and figures out the shortest path to the destination city. Instead of distance, the heuristic could be time, or number of tolls, etc.

The team decided that this project would use the *A* algorithm*, which uses a heuristic to calculate the best path [99]. The most common heuristic used is the sum of the Euclidean distance and the Manhattan distance to the target [100]. The Euclidean distance is a straight-line distance between a start point and an end point, while the Manhattan distance is the distance that follows the road on a rectilinear grid between a start point and an end point. To reach the target faster, the heuristic can also include the time it takes to turn versus the time it takes to drive directly the final location in order to search for a most optimal path. A* is a derivative of Dijkstra's algorithm and adds a more developed heuristic, combining elements of Dijkstra's algorithm and GBFS. A* considers the shortest path to the target location, as well as the time required to complete neighboring paths. This results in the most optimized path to the target.

To implement A*, cells are assigned with values that represent travel costs, as discussed above. A Priority Queue sorts these values in the queue, largest to smallest. A dictionary for visited areas and an array for visited indexes in the occupancy grid keeps track of cells and paths that the algorithm has already searched. A start node with a value of `-1` would be appended to the dictionary. The *start nodes* index would be added to the visited indexes array, then the node would be added to the PriorityQueue. While the PriorityQueue is not empty, the lowest cost node is popped (removed and stored), and then its neighbors are searched if the goal has not yet been

reached. For each of the neighbors, the algorithm looks to see if the neighboring node has been visited and checked to see if the values in the dictionary are larger than the one being currently examined. If they are, the new node with updated parents and values are added to the priority queue. If the node was never visited, the dictionaries and arrays are populated with the node information.

One problem with traditional A* is that the algorithm searches through the four-connected or eight-connected neighbors of each grid cell and generates optimal paths consisting of only 90-degree and 45-degree angles. These paths cannot be followed by vehicles with Ackerman steering [101], because they cannot turn with respect to their center. Thus, *Hybrid A** is used instead. In Hybrid A*, the algorithm does not search through each adjacent node, and it instead searches the possible next positions by applying the kinematic model of the vehicle.

Due to the lack of the rear differential, the dynamic model of the Baja vehicle can be simplified to a bicycle model. In this model, a maximum steering angle (for example, 30 degrees to -30 degrees) and steering resolution (for example, 5 degrees) can be predetermined, given the velocity and the current location of the vehicle represented as a grid cell on the occupancy grid. Within a set time interval (for example, 1 second), the program can search through all possible steering angles and apply the bicycle model equation to calculate all possible vehicle locations and velocities after that time interval. By continuously searching the next possible location, a valid path can be generated for the vehicle to follow.

Trajectory Tracking

In order to determine the actuation commands needed for the microcontrollers, the path created by Hybrid A* must be transformed into a trajectory. This translates into each position along the path also having a desired time at which it is reached. The trajectory tracking controller is a feedback controller designed to ensure that the vehicle actually follows the planned trajectory. The trajectory controller needs to have access to the localization data, as this is how the controller verifies that the vehicle is following the trajectory. If the vehicle deviates from the desired trajectory, the controller will make an appropriate adjustment to the actuation commands sent to the microcontrollers. The gains of the trajectory tracking controller determine the transient performance of how the controller corrects for deviations from the trajectory.

Since the Baja vehicle has a limited top speed and an inability to quickly turn the steering wheel from lock to lock, each targeted position is set to be a fixed time-span apart. If the vehicle had a larger top speed or the ability to change its steering angle at a faster rate, then a variable time-span would be appropriate. This would be implemented using a function that increases the time span for curves along the path based on both how fast the instantaneous tangential heading is changing and how fast the vehicle is moving at the given position. Essentially, this is done so the vehicle does not need to be slowed down in order to make a turn.

It is possible to design simple and effective trajectory tracking controllers with no knowledge of the systems they operate on by experimentally tuning the gains that impact the performance of the controllers [102]. Some examples include proportional (P) controllers, proportional derivative (PD) controllers, and proportional integral derivative (PID) controllers. Trajectory tracking controllers with feedforward components are more complicated to integrate, although these controllers can have superior performance. Such controllers include both linear and non-linear feedback controllers where the feedforward component is based off of the kinematic or dynamic model of the system.

For this project, the team decided to select a PID controller, which is a simple feedback controller that is robust to uncertainty and does not require a model of the system. This controller was selected because an off-road vehicle will experience many disturbances that impact the system's ability to track the trajectory. Additionally, the controller is simple to tune in order to achieve the best performance on a given system. This controller was used to track trajectories at speeds of up to 33 mph in an off-road desert environment on a nonholonomic Ackerman drive vehicle [102].

6.2 Methods, Testing and Results

This section of the report will discuss the team's software development work, as well as the results of testing the software development.

The entirety of this project is programmed using the general-purpose programming language, C++. The language is known for being object-oriented, but also provides generic programming functionality. Figure 73 shows a high-level view of the software architecture designed for this project. The data from the sensors are combined, normalized, and corrected by SLAM and sensor fusion algorithms, and then interpreted to make motion decisions that are

carried out by electromechanical actuators. All sensor data is centrally corrected through software before being represented in a matrix structure. The decision-making algorithms use this structured representation to determine the path for the vehicle to follow. The control loop operates the electromechanical actuators to guide the vehicle along its trajectory.

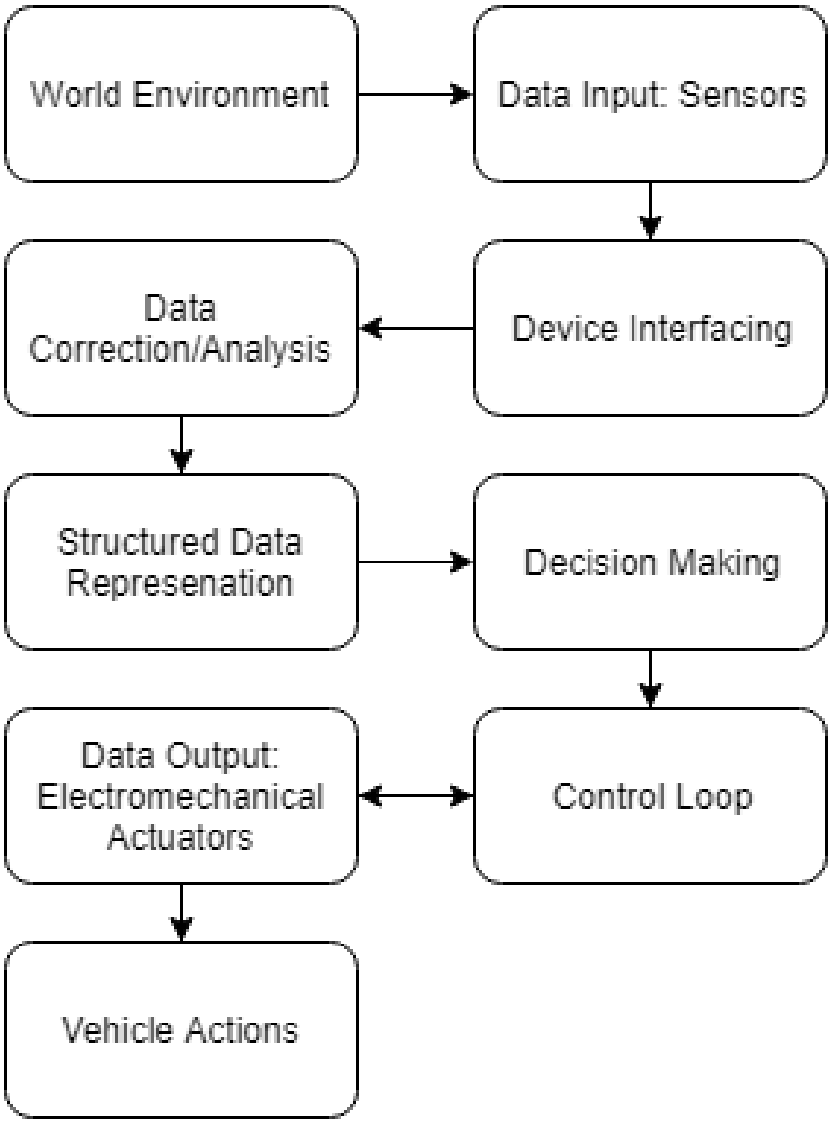


Figure 73. System Flowchart: Data Processing to Vehicle Actuation

Figure 74 represents how the data flows as the vehicle searches if an obstacle is in its path. The various exteroceptive sensors, first introduced in Section 4.1, take information from the environment and process it in the device interfacing portion of the code. The information is then processed in the data correction and analysis portion. After being processed, the data is then combined into an accurate map, from which the vehicle makes decisions on path planning and

movement. The map itself is stored in a data array where each cell represents a point in the map. If the map cell value is `-1`, then it is an unexplored area. Otherwise, the cell will hold a value between `0 - 100`, representing the likelihood of an obstacle existing at that location. As the map updates via the data input from the sensors, the path of the vehicle towards its final goal is updated through the path finding algorithm, Hybrid A*. The vehicle makes its decisions on where it should travel next. The vehicle then communicates this path to the actuators accordingly and travels via a controlled path that avoids obstacles.

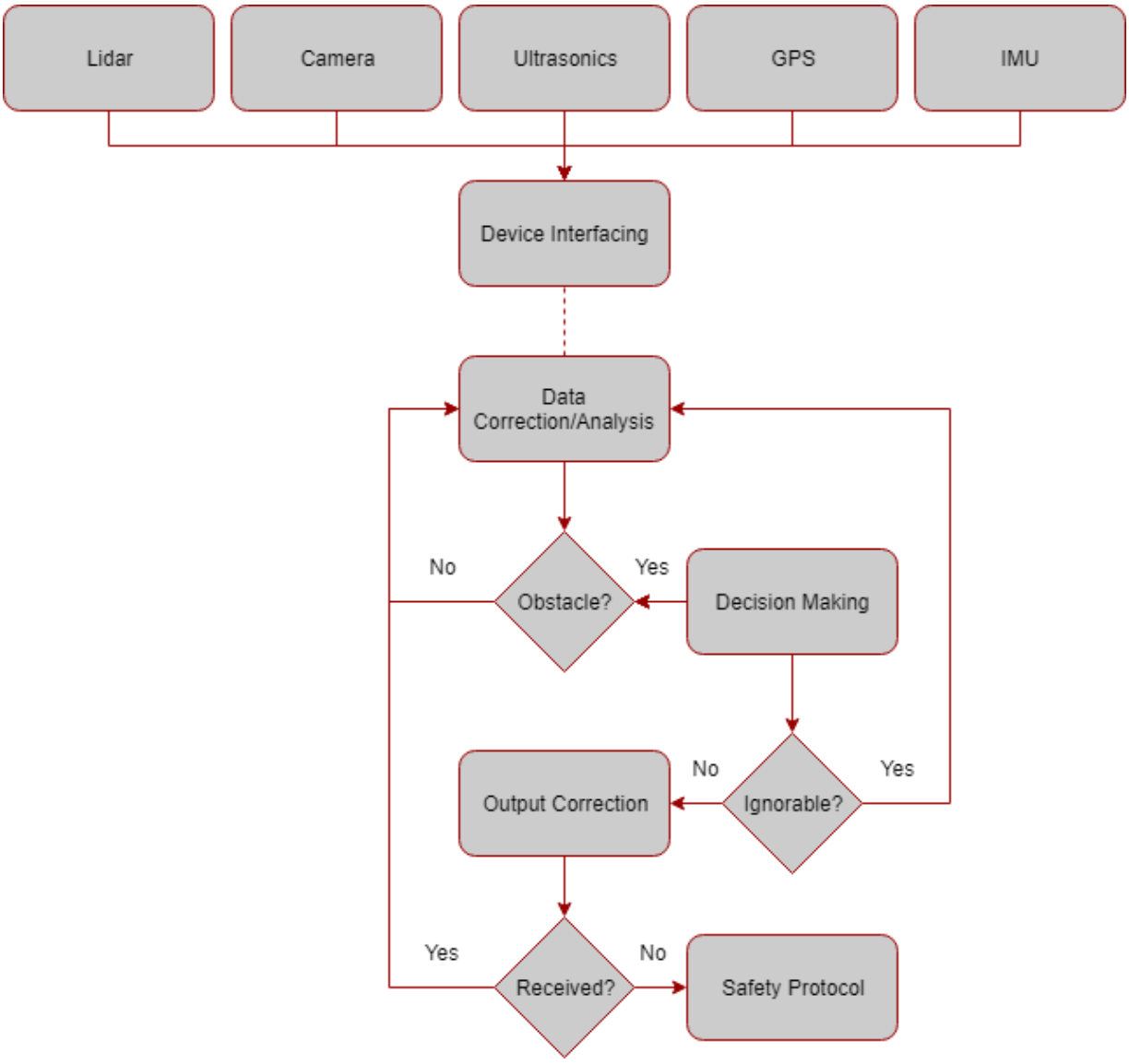


Figure 74. Data Fusion and Decision-Making Flowchart

Within the programming structure, a handful of classes and functions represent the vehicle system (actuation, sensors, and computing systems) and the vehicle communication system. They represent how the vehicle software system processes data, represents data, and distributes commands. These functions are housed on different processors based on their use case, mainly between the MCU and the actuation devices. This structure is shown in Figure 75. Many of the algorithms and sensor drivers used in this project were adapted from previous open-source versions that can be found online. The team wrote control functions used to accelerate, brake, and steer the vehicle.

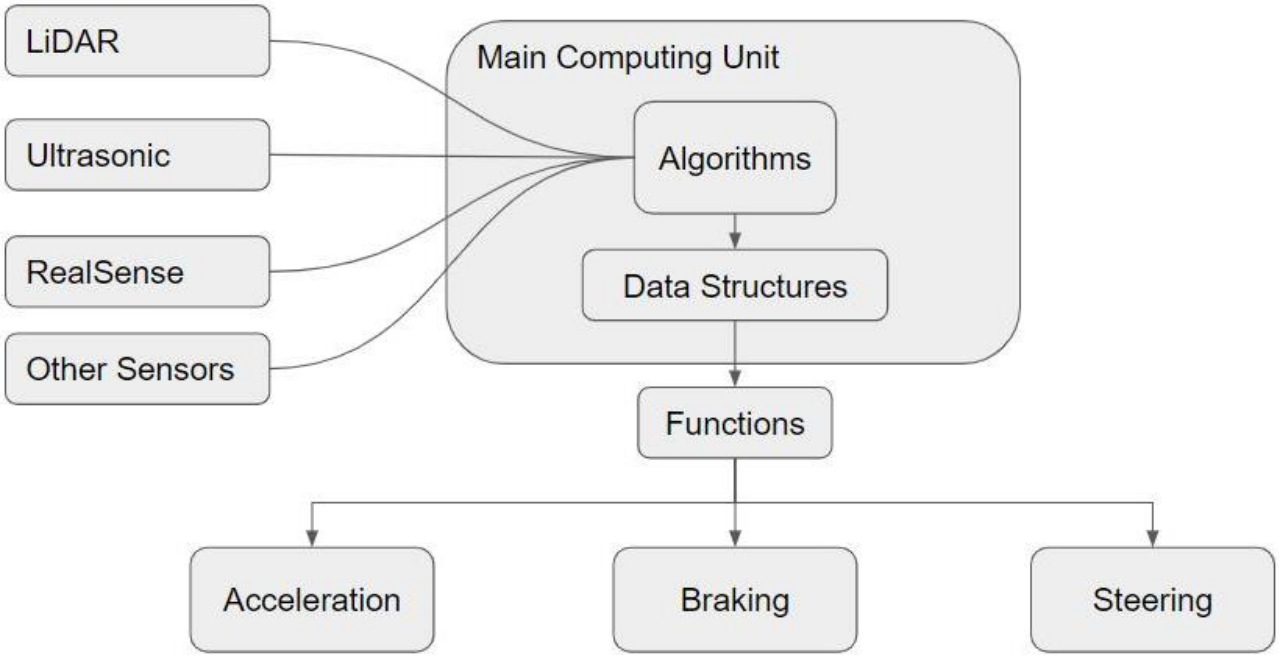


Figure 75. Programming Structure

The Main Computing Unit, as shown in Figure 75, is where the bulk of the data processing occurs. It is also where most of the commands are sent and the decisions are made. The MCU class design is as follows:

Table 16. MCU Class Design

MCU (Main Computing Unit)
Contains an Occupancy Grid Object Contains a Vehicle Snapshot Object

Key Functions
• accelerate (int speed)
  o sends a drive message to slaves to accelerate or deaccelerate to a certain speed
• brake (bool engage, int percentage)
  o sends engage or disengage message, as well as percentage of brake engagement
• turn (int angle)
  o sends message detailing steering amount
• currentObstacles()
  o reads occupancy grid object
  o returns obstacles currently nearest to the vehicle and their respective quadrant locations relative to the vehicle
• stop()
  o Emergency stop for edge case detection
  o calls break function to full disengage and calls drive function to speed 0
• currentState()
  o Attains the current position of the vehicle within the generated map relative to world frame of reference
  o Attains physical state of car from Vehicle Snapshot object
• readMessage()
  o reads message from slave
  o updates Vehicle Snapshot object

Two classes are housed within the MCU to receive data from the sensors and then process the data into usable formats: The Occupancy Grid Class and the Vehicle Snapshot Class. These are shown in Table 17.

Table 17. Occupancy Grid Class and Vehicle Snapshot Class

Occupancy Grid	**Vehicle Snapshot**
Class holds an overall map of current surroundings and explored surroundings. Directly subscribes to all related ROS Topics. Contains: Lidar Data Matrix Realsense Data Matrix Ultrasonic Data Matrix Contains closest obstacles to car All known obstacles locations relative to vehicle frame of reference Algorithm: Hybrid A* • Procedurally generates a map of traveled space of vehicle as well as frontiers where the vehicle can and cannot go • Produces path planning commands for the vehicle to be able to drive to appropriate locations.	Current physical state of the car. Contains: Orientation of Vehicle State of Acceleration State of Steering Actuator State of Brake Actuator Current location of car

Slave devices are designed to control their individual element of the cars control. In this system there are three specific sub-systems: acceleration, braking and steering. Each slave will have the same overall class structure but will be altered to fulfill its specific tasks. The slave class design will occur as follows:

Table 18. Slave Class Design

Slave

Key Variables
int message
• Stores message received from MCU in bytes
Key Functions
• receiveEvent()
  ○ Receives event from MCU class
  ○ Stores event in message variable
• processEvent()
  ○ Processes message from MCU and executes command based off of taskf of slave
  ▪ Acceleration - applies acceleration of value received
  ▪ Brake - Brake engage/disengage, and percentage
  ▪ Steering - Turn right/left certain amount.
• errorMessage()
  ○ Encountered some sort of error, sends to jetson message containing error data.
• returnState()
  ○ Sends a return message to the Jetson about the current state of that particular element. Acts as feedback loop for closed feedback.
  ▪ Acceleration - current Acceleration
  ▪ Brake - is brake on or off (how much is brake actuated)
  ▪ Steering - current steering position

In conjunction with the two main classes (MCU and Slave), the system includes multiple pre-built ROS packages that support algorithmic calculations. These packages are housed on the MCU itself but are also standalone executables that communicate with ROS via ROS Topics.

Table 19. Pre-built Packages

SLAM ROS package	**GPS**

From Algorithm, pre-built package
These functions are generated and executed for localization algorithm
• ROS_Publisher(PCL::pointCloud world_map,double[] current_location_velocity)
  ○ Publish the SLAM determined location, velocity through ROS, as well as stitched point cloud map
	From Algorithm, pre-built package
	These functions are generated and executed for localization algorithm
	• ROS_Publisher(double[] current_location)
	  ○ Publish the determined location through ROS

IMU	**Kalman Filter**

From Algorithm, pre-built package
These functions are generated and executed for localization algorithm
• ROS_Publisher(double[] current_location_velocity)
  ○ Publish the determined location, velocity through ROS
	From Algorithm
	These functions are generated and executed for the localization algorithm
	• ROS_subscriber(ROS::location)
	  ○ Receive location data from ROS
	• prediction(double[] current_position, double[] current_velocity, double[] car_kinematic_model)
	  ○ Predict the vehicles next location after time t
	• Correction(double[] IMU, double[] SLAM, double[] GPS, double[] predicted_position)
	  ○ Correct the predicted vehicle location after receiving sensor input

Occupancy Grid	

From Algorithm
These functions are generated and executed for the localization algorithm
• ROS_subscriber(ROS::location)
  ○ Receive location data from ROS
• prediction(double[] current_position, double[] current_velocity, double[] car_dynamic_model)
  ○ Predict the vehicles next location after time t
	• Correction(double[] IMU, double[] SLAM, double[] GPS, double[] predicted_position)
	  ○ Correct the predicted vehicle location after receiving sensor input

The first software system designed in this project is the control system, used to create a drive-by-wire vehicle. The control system uses two main communication protocols to actuate the brakes, steering, and throttle. First, the software running on the MCU receives and interprets data from the sensor array in order to determine which actuator to command. For example, if the

vehicle is traveling down a hill then the IMU will send data to the Jetson indicating its angle; if the throttle is not being pressed, the MCU will actuate the brake. Second, once the data is interpreted, the MCU sends a message over serial communication to the master Arduino microcontroller, which interprets this command and instructs the slaves to tell the corresponding actuator(s) to move. This command consists of the slave address corresponding to a particular actuator and the data relevant to the actuator.

The system is represented by Figure 76, which is separated by serial communication, I^2C protocol, and slave devices. The central master Arduino contains embedded software that controls the flow of information. It then sends that information to one of three slave Arduinos controlling acceleration, steering, and braking.

The MCU sends information to the master Arduino after processing and analyzing the data it receives. The sensor array communicates with the MCU using different ROS packages for data correction and analysis. Data is received from the RealSense, GPS, IMU, and LiDAR, and it is then implemented for sensor fusion and SLAM. The data received is filtered and fused through software on the Jetson and motion commands are sent to the master Arduino microcontroller to actuate the vehicle accordingly.

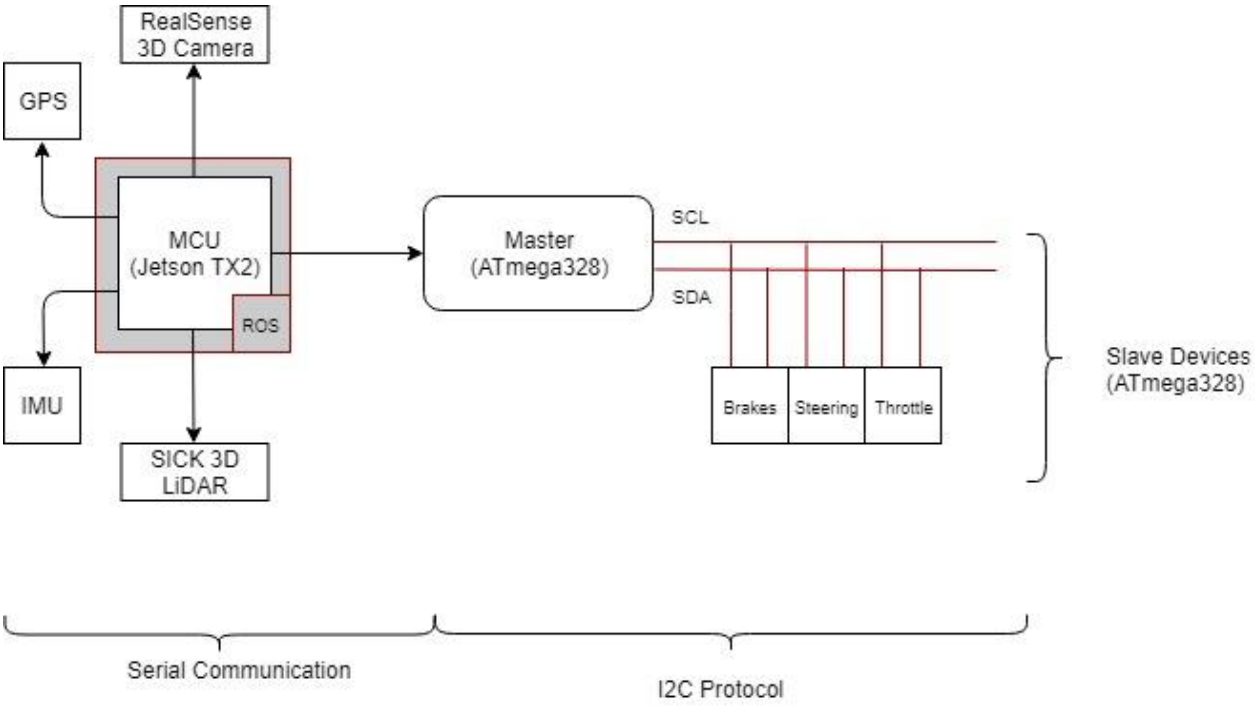


Figure 76. Communication between Sensors and Actuators

The development of the communication network began when the team began working on the actuation process, discussed in Section 4.2. In short, one master microcontroller gives orders to its slaves. The slave microcontrollers are each responsible for the tasks of braking, steering, and accelerating. The slaves and the master communicate to each other over I^2C (Inter-Integrated Circuit). The command that is sent from the master microcontroller includes the SlaveID and the data. The SlaveID allows communication to a particular slave, and the data is dependent on the actuation system, specifically percent extension for braking, speed for acceleration, and position for steering.

Initially, each Arduino was programmed with individual scripts using *Arduino integrated development environment (IDE),* which is a cross-platform application written in Java. To test the systems, the team needed to recompile the master program every time. While debugging the drive-by-wire controls, the team decided that instead of needing to recompile the program to send a new set of commands to the vehicle, the vehicle could be fed commands over UART using a MATLAB program in the exact format that they are received from the MCU. As an example, the serial port connected to the master sent commands in the form of "8 50" (in the case of the braking system), where 8 is the SlaveID and 50 is the data. The team initially only tested the serial communication with the braking system on the linear actuator. Once the linear actuator could move instantaneously after a command was sent to it, the team implemented the same system with both the acceleration and steering systems.

The design of the system is simple: the master takes in an incoming command 10 times per second and parses it into a SlaveID and data, before sending it to the corresponding slave. The MATLAB program, discussed above, was able accelerate and decelerate the vehicle using the "w" and "s" keys, as well as turn left and right using the "a" and "d" keys. The "q" and "e" buttons were set up as "emergency stops." This allowed the team to quickly control the vehicle from a computer to test the I^2C network in depth. See Appendix E: Master-Slave Arduino Communication, for the master-slave code. See Appendix F: Acceleration-Slave Arduino Code (slaveAccel.ino) for the acceleration slave code, Appendix G: Braking-Slave Arduino Code (slaveLinAct.ino) for the braking slave code, and Appendix H: Steering-Slave Arduino Code (slaveSteering.ino) for the steering slave code.

It is imperative that the embedded control system has the ability to switch directions or stop the vehicle quickly because the MCU is sending the control system new commands ten

times per second. The code had to be written in such a way that the instance that a slave command is changed, the vehicle acts accordingly. An example would be that the vehicle is initially instructed to turn to the right and is then told to switch directions and turn to the left. During initial testing, the team executed most of the slave data parsing and execution in an Interrupt Service Routine (ISR). Several embedded platforms allow for interrupt prioritization, allowing one interrupt to pre-empt another. This method did not allow for new commands to be interpreted immediately; new commands were only being interpreted once the previous ISR had finished. To remedy this issue, the team modified the ISR to only set certain flags, and the controller that manipulates the actuator runs at a much lower priority in the main loop. When a new command is sent and received by the slave, the flag is set that a new command has been received and the position to navigate to is updated.

Although the slave responsible for the throttle effectively changes the speed of the vehicle, there needed to be a way to account for certain times when the vehicle speed needs to remain constant, such as driving on an incline. The IMU located on the perf board of the vehicle, shown in Section 5.2, is used to help compensate for speed lost or gained on slopes. When the module detects a slope, it adds (or subtracts) to the voltage that is being sent to the motor controller of the vehicle.

This worked in individual testing using the MATLAB script in the laboratory but integrating it with ROS when testing outside caused issues. The vehicle would either refuse to listen to the IMU data or would otherwise accelerate to speeds faster than intended. At the time of writing this report, the infrastructure exists to implement this feature, but it has not been fully integrated.

Visual SLAM has a high level of complexity, consisting of CPU and GPU parallel processing, complex mathematical systems, and probability filtering for uncertainty removal. Due to this complexity, the team incorporated pre-built visual SLAM packages within the sensor fusion programming. The team first experimented with a package titled ORB_SLAM2 with the RealSense camera to properly implement accurate localization and mapping. The ORB_SLAM2 package, originating from Cornell University [71], provides a solution to frame and point cloud stitching and relative odometry calculation. This method uses stereo vision to extract useful feature points within each frame of the video stream. These feature points are determined by calculating image gradient and edge descriptors. About 400 of these feature points are

continuously tracked across each frame to calculate the orientation and translation changes of the camera. This ORB_SLAM2 package takes in the RealSense camera calibration file and the ROS topic name that is published by the camera. The package continuously returns a current location and orientation of the camera. The current camera location and orientation are represented by six data points consisting of X, Y, and Z translation distances and three Euler rotation angles with respect to a starting location. The package also returns a stitched point cloud map (defined in Section 5.1), which consists of all the consistent feature points extracted within every frame. The location, orientation, and point cloud map are outputted and published to their individual ROS topics at a rate of 10 Hz. However, after extensive simulated testing with indoor live navigation and outdoor recorded navigation data such as the KITTI dataset [103], the team drew the conclusion that although navigation through ORB features produced actuate tracking within the environment, the 3D features it produced were not enough to form a 2D map for the vehicle to traverse.

Instead, the team used the RTABMAP package for visual SLAM, a platform for RGB-D SLAM and stereo SLAM with ROS integration [104]. Similar to traditional visual SLAM approaches, the package first extracts image features within each frame and then matches the image features with their respective depth gained from the LiDAR and the 3D camera. However, unlike traditional SLAM packages, RTABMAP has the ability to project point clouds onto a ground plane to form a 2D occupancy grid map. The team provided the 3D camera calibration data, camera and LiDAR frame transformation, LiDAR point clouds, camera color image and depth image stream ROS topics, to the RTABMAP API. The package only provides a rough framework for visual SLAM, while the inner workings of the visual odometry, ROS environment interface, and map generation, have to be set for the specific specifications of the vehicle and test environments through environment files (.yaml) and launch files (.launch), which the team generated. The team was able to tune the performance of visual SLAM through testing and data collection.

While visual SLAM can provide accurate tracking and localization using camera and LiDAR data, the team has found that based on preliminary testing that SLAM can lose track of the location of the vehicle under specific circumstances. Examples of such circumstances are when the vehicle is making a tight turn around a close corner, or facing a larger planar surface without any features, such as a wall or a flat parking lot ground. Thus, the team combined sensor

fusion and visual SLAM for more accurate vehicle tracking and map generation. The team researched the possibility of implementing the Kalman filter from scratch. However, because the vehicle will be traveling with non-constant acceleration and the sensors on the vehicle do not produce consistent readings and are prone to sudden jumps, the system cannot be mapped linearly. This would make a manual implementation very complex and time intensive. The Kalman filter is based on the multiplications of linear gaussian functions, meaning it is unsuitable for the vehicle to use [105]. The team used the pre-built Robot Localization package [106], which implemented an Extended Kalman Filter (EKF) [107] and used Taylor Series expansion [108] to linearize the system. This package, which is similar to the RTABMAP package, provides a framework that the team can incorporate into the system. With this framework, the team wrote scripts to take the published odometry topic from visual SLAM and the IMU and the GPS and fused them into the robot localization package.

To seamlessly fuse sensor data together, the team had to synchronize the sensors. The sensors are placed on various places of the vehicle and produce their own data with respect to their unique sensor origin. To fuse the data together, the respective origins of the sensors need to align properly. The team calibrated and synchronized the sensors by first choosing the vehicles center of gravity, and then measuring the length and orientation displacement of each sensor with respect to that point. The team accomplished this using accurate laser range finders. With these measurements, the team was able to broadcast transformation matrices consisting of sensors' XYZ translations and Euler angle rotations from the the chosen point into the ROS system so that all measurements from the sensors are aligned with the center of gravity of the vehicle. These scripts are stored under the localization package in the MCU and are members of the ROS native localization package.

After fusing the previous IMU and visual odometry data, the ROS localization node outputs the fused and corrected relative position, orientation, and velocity tracking with respect to the starting point (ROS/odom frame). However, the team encountered difficulties when trying to fuse the GPS data within the same EKF localization filter. With thorough inspection, the team discovered that although the GPS receiver provides position data at a constant rate, the positions it provides consist of discrete jumps. This means that the GPS data would be stuck in the same position for a period of 2 to 3 seconds, and would then jump to the next position, despite the fact that the vehicle was moving at a constant velocity the whole time. This kind of discrete jump

created problems when trying to fuse continuous data, such as the IMU and the visual odometry data. Although merging the GPS data posed a challenge, the team did not want to give up on this source of localization because only an absolute positioning device could provide position corrections after errors from the relative position sensors accumulated over time.

After research, the team discovered a way to merge the GPS with the rest of the system. The team implemented a second instance of the EKF filter running in parallel with the first EKF filter. The first EKF filter takes data from the IMU and visual odometry fusion, and the second EKF filter takes in the GPS data and the output from the first EKF filter instance. This second EKF instance outputs the final fused data within the ROS/map frame. The second EKF filter is running at a much slower rate and essentially is updating the predicted position whenever the GPS data is updated. This is done while not affecting the continuous, rapid fusing of the first EKF filter. The output of the completely fused localization data is then fed into the RTABMAP visual SLAM package as the foundation of the visual odometry estimation of the next frame. See the software system flow diagram, Figure 79, for more detailed information of this process.

For the team to properly use this map data created by SLAM, a 3D occupancy grid had to be constructed in order to simplify the point cloud data and determine a clear path for the vehicle. The package OctoMap is used to construct occupancy grids efficiently [109]. It takes in the combined map data from the SLAM package and represents the world with 3D grid cells. It also uses a technique called variable grid cell to maximize performance [109]. The world starts from a giant single cube cell, so if any cell has one or more obstacles, the cell is divided to represent more intricate details. The dividing process continues until all features are represented or the pre-defined resolution is reached. This method speeds up the process because it can reduce cells needed by representing a large area of space or obstacles with one or few large cells, instead of high-resolution cells. The occupancy grid created is published to the ROS occupancy grid topic and will be further processed to analyze available frontiers and paths.

Once the outside data is collected and represented by an occupancy grid, the software uses Hybrid A* to plan the vehicles trajectory. The team used a package created by karlkurzer to handle Hybrid A* path planning [110]. Unfortunately, this package did not solve the issue of generating a path towards a goal that is outside of the given map. While the Hybrid A* package can generate paths between two points, these points have to be known points on a known map.

In order to generate a path towards an unknown point, the team created a separate ROS node to act as an intermediary solution to this problem. The node itself utilizes standard A* to find the closest map location to the final goal. The node takes in a goal and starting point coordinate from the vehicle and searches between them to find the nearest frontier to traverse towards. A *frontier cell* on the map is defined as a cell that represents the boundary between what is known and unknown [111]. Once a goal position and a starting position is received by the node, the A* algorithm can then find the frontier cell that is nearest to the goal. Then it chooses that frontier cell as a temporary goal for the vehicle to travel to next. The resulting path to the selected frontier cell is generated using the Hybrid A* path planning package which considers the motion of the vehicle in terms of a bicycle model (discussed in Section 6.1). It is possible that an edge to the map is discovered before the frontier. In this case, the cell of the edge that has the lowest heuristic to the final goal is chosen as the new temporary goal. The orientation of the vehicle at the temporary goal is defined as facing the final goal and is sent as a part of the outgoing message containing the temporary goal. This goal is updated every time the vehicle updates its current position and sends it through ROS as its new start position.

Once Hybrid A* generated a path, the software system needed to define commands to instruct the vehicle to follow that planned path. The team modified a trajectory tracking package, devised by Sam White and Davis Catherman [112], as the framework for trajectory tracking. The team modified the package to simulate a desired path by using parametric equations that define a desired x and y position as well as a desired tangential heading. For the non-simulation portion, the team modified the package to observe localization data published from the localization package rather than the VICON state estimation system used with the original package. Additionally, the package was modified to observe the desired path that should be published by the Hybrid A* package and convert that path into a trajectory. The package was further modified to publish appropriate actuation commands to the Baja vehicle microcontrollers. The team tested and tuned the controller using the simulation component to stand in for the localization and path finding components while they were under development.

Full Software System Integration

Several critical layers of the software system were developed independently but needed to be integrated together. The sensing and localization layer needs to be able to communicate its

findings to the path planning layer such that Hybrid A* has both a map and a beginning pose to work from. The sensing and localization layer also needs to share the pose of the vehicle with the trajectory tracking layer as a feedback mechanism in the control loop. The path planning layer needs to communicate the desired path with the trajectory tracking layer as an input to the control loop. The trajectory tracking layer, having both the current localized pose of the vehicle and the next several desired poses, needs to send the output of the control loop to the microcontrollers to instruct vehicle actuation. The microcontrollers need to receive these actuation instructions and then command the vehicle actuators appropriately.

These communication needs are met by ROS, or Robot Operating System [70], which enables different pieces of software to communicate through a library of defined message protocols. For example, ROS enables the trajectory tracking layer written in Python to talk with the sensing and localization layer as well as the path planning layer, both written in C++. This same trajectory tracking layer is then able to communicate to the microcontrollers, whose embedded software is written in Arduino C (a C/C++ hybrid computer language).

The software workflow started with various sensors that communicate to the MCU through wired UART connections and their respective ROS message types such as Pointcloud2, IMU, Image, CameraInfo, and GPS Fix. The SLAM package takes the Image, CameraInfo, and Pointcloud2 ROS data to produce the preliminary ROS odometry data, stitched Pointcloud, and Occupancy Grid map. The localization stack takes in preliminary visual odometry data, IMU, and GPS fix, and outputs the corrected odometry for the SLAM package. The path planning stack then takes in the ROS corrected odometry data and the Occupancy Grid map and generates an array of poses in ROS Path format for the trajectory control stack.

The trajectory tracking software layer of the system receives messages from the sensing and localization layer. This message, an odometry navigation message, contains a time stamp, current pose, and twist. The twist consists of the current linear and angular velocities. The time, current position, and heading are extracted from this message and fed into the control loop, while the path planning software layer publishes the desired path as an array of poses. Due to these poses not being time-stamped by the path planning software, the trajectory tracking controller attaches a desired time for each pose such that the controller knows exactly at what time the vehicle should try to reach each desired pose. The time assigned to each pose is 1/10 of a second because the system operates at 10 Hz. This allows for smoother trajectories because there are

more waypoints on the path to interpolate between. The desired linear velocity and acceleration, as well as the desired heading, angular velocity of desired heading, and angular acceleration of desired heading, are all extracted from the trajectory that is created from the desired path.

For most of this project cycle, the team used the aforementioned MATLAB script to communicate with the different actuators on the vehicle over UART. However, this was always meant to be a temporary step before transitioning the system to ROS such that the embedded control system could interface with the decision-making and trajectory-generation software. To have the embedded control system talk to ROS, minute changes were needed in the embedded software.

The team knew the embedded control system would still be communicating with the Jetson over UART. To integrate the embedded system with ROS, a library known as "rosserial" was included on the Master microcontroller [113]. The rosserial library allows ROS messages to be sent to the serial port, where the microcontrollers can listen to them. Once the Master was set up as a subscriber to a string message type on the topic "motion_command," the system was ready to receive commands from the trajectory tracking software that lived on the Jetson.

Although the Master microcontroller was set up to listen on the motion_command topic of type "string", the messages that the trajectory tracking dealt with primarily were of type "Twist". A "Twist" message is made up of two parts: linear velocity and angular velocity. The "Twist" message was used to test the trajectory software on a platform known as the "TurtleBot" in Gazebo, a physics simulator [114]. For the TurtleBot to move, the software must send it messages of type "Twist" to have it move at instructed linear and angular velocities. For the vehicle to listen to the same software, the message type "Twist" must be converted into a string data type, then reformatted into the SlaveID and data format mentioned previously in this chapter. For example, the Twist message that might tell the TurtleBot to move at 3 m/s, must be converted into a string message that looks like: "8 3.0" where 8 is the SlaveID for the throttle, and 3 is the vehicle velocity. Translating the linear velocity portion of the Twist message into a string that the vehicle could understand was straightforward, as one only needs to cast this to a string and publish it. Linear velocity behaves similarly, whether it is for the TurtleBot or the vehicle.

However, both the steering and the brakes expect a position rather than a velocity, which is what the Twist message puts out. To obtain a position for the steering, the angular velocity is taken from the Twist message and integrated over the period of the message.

Since the team is sending messages at 10 Hz, the period of the message is 100 milliseconds (ms). Thus, the team knows where the steering should be after 100 ms, using the equation:

$$d_f = d_i + v_t$$

Above, d_f is the final destination, d_i is the initial destination, and v_t is the velocity. However, if the desired position was not yet reached and the position needs to be altered, the embedded software allows for the position to be changed at any time, which was mentioned in Section 6.2. A similar solution was implemented for the brakes, which would actuate if the vehicle is commanded to go to a negative velocity.

After the mathematical integration used to calculate the position of the steering is completed, the command still needs to be converted to the correct format for the embedded software to understand. The steering slave Arduino expects a byte data type (an 8-bit signed two's complement integer) in the range of 0 to 255. For the command to be in the correct format for the embedded software to understand, the 0 (full lock left) to 90 (full lock right) values of the steering motor are translated into values between 0 and 255, such that the slave can interpret them. Once the slave receives these values, it multiplies them by a scaling factor and adds an offset. The offset is needed as full lock left has a potentiometer value that is non-zero. Without the offset, the steering would attempt to go to position zero, and would likely overdraw current from the system or break a mechanical component due to the high torque of the window motor (the motor used for the steering system as discussed in Chapter 4).

Once the embedded control system was able to talk with ROS, the team worked on giving the TurtleBot a path to traverse. The Twist messages that the TurtleBot listens to are translated into messages for the vehicle, and both the TurtleBot and the vehicle perform the same path; the TurtleBot in simulation and the vehicle on a dolly in the lab.

The vehicle was taken out to the Gateway Garage to test localization capabilities and Hybrid A* path planning. A section of the garage was closed off in order to test, and obstacles were set up using cones, cardboard boxes, and team member's personal vehicles. The vehicle was then manually driven around these obstacles in a dedicated path as shown in Figure 77.

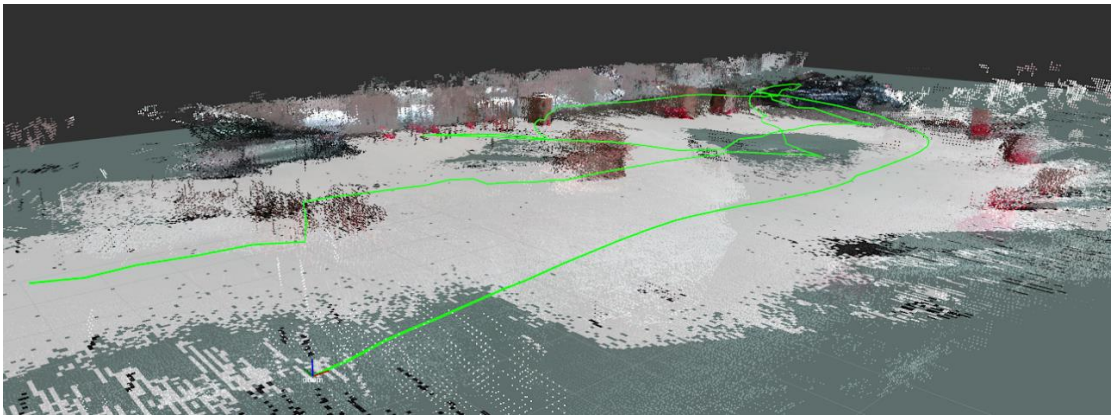


Figure 77. Manually Driven Baja Vehicle Path Taken at Gateway Garage on 4/15

As the vehicle was being driven, it was able to create a point cloud of data using the LiDAR and RealSense. This point cloud represents the space in which the vehicle was driving. This point cloud is then converted into a map that displays the empty space and the occupied space as seen by the sensors on the vehicle. Frontiers, the boundaries between known and unknown territory, are also displayed on this map as areas that have not been searched yet. The accuracy and validity of the map is measured by visually checking how closely it represents the real-world setup and how clearly each of the obstacles show up on the generated map. The final map is then passed to the Hybrid A* package. In the Hybrid A* package, a start node and a goal node are defined. Both of these nodes are passed into the algorithm in order to generate a vehicle path as seen in Figure 78. The success of the Hybrid A* package is based on if a path can be generated that does not pass through obstacles or unknown terrain.

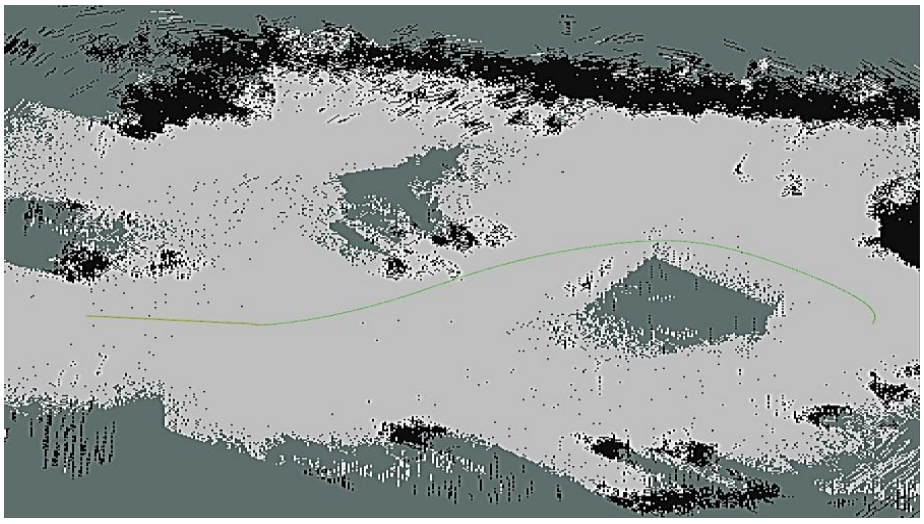


Figure 78. Hybrid A* Generated Baja Vehicle Path (Green Line)

In the event that a goal is positioned in an unknown region of the map, or outside the map constraints, the frontierFinder node is then used to find the frontier that the vehicle should traverse towards. The success of this node is measured by if it is able to find an accurate frontier node to travel and if that frontier is closest to the set goal point. This would be then sent to the Hybrid A* package as a new goal node to path plan towards. To see if the vehicle would be able to intelligently follow that path in the laboratory, the team substituted Gazebo for localization data and allowed the vehicle to follow the generated path while on a dolly. This allowed the team to work out most of the bugs prior to testing this project in an outdoor environment. The entire software communication system designed and implemented for this MQP is shown in Figure 79.

6.3 Summary

Fully integrating all necessary sensors and actuators for level three autonomy, discussed in Chapter Two, required the design and implementation of a software architecture that efficiently acquires and interprets all system data, and then calls upon the appropriate actuator. Based on the results of the data acquired, the MCU generates an optimal path for the vehicle to move from point A to point B. The MCU actuates the vehicle by commanding a master of an I^2C bus with a message containing a SlaveID followed by data. The data is specific to each actuator. Once the master node receives this message from the MCU, it commands the appropriate slave to move the corresponding actuator.

This project uses Simultaneous Localization and Autonomous Mapping (SLAM), in conjunction with a global positioning system (GPS) and an Inertial Measurement Unit (IMU), to have the vehicle travel from point A to point B without colliding into real world objects. SLAM serves as the primary source of collision avoidance. Hybrid A* was implemented as the primary path finding method, as it is able to generate a path that is consistent with the movement constraints of a vehicle that moves with a bicycle model, where its center of rotation is outside of the vehicle itself. An A* search algorithm was also implemented in order to allow the vehicle to generate intermediary goal coordinates in order to traverse towards the final goal coordinate. Robot Operating System (ROS) provides libraries, tools and basic operating system services. ROS is used as the primary framework of data interpretation and communication between subsystems connected to the MCU.

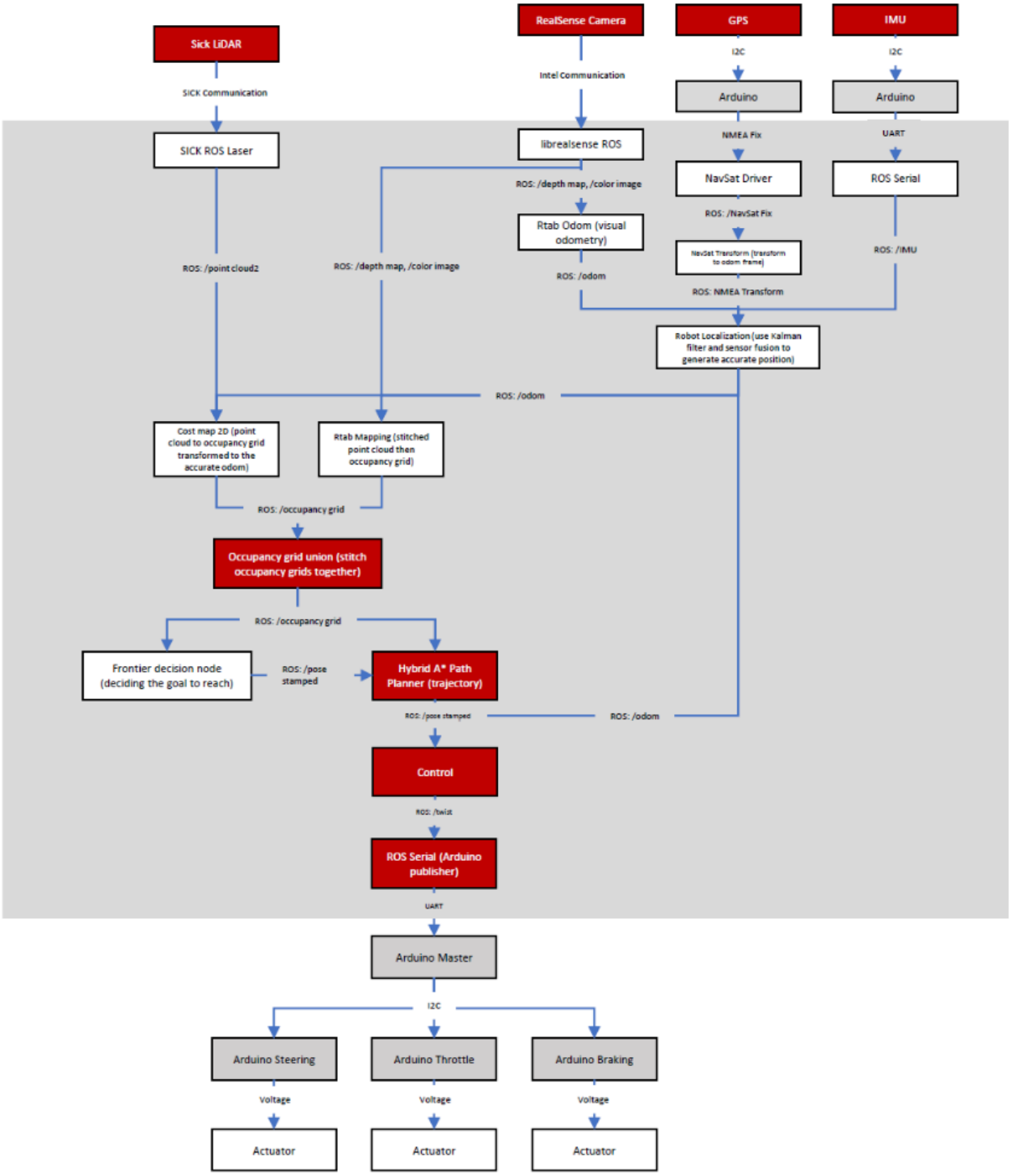


Figure 79. Software System Flow Diagram

7: System Integration & Results

Acceleration, braking, and steering, also categorized as the three major mechanical systems, were examined to determine the most efficient way of controlling them. The acceleration signal is entirely electrical, meaning it could be replicated as a signal from a microcontroller. The braking system required actuation of the physical brake pedal, so a linear actuator equipped with a potentiometer was used to control this system. Finally, the steering system was actuated using a rotational window motor that can be disconnected with a simple clutch. The shaft connecting the steering system to the steering wheel was equipped with a rotary potentiometer to detect the current steering position of the vehicle.

An Intel® RealSense™ Depth Camera, a SICK 3D LiDAR, a GPS module, and two consumer grade motion sensors were purchased and mounted to the vehicle for data collection purposes. Microcontrollers are used for actuation and microprocessors are used for data analysis. After issues with the grounding of the vehicle were solved, the team created a comprehensive drive-by-wire vehicle with all necessary sensors for autonomy mounted to it.

Fully combining the three actuators and the various sensors for level three autonomy required the design and implementation of a software architecture that would efficiently acquire and interpret all system data to call upon the appropriate actuator and drive the vehicle. Based on the combined and analyzed sensor data, the MCU generates a path for the vehicle to traverse. The MCU will actuate the vehicle by commanding a master of an I^2C bus with a message containing a SlaveID followed by data. Once the master node receives this message from the MCU, it commands the appropriate slave to move the actuator it is responsible for, to the desired position. SLAM is used with a GPS and an IMU to assist with localizing the vehicle in its environment, while Hybrid A* was chosen as the primary path finding technique and an A* search algorithm was implemented to allow the vehicle to generate intermediary goal coordinates to assist with traversing towards the final goal coordinate. ROS provides libraries, tools, and basic operating system services such that it can be used as the primary framework for data interpretation and communication between subsystems connected to the MCU.

The integration of the mechanical systems, electrical system, sensors and software architecture involved extensive testing. Similar to any robotics project, initially very little or

none of the added features (mechanical, electrical, or software) worked properly. Between the various grounding issues, bugs in software causing data type overflow, and improper braking mechanisms, the vehicle occasionally accelerated to full speed when told to only move 0.25 meters per second. The team used the WPI Gateway Garage for testing purposes, with explicit permission from WPI Police, and typically tested in five-hour increments. SAE assisted the team with transporting the vehicle and any needed testing materials to the garage for almost every testing session.

The first feature that the team tested was the RC capability. Unfortunately, performance was flaky, with connections coming loose from the embedded control system and the linear potentiometer, which normally gives feedback, was disconnecting from the steering rack. Although a minor set-back, the team used any spare testing time as opportunities to collect ROS bags of sensor data that could later be used in simulation and to test the software features in the lab.

Testing at the Gateway Garage was not particularly convenient as the team could only test on weekends when the Garage was mostly empty. However, taking away from bugs and setbacks found at the testing site, the team could debug on the dolly inside of the Higgins MQP lab. These debugging sessions on the dolly allowed the team to continue making significant progress in between outdoor testing sessions. The grounding issue (discussed in Section 5.2) was first re-discovered in one of these critical testing sessions when the car unexpectedly accelerated when commanded to turn. Rigorous debugging on the dolly during the following weeks allowed the team to come back the next week with the bug completely resolved.

Once RC capability was sufficiently tested and the team was confident that a drive-by-wire vehicle had been created, the vehicle was given a pre-programmed path and could drive that path autonomously. Figure 80 shows a team member in the vehicle traversing a parking lot given a pre-programmed path with no human intervention. In the figure, the vehicle is in motion and avoiding obstacles, but the driver’s hands are not touching the steering wheel, nor are his feet touching either the brake or throttle pedals.

Figure 80. Baja Vehicle Traversing Parking Lot with No Human Intervention

After this success, the team decided that it was time to add decision-making to the vehicle software capabilities. With the Jetson and its software communicating within its different subsystems using ROS, it was imperative that the team add the embedded control system as a "module" to the entire system. As discussed in Section 6.2, the embedded control system was added to this network, and the commands sent to a TurtleBot platform were immediately translated and sent to the vehicle as well. This test was twofold: it showed a platform using this projects software in simulation, as well as demonstrated modularity. The same software was controlling a twenty-pound robot, and a six-hundred-pound off-road vehicle.

Integrating the embedded control system with ROS allowed the team to put all parts of the system together. The team quickly realized the Jetson was running out of computing power, and although the vehicle can simultaneously localize and map within the unknown environment, it does not have enough computational resources to simultaneously plan a path towards the goal and implement vehicle velocity control. It should be noted that this is not a function of hardware but rather a function of the implementation. The Jetson is capable of performing all of these tasks, but the team did not properly utilize its GPU performance capabilities and did not program tasks in parallel, something the Jetson excels at. This was not implemented due to time

constraints. To compensate for this, the team first made a map by manually driving the vehicle around obstacles. The course the team setup is shown in Figure 81.

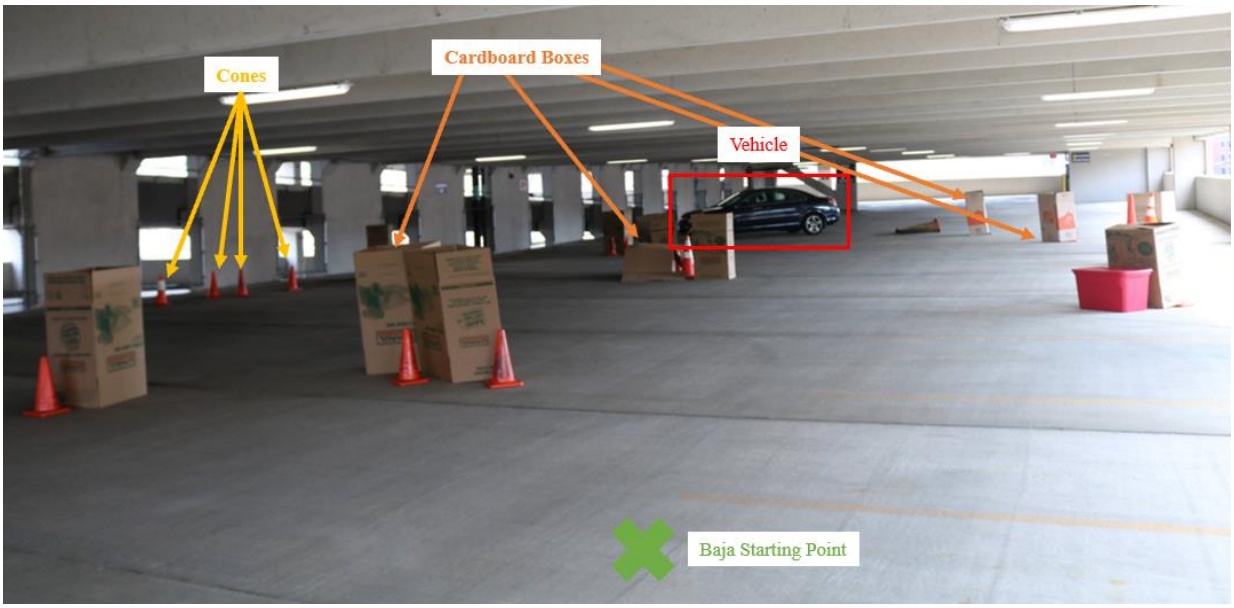


Figure 81. Obstacles Setup for Testing in Gateway Garage on 4/15

Using the data from the sensors obtained from manually driving the vehicle through the aforementioned course, the team was able to create both a 3D point cloud and a 2D occupancy grid of the space, as shown in Figure 82 and Figure 83.

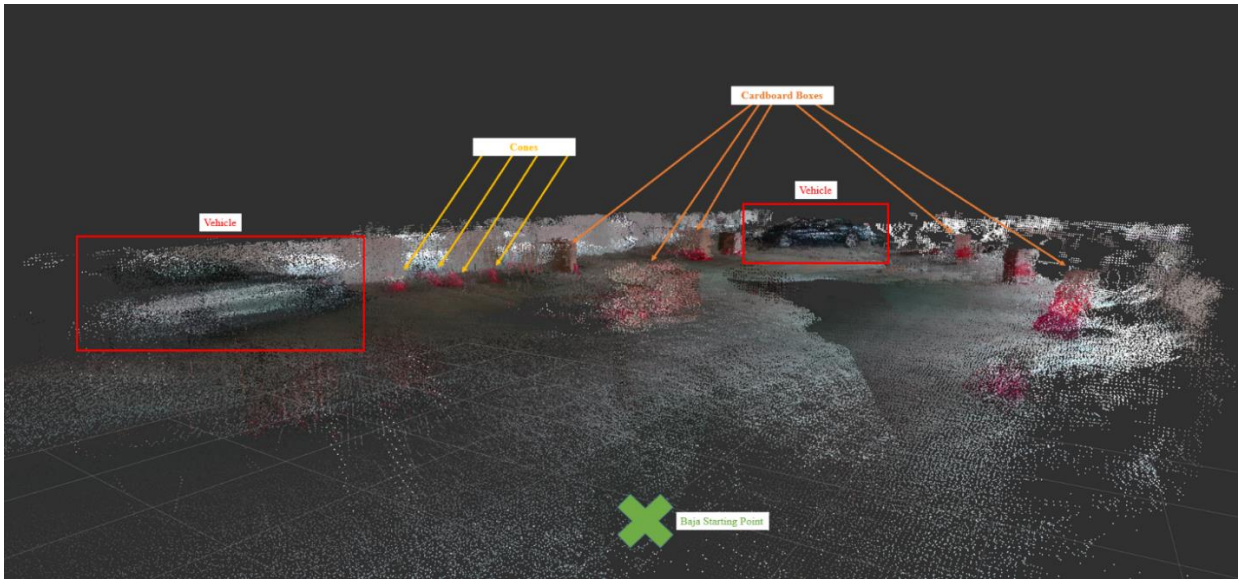


Figure 82. 3D Point Cloud of Obstacles Setup for Testing in Gateway Garage on 4/15

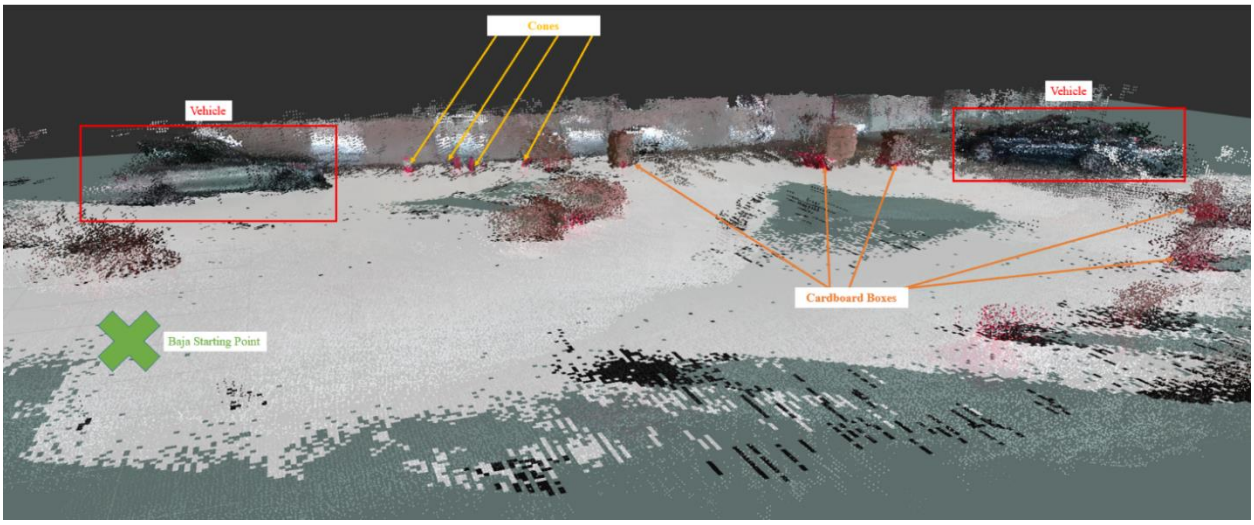


Figure 83. 3D Point Cloud & 2D Occupancy Grid of Obstacles Setup for Testing in Gateway Garage on 4/15

Using this 2D Occupancy Grid, the Hybrid A* path planning algorithm generates a path on this map to the goal destination, as shown in Figure 78. This path, which is an array of poses, avoids the obstacles that are in front and around the vehicle, and is sent to the velocity control software. The velocity control software works in conjunction with the localization software to determine the correct instruction set to send to the embedded control system, which ultimately moves the vehicle along the drawn path. The platform could use tuning depending on the vehicle it is operating on, but the vehicle is capable of autonomously navigating an environment given a map and is by definition, a Level 3 autonomous vehicle.

This project has left future teams with a very capable platform. This platform is an excellent starting point for SAE's upcoming autonomous Baja competition, and also leaves the door wide open for new branching off points in the field of Swarm robotics (multiple vehicles communicate with one-another) and search and rescue purposes. Chapter 8 discusses future opportunities in greater detail.

8: Recommendations, Future Opportunities & Applications

At the conclusion of this MQP, all goals had been achieved. The Baja vehicle was transformed into an autonomous system capable of driving to a given location while avoiding obstacles. Specifically, the Baja vehicle could drive from a starting location to a specified end location in a parking lot. Obstacles set-up by the team were simple in shape and consisted of traffic cones, cardboard boxes, or large vehicles.

When working on the physical electronics on the vehicle, future teams are advised to be careful with vehicle grounds and to use solid core wires or thick threaded wires. If there is ever an issue of unintended acceleration, steering not turning to the desired location, or the brake actuator not retracting, the grounds on the vehicle should be checked before anything else. Future teams also might consider purchasing a new Jetson TX2, as the micro-USB port on the current Jetson TX2 became detached (this is a fairly common problem with Jetsons). To do this, they would need to back up the current Jetson to a host computer, and then flash the new Jetson with the current Jetson's image. Another recommendation for future work with the Jetson is to implement multithreaded computing on the system. This would very significantly decrease communication and computing times and allow computationally demanding functions to operate simultaneously.

When working on the embedded software, future teams are advised to look into why the IMU gravity compensation algorithm does not function properly when integrated with ROS, but otherwise works as desired, compensating for different slopes the car is traversing. They should also watch out for data type overflows and improper conversion factors.

When working on the software architecture, they are advised to optimize the Hybrid A* package functions. While the Hybrid A* package functions appropriately and can plan a path, it is not optimized to work with larger maps and high memory intensive data sets. An additional step a new MQP team could take is to write a custom Hybrid A* package. Not only would this take care of the current issues with the Hybrid A* package, they would also be able to combine the frontierFinder into the rest of the Hybrid A* package, allowing complete path planning control.

Future teams that continue the work on this project would ideally advance the autonomous capabilities of the vehicle, such as enabling it to drive autonomously in denser environments or modifying the software system to make it modular and operate in real-time. They could also modify the software system to include a user interface, where users can input a multitude of varying vehicle parameters in simulation, and each vehicle can then autonomously drive. They also might consider the implementation of wireless network communication systems, allowing for the use of a wireless button with GPS location that when pressed, would essentially "call the vehicle" to that location.

A vehicle meeting the previously mentioned abilities would be an object of interest for search and rescue operations, such as in the medical or military fields. It could be used to transport goods through dangerous terrains or in a less intimidating atmosphere. The vehicle could also be used for public transportation.

Similarly, it is difficult to find autonomous public vehicle testing platforms that allow testing at a real-world scale that are simultaneously accessible and easy to modify and experiment with. This vehicle could be designed in the future to solve this issue, providing a physical platform that is catered to autonomous car test and development.

The mass production and use of autonomous, off-road-capable, search-and-rescue vehicles would save many lives. Imagine a world where hikers would have a button, similar to Life Alert [115], an emergency alert system made to assist the elderly. The button would notify dispatchers of the hiker's GPS location, and could immediately send an autonomous, off-road vehicle to that location. People stranded due to the after effects of a natural disaster would benefit from this vehicle. "The use of robots that can assist first responders and thus reduce their risk is desirable" [116]. This technology could be implemented in the form of swarm robotics [117], which could go on missions to help find victims after such a disaster. In the military, sending an autonomous, off-road vehicle to injured soldiers in unsafe areas would eliminate the need for a team to go out on a rescue mission to save that soldier. "Finding new ways to lower the risk soldiers face from hostile actors — an effort known as force protection — has become a top priority for various branches of the military" [118].

The military could also use this vehicle to take the task of driving away from soldiers who might need to prioritize other tasks, as well as for delivering items to specified locations. The idea would be to "take the man out of the machine" [119]. Myron Mills, program manager

for autonomous systems at Lockheed, told Foxtrot Alpha in an interview that you “can take more soldiers out of harm’s way if you’re in environments that are not secured, and even in environments that are secured. You can start out by reducing the number of drivers” [120].

In another view, the system this project is designing could be useful to improve public transportation. Several companies have already worked towards fully autonomous on-road driving vehicles for public transportation as mentioned in Chapter 2. The vehicle AI could be enhanced to function appropriately for on-road country and city road environments, while also being used for off-road personal transportation. The electrical and software system could also be implemented on other vehicles based on user personal preference, as opposed to requiring the purchase of a specific brand of vehicle.

Works Cited

[1] A. Marshall, "Wired," 7 November 2017. [Online]. Available: https://www.wired.com/story/self-driving-cars-rand-report/. [Accessed 9 December 2018].

[2] Allianz, "Materials used to make a car infographic," Allianz, 2018. [Online]. Available: https://www.allianz.com.au/car-insurance/infographic/materials-used-to-make-a-car-infographic/. [Accessed March 2018].

[3] J. Murdoch, "Cars And Metal, Metal and Cars," ETF, 17 November 2008. [Online]. Available: http://www.etf.com/sections/features-and-news/1289-cars-and-metal-metal-and-cars?nopaging=1. [Accessed April 2018].

[4] T. Swift, M. Moore and E. Sanchez, Plastics and polymer composites in light vehicles, Economics and Statistics Department/American Chemistry Council Google Scholar, 2017.

[5] A. Conover, Director, *Electric Cars Aren't As Green As You Think.* [Film]. TruTV.

[6] J. Voelcker, "1.2 Billion Vehicles On World's Roads Now, 2 Billion By 2035: Report," Green Car Reports, 29 July 2014. [Online]. Available: https://www.greencarreports.com/news/1093560_1-2-billion-vehicles-on-worlds-roads-now-2-billion-by-2035-report. [Accessed April 2018].

[7] M. Burgess, "We went off-road in Jaguar Land Rover's autonomous car," Wired, 22 July 2016. [Online]. Available: https://www.wired.co.uk/article/self-driving-autonomous-land-rover-jaguar-technology. [Accessed April 2018].

[8] admin, "MEET THE PENTAGON'S NEXT AUTONOMOUS DUNE BUGGY FOR US SPECIAL FORCES," Stock Board Asset, 13 February 2018. [Online]. Available: https://stockboardasset.com/insights-and-research/meet-pentagons-next-autonomous-dune-buggy-special-forces/.

[9] SAE International, "SAE International Overview - About," 2018. [Online]. Available: https://www.sae.org/attend/student-events/baja-sae-rochester/about. [Accessed 9 December 2018].

[10] E. Garsten, "Forbes - Sharp Growth In Autonomous Car Market Value Predicted But May Be Stalled By Rise In Consumer Fear," 13 August 2018. [Online]. Available: https://www.forbes.com/sites/edgarsten/2018/08/13/sharp-growth-in-autonomous-car-market-value-predicted-but-may-be-stalled-by-rise-in-consumer-fear/#48290ee3617c. [Accessed 9 December 2018].

[11] S. O'Kane, "The Verge," 19 April 2018. [Online]. Available: https://www.theverge.com/transportation/2018/4/19/17204044/tesla-waymo-self-driving-car-data-simulation. [Accessed 9 December 2018].

[12] T. Kanade, C. Thorpe and W. Whittaker, "Autonomous land vehicle project at CMU," in *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, Cincinnati, 1986.

[13] M. DeBord, "Business Insider - Waymo could be worth as much $175 billion — here's a brief history of the Google Car project," 9 September 2017. [Online]. Available: https://www.businessinsider.com/google-car-project-history-2018-8. [Accessed 9 December 2018].

[14] D. Radovanovic and D. Muoio, "This is what the evolution of self-driving cars looks like," Business Insider, 4 October 2016. [Online]. Available: http://www.businessinsider.com/what-are-the-different-levels-of-driverless-cars-2016-10. [Accessed April 2018].

[15] C. Thompson, "Here's how Tesla's Autopilot works," Business Insider, 1 July 2016. [Online]. Available: https://www.businessinsider.com/how-teslas-autopilot-works-2016-7. [Accessed 26 January 2019].

[16] F. Lambert, "A look at Tesla's new Autopilot hardware suite: 8 cameras, 1 radar, ultrasonics & new supercomputer," electrek, 20 October 2016. [Online]. Available: https://electrek.co/2016/10/20/tesla-new-autopilot-hardware-suite-camera-nvidia-tesla-vision/. [Accessed March 2018].

[17] Waymo, "Waymo," Waymo, 2018. [Online]. Available: https://waymo.com/. [Accessed April 2018].

[18] Waymo, "Waymo Safety Report," Waymo, 2018. [Online]. Available: https://waymo.com/safety/. [Accessed April 2018].

[19] J. Barrett, R. Fitzpatrick, C. Gamache, R. Madera, A. Panzica, B. Roy, D. Sacco, V. Truchanovicius and B. Wang, "Prometheus - Design and Realization of an Intelligent Ground Vehicle," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2010.

[20] G. Hotz, "comma.ai," Comma.ai, 2018. [Online]. Available: https://comma.ai/. [Accessed 1 May 2018].

[21] D. J. Baker, A. W. Briskman, N. D. F. Camila, M. A. Mahan, J. G. Perkins and J. A. Pierce, "Goat Cart; An Autonomous Golf Cart," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2018.

[22] G. S. Isko, "Robocart Machine Vision Framework," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2016.

[23] M. Weinmann, "Preliminaries of 3D Point Cloud Processing," in *Reconstruction and Analysis of 3D Scenes*, Springer International Publishing Switzerland, 2016, p. 19.

[24] "Deep Cycle Battery FAQ," Northern Arizona Wind & Sun, 2018. [Online]. Available: https://www.solar-electric.com/learning-center/batteries-and-charging/deep-cycle-battery-faq.html#Starting,%20Marine,%20and%20Deep-Cycle%20Batteries. [Accessed April 2018].

[25] D. Crook and C. Cothrun, "The Care and Feeding of Rechargable Batteries - Revisited," Ingenuity Inc, 1997. [Online]. Available: https://projects.eri.ucsb.edu/scec/pbic/equip/man/battery_care.html. [Accessed March 2018].

[26] Jippie, "Batteries in series with different amp-hour ratings," StackExchange, 12 March 2013. [Online]. Available: https://electronics.stackexchange.com/questions/60732/batteries-in-series-with-different-amp-hour-ratings. [Accessed March 2018].

[27] "Fuses and Types of Fuses," Electrical Technology, 7 November 2014. [Online]. Available: https://www.electricaltechnology.org/2014/11/fuse-types-of-fuses.html. [Accessed March 2018].

[28] J. Lakkonen, "What Is Drive-by-Wire Technology?," Lifewire - Dotdash, 21 March 2017. [Online]. Available: https://www.lifewire.com/what-is-drive-by-wire-534825. [Accessed March 2018].

[29] Wordpress, "Proton car virtual showroom - Anti Lock Brake System (ABS)," Wordpress, 14 November 2010. [Online]. Available: https://protoncar.wordpress.com/2010/11/14/anti-lock-brake-system-abs/. [Accessed 2 February 2019].

[30] M. Fernie, "How Electric Power Assisted Steering Works, And Why It's Better Than Hydraulic," CarThrottle, 2016. [Online]. Available: https://www.carthrottle.com/post/electronic-power-assisted-steering-how-does-it-work/. [Accessed April 2018].

[31] D. Sherman, "Are We Losing Touch? A Comprehenseive Comparison Test of Electric and Hydraulic Steering Assist," Car and Driver, January 2012. [Online]. Available: https://www.caranddriver.com/features/electric-vs-hydraulic-steering-a-comprehensive-comparison-test-feature. [Accessed April 2018].

[32] M. Harrer and P. Pfeffer, Steering Handbook, Springer International Publishing Switzerland, 2017.

[33] MatWeb, "ASTM A36 Steel, bar," MatWeb Material Property Data, 1982-1999. [Online]. Available: http://www.matweb.com/search/datasheet_print.aspx?matguid=d1844977c5c8440cb9a3a967f8909c3a. [Accessed 9 October 2018].

[34] The Lincoln Electric Company, "Excalibur 308/308L-17," February 2013. [Online]. Available: https://m.lincolnelectric.com/assets/global/Products/Consumable_StainlessNickelandHighAlloy-Excalibur-Excalibur308308L-17/c6104.pdf. [Accessed 10 January 2019].

[35] L. Shaw, "Gear Nomenclature," Slideplayer, 2014. [Online]. Available: https://slideplayer.com/slide/1701535/. [Accessed 20 December 2018].

[36] A. Porreco, "Calculating Yield & Tensile Strength," Portland Bolt & Manufacturing Company, 1 December 2017. [Online]. Available: https://www.portlandbolt.com/technical/faqs/calculating-strength/. [Accessed 4 March 2019].

[37] Colorado State University, "Laboratory 1 Four-Bar Linkages," Colorado State University, [Online]. Available: https://www.engr.colostate.edu/~dga/mech324/Labs/Lab%201/MECH324%20-%20Lab%201.htm. [Accessed 11 March 2019].

[38] ME Mechanical, "Grashof's Law," ME Mechanical, 10 June 2017. [Online]. Available: https://me-mechanicalengineering.com/grashofs-law/. [Accessed 11 March 2019].

[39] "Stereo Vision," Mathworks, 2018. [Online]. Available: https://www.mathworks.com/discovery/stereo-vision.html. [Accessed March 2018].

[40] R. R. Tummala, "Autonomous Cars: Radar, Lidar, Stereo Cameras," Georgia Tech Packaging Research Center, Atlanta, 2017.

[41] Intel Corporation, "Intel® RealSense™ Depth Camera D400-Series," September 2017. [Online]. Available: https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf. [Accessed 28 January 2019].

[42] ABC Science, "How does radar work?," ABC Science, 17 March 2014. [Online]. Available: http://www.abc.net.au/science/articles/2014/03/17/3964782.htm. [Accessed 4 February 2019].

[43] National Oceanic and Atmospheric Administration (NOAA), "What is LIDAR?," National Oceanic and Atmospheric Administration Department of Commerce, 25 July 2018. [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html. [Accessed 4 February 2019].

[44] SICK Sensor Intelligence , "LD-MRS400001S01 Online data sheet," 13 September 2018. [Online]. Available: https://cdn.sick.com/media/pdf/2/42/942/dataSheet_LD-MRS400001S01_1052960_en.pdf. [Accessed 28 January 2019].

[45] SICK AG Waldkirch, "LD-MRS 3D LiDAR Sensors Operating Instructions," SICK AG, Waldkirch, 2017.

[46] C. M. Martinez, F. Zhang, D. Clarke, G. Hinz and D. Cao, "Feature uncertainty estimation in sensor fusion applied to autonomous vehicle location," in *Information Fusion (Fusion), 2017 20th International Conference*, Xi'an, 2017.

[47] MaxBotix, "LV-MaxSonar-EZ0 High Performance Sonar Range Finder," January 2007. [Online]. Available: http://www.robotstorehk.com/sensors/doc/LV-MaxSonar-EZ0-Datasheet.pdf. [Accessed 28 January 2019].

[48] SICK Sensor Intelligence, "Ultrasonic sensors," SICK AG, 2018. [Online]. Available: https://www.sick.com/us/en/distance-sensors/ultrasonic-sensors/c/g185671. [Accessed 17 December 2018].

[49] SICK Sensor Intelligence, "Ultrasonic sensors UC30," SICK AG, 2018. [Online]. Available: https://www.sick.com/us/en/distance-sensors/ultrasonic-sensors/uc30/c/g323151. [Accessed 17 December 2018].

[50] SICK Sensor Intelligence, "Ultrasonic sensors UM30," SICK AG, 2018. [Online]. Available: https://www.sick.com/us/en/distance-sensors/ultrasonic-sensors/um30/c/g185672. [Accessed 17 December 2018].

[51] SICK Sensor Intelligence, "Ultrasonic sensors UC30 - UC30-21416B," SICK AG, 2018. [Online]. Available: https://www.sick.com/us/en/distance-sensors/ultrasonic-sensors/uc30/uc30-21416b/p/p580213. [Accessed 17 December 2018].

[52] SICK AG Waldkirch, "Ultrasonic sensors - Ultimate ultrasonic sensor solution from SICK," SICK AG, Waldkirch, 2015.

[53] W. Rahiman and Z. Zainal, "An Overview of Development GPS Navigation for Autonomous Car," Universiti Sains Malaysia, Pulau Pinang, 2013.

[54] GlobalTop Technology Inc, "FGPMMOPA6H GPS Standalone Module Data Sheet," 2011. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf. [Accessed 9 April 2019].

[55] Adafruit, "Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3," Adafruit, 2018. [Online]. Available: https://www.adafruit.com/product/746. [Accessed 18 December 2018].

[56] Amazon, "Gowoops GPS Module with TTL Ceramic Passive Antenna for Arduino Raspberry Pi 2 3 B+ MCU," Amazon, 2018. [Online]. Available: https://www.amazon.com/Gowoops-Ceramic-Passive-Antenna-Raspberry/dp/B01AW5QYES. [Accessed 18 December 2018].

[57] u-blox, "NEO-6 u-blox 6 GPS Modules Data Sheet," u-box, Thalwil.

[58] H. R, "Amazon Customer Rewview," Amazon, 28 January 2018. [Online]. Available: https://www.amazon.com/gp/customer-reviews/R31HQWAQ4XZX27/ref=cm_cr_dp_d_rvw_ttl?ie=UTF8&ASIN=B01AW5QYES. [Accessed 18 December 2018].

[59] P. Furgale, J. Rehder and R. Siegwart, "Unified Temporal and Spatial Calibration for Multi-Sensor Systems," in *Intelligent Robots and SYstems (IROS) 2013 IEEE/RSJ International Conference on*, Tokyo, 2013.

[60] Z. Lin, Y. Xiong, H. Dai and X. Xia, "An Experimental Performance Evaluation of the Orientation Accuracy of Four Nine-Axis MEMS Motion Sensors," in *Enterprise Systems 2017 Fifth International Conference on*, 2017.

[61] M. F. Ruzaij, S. Neubert, N. Stoll and K. Thurow, "A Speed Compensation Algorithm for a Head Tilts Controller Used for Wheelchairs and Rehabilitation Applications," in *IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herl'any, 2017.

[62] K. Townsend, "Adafruit BNO055 Absolute Orientation Sensor," Adafruit, 23 October 2015. [Online]. Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview. [Accessed March 2018].

[63] K. Townsend, "Adafruit BNO055 Absolute Orientation Sensor," 2019 28 February. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bno055-absolute-orientation-sensor.pdf?timestamp=1554817205. [Accessed 2019 9 April].

[64] J. Chen, A. Huynh, Z. Jiang and Z. Wu, "Self-Driving On 1/10 Racer Car," Worcester Polytechnic Institute, Worcester, 2018.

[65] MIT, "Racecar," MIT Racecar, 2017. [Online]. Available: https://mit-racecar.github.io/. [Accessed 9 December 2018].

[66] A. Benini, M. J. Rutherford and K. P. Valanis, "Real-time, GPU-based pose estimation of UAV for autonomous takeoff and landing," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, Stockholm, 2016.

[67] NVIDIA Corporation, "Jetson TX2 Datasheet," NVIDIA, Santa Clara, 2014-2017.

[68] "The ABC's of PID's," The Society of Aerial Cinematography, 16 April 2015. [Online]. Available: https://thesoac.com/the-abcs-of-pids/. [Accessed March 2018].

[69] Amtel Corporation, "8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash," Amtel Corporation, San Jose, 2009.

[70] Open Source Robotics Foundation, "About ROS," Open Source Robotics Foundation, [Online]. Available: http://www.ros.org/about-ros/. [Accessed 9 April 2019].

[71] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," IEEE, 2017.

[72] SICK Sensor Intelligence, "LMS291-S05," 29 September 2016. [Online]. Available: https://www.sick.com/media/pdf/9/49/849/dataSheet_LMS291-S05_1018028_en.pdf. [Accessed 9 April 2019].

[73] Bosch Sensortec, "BNO055 USB Stick user guide," June 2016. [Online]. Available: https://ae-bst.resource.bosch.com/media/_tech/media/application_notes/BST-BNO055-AN009.pdf. [Accessed 25 March 2019].

[74] Adafruit, "Adafruit_GPS," GitHub Inc, 2019. [Online]. Available: https://github.com/adafruit/Adafruit_GPS. [Accessed 9 April 2019].

[75] Analog Devices, "Low Distortion, High Speed Rail-to-Rail Input/Output Amplifiers," 2013-2015. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8027_8028.pdf. [Accessed 12 March 2019].

[76] He Li Shun, "HLS-14F3 D - DC12V - C Datasheet," [Online]. Available: https://cdn.datasheetspdf.com/pdf-down/H/L/S/HLS-14F3D-DC12V-C-HLS.pdf. [Accessed 12 March 2019].

[77] Cytron Technologies, "MDD10A Dual Channel 10A DC Motor Driver User's Maual," June 2017. [Online]. Available: https://www.robotshop.com/media/files/pdf2/rb-cyt-153_-_mdd10a_users_manual_v2.0_-_2017-06.pdf. [Accessed 4 March 2019].

[78] M. Batra, "Dynamics and Model-Predictive Anti-Jerk Control of Connected Electric vehicles," University of Waterloo, Waterloo, 2018.

[79] Wikipedia, "Bus (computing)," Wikipedia, 1 January 2019. [Online]. Available: https://en.wikipedia.org/wiki/Bus_(computing). [Accessed 4 March 2019].

[80] CSS Electronics, "CAN Bus Explained - A Simple Intro (2019)," CSS Electronics, 2019. [Online]. Available: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en. [Accessed 14 April 2019].

[81] Corelis Inc, "SPI Tutorial," Corelis Inc, 2019. [Online]. Available: https://www.corelis.com/education/tutorials/spi-tutorial/. [Accessed 14 April 2019].

[82] Tom Sheldon and Big Sur Multimedia, "NAK (Negative Acknowledgment)," 2001. [Online]. Available: https://www.linktionary.com/n/nak.html. [Accessed 14 April 2019].

[83] J. W. van Dam, B. K. Krose and F. C. Groen, "Neural Network Applications in Sensor Fusion For an Autonomous Mobile Robot," Faculty of Mathematics, Computer Science Physics and Astronomy, University of Amsterdam, Amsterdam, 1995.

[84] W. Elmenreich, "An Introduction to Sensor Fusion," Institut f¨ur Technische Informatik Vienna University of Technology, Austria, Vienna, 2002.

[85] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *Intelligent Robots and Systems IROS '91. Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, Osaka, 1991.

[86] Wikipedia, "Recursive Bayesian estimation," Wikipedia, 25 August 2018. [Online]. Available: https://en.wikipedia.org/wiki/Recursive_Bayesian_estimation. [Accessed 19 December 2018].

[87] Wikipedia, "Kalman filter," Wikipedia, 17 December 2018. [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter. [Accessed 19 December 2018].

[88] Wikipedia, "Particle filter," Wikipedia, 16 December 2018. [Online]. Available: https://en.wikipedia.org/wiki/Particle_filter. [Accessed 19 December 2018].

[89] M. Ulusoy, "Understanding Kalman Filters, Part 3: An Optimal State Estimator," MathWorks, 27 March 2017. [Online]. Available: https://www.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html. [Accessed 21 December 2018].

[90] Bzarg, "How a Kalman filter works, in pictures," Bzarg, 11 August 2015. [Online]. Available: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/?fbclid=IwAR1X9ps5nXB1bVJzlV4-TTR-0E1O10c0vB1ZsgdsE6mFCLBkKNxb_zPvtjk2015. [Accessed 21 December 2018].

[91] U.S. Air Force, "GPS Accuracy," U.S. Air Force, 5 December 2017. [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/. [Accessed 7 April 2019].

[92] Vision Online Marketing Team, "What is Visual SLAM Technology and What is it Used For?," AIA - Vision Online, 15 May 2018. [Online]. Available: https://www.visiononline.org/blog-article.cfm/What-is-Visual-SLAM-Technology-and-What-is-it-Used-For/99. [Accessed 19 December 2018].

[93] Wikipedia, "Simultaneous localization and mapping," Wikipedia, 15 September 2018. [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping. [Accessed 19 December 2018].

[94] E. Karami, S. Prasad and M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images," Memorial University, 2017.

[95] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," Carnegie Mellon University, Pittsburgh, 1989.

[96] S. Rajko and S. M. LaValle, "A Pursuit Evasion BUG Algorithm," Iowa State University, Ames, 2001.

[97] A. Coles and A. Smith, 9 January 2007. [Online]. Available: http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume28/coles07a-html/node11.html#modifiedbestfs. [Accessed 19 December 2018].

[98] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in *Numerische Mathematik*, Secaucus, Springer-Verlag New York, Inc, 1959, pp. 269-271.

[99] A. Patel, "Introduction to A*," Stanford, 2019. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. [Accessed 14 April 2019].

[100] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *IEEE Transactions of Systems Science and Cybernetics*, 1968.

[101] DataGenetics, "Ackerman Steering," DataGenetics, 2009-2016. [Online]. Available: http://datagenetics.com/blog/december12016/index.html. [Accessed 14 April 2019].

[102] C. Urmson, J. Anhalt, D. Bartz, M. Clark, T. Galatali, A. Gutierrez, S. Harbaugh, J. Johnston, H. Kato, P. L. Koon, W. Messner, N. Miller, A. Mosher, K. Peterson, C. Ragusa, D. Ray, B. K. Smith, J. M. Snider, S. Spiker, J. C. Struble, J. Ziglar and W. (. L. Whittaker, "Journal of Field Robotics," *Carnegie Mellon University*, vol. 23, pp. 467-508, 2006.

[103] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, "Welcome to the KITTI Vision Benchmark Suite!," 2019. [Online]. Available: http://www.cvlibs.net/datasets/kitti/. [Accessed 10 April 2019].

[104] introlab, "RTAB-Map Real-Time Appearance-Based Mapping," GitHub Pages, [Online]. Available: http://introlab.github.io/rtabmap/. [Accessed April 7 2019].

[105] H. Singh, "Extended Kalman Filter: Why do we need an Extended Version?," Towards Data Science, 7 April 2018. [Online]. Available: http://introlab.github.io/rtabmap/. [Accessed 7 April 2019].

[106] T. Moore, "robot_localization wiki," Sphinx, 2016. [Online]. Available: http://docs.ros.org/melodic/api/robot_localization/html/index.html. [Accessed 4 April 2019].

[107] Washington and Lee University, "The Extended Kalman Filter: An Interactive Tutorial for Non-Experts," WLU, 19 December 2019. [Online]. Available: https://home.wlu.edu/~levys/kalman_tutorial/kalman.pdf. [Accessed 7 April 2019].

[108] Wolfram MathWorld, "Taylor Series," Wolfram MathWorld, 1999-2019. [Online]. Available: http://mathworld.wolfram.com/TaylorSeries.html. [Accessed 7 April 2019].

[109] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, "OctoMap - An Efficient Probabilistic 3D Mapping Framework Based on Octrees," in *Auton Robot*, New York, Springer Science+Business Media New York, 2013, pp. 189-206.

[110] K. Kurzer, "path_planner," GitHub Inc, 2019. [Online]. Available: https://github.com/karlkurzer/path_planner. [Accessed 7 April 2019].

[111] S. S. Belavadi, R. Beri and V. Malik, "Frontier Exploration Technique for 3D Autonomous SLAM Using K-Means Based Divisive Clustering," in *IEEE*, Kota Kinabalu, 2017.

[112] S. S. White and D. Catherman, "Mobile Robot Controller Performance over Unexpected Terrain Disturbances," in *SoutheastCon 2019*, Worcester, 2019.

[113] Open Source Robotics Foundation, "rosserial," Open Source Robotics Foundation, 1 October 2018. [Online]. Available: http://wiki.ros.org/rosserial. [Accessed 9 April 2019].

[114] Open Source Robotics Foundation, "TurtleBot," Open Source Robotics Foundation, 4 April 2018. [Online]. Available: http://wiki.ros.org/Robots/TurtleBot. [Accessed 9 April 2019].

[115] Life Alert Emergency Response Inc, "Life Alert," Life Alert Emergency Response Inc, 1987-2019. [Online]. Available: http://www.lifealert.com/. [Accessed 20 April 2019].

[116] C. Armbrust, G. De Cubber and K. Berns, "ICARUS Control Systems for Search and Rescue Robots," in *Field and Assistive Robotics - Advaces in SYstems and Algorithms*, Shaker Verlag, 2014.

[117] Wyss Institute, "Programmable Robot Swarms," President and Fellows of Harvard College, 2019. [Online]. Available: https://wyss.harvard.edu/technology/programmable-robot-swarms/. [Accessed 20 April 2019].

[118] P. Holley, "The military's latest plan to save lives on the battlefield: Building driverless vehicles," THe Washington Post, 2 May 2018. [Online]. Available: https://www.washingtonpost.com/news/innovations/wp/2018/05/02/the-militarys-latest-plan-to-save-lives-on-the-battlefield-building-driverless-vehicles/?noredirect=on&utm_term=.80e878193275. [Accessed 1 October 2018].

[119] G. Evans, "Driverless vehicles in the military – will the potential be realised?," Army Technology, 12 February 2018. [Online]. Available: https://www.army-technology.com/features/driverless-vehicles-military/. [Accessed October 2018].

[120] R. Felton, "Lockheed Martin's Autonomous Military Vehicles Aim To Save Lives In A Different Way," Foxtrot Alpha, 8 February 2017. [Online]. Available: https://foxtrotalpha.jalopnik.com/lockheed-martins-autonomous-military-vehicles-aim-to-sa-1792128164. [Accessed October 2018].

[121] Adafruit, "Pixy2 CMUcam5 Sensor," Adafruit, [Online]. Available: https://www.adafruit.com/product/1906?gclid=Cj0KCQjwn-bWBRDGARIsAPS1svt-__3RyfhSTYgpBGF0OBZ2W-_6NEat12aG5rOZ7vliQMuTaUax-p4aArJKEALw_wcB. [Accessed April 2018].

[122] H. Piggott, "ASK THE EXPERTS: Batteries in Series & Parallel," Home Power Inc, December/January 2008. [Online]. Available: https://www.homepower.com/articles/solar-electricity/design-installation/ask-experts-batteries-series-parallel. [Accessed March 2018].

[123] L. P. N. Sekar, A. Santos and O. Beltramello, "IMU Drift Reduction for Augmented Reality Applications," in *Augmented and Virtual Reality Second International Conference, AVR 2015*, Lecce, 2015.

[124] MathWorks, "Occupancy Grids," MathWorks, 2018. [Online]. Available: https://www.mathworks.com/help/robotics/ug/occupancy-grids.html. [Accessed 7 October 2018].

[125] Wikipedia, "Markov property," Wikipedia, 8 July 2018. [Online]. Available: https://en.wikipedia.org/wiki/Markov_property. [Accessed 19 December 2018].

[126] R. C. Crimmins and R. Wang, "Autonomous Ground Vehicle Prototype via Steering-, Throttle-, and Brake-by Wire Modules," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2016.

[127] K. Cederberg, R. Dall'Orso, C. L. Figuereo-Supraner and P. L. Long, "Goat Cart: The Autonomous Golf Cart," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2017.

[128] E. A. Miller, "Robocart: System Design for the First Generation Autonomous Golf Cart," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2015.

[129] P. Sahay, "Robocart: Autonomous Ground Vehicle," Major Qualifying Project, Worcester Polytechnic Institute, Worcester, 2015.

Appendix A: Proposed Budget

Table 20: Budget

Category	Item	Quantity	Unit Cost	Total Cost
Hardware	12V Deep Cycle Battery	4	$250	$1000
Hardware	Linear Actuator	1	$250	$250
Hardware	High Torque Motor	1	$200	$200
Hardware	Metal Stock		$500	$500
Hardware	Fasteners		$100	$100
Electrical	MCU	1	$600	$600
Electrical	Arduino Mega	5	$20	$100
Electrical	Wires, switches, connectors, etc.		$500	$500
Sensors	#D LiDAR	1	$7500	$7500
Sensors	3D Camera	1	$200	$200
Sensors	2D Camera	3	$100	$300
Sensors	IMU	4	$75	$300
Sensors	GPS Module	1	$125	$125
Sensors	Encoders	10	$25	$250
Sensors	Ultrasonic	5	$30	150
Sensors	Potentiometers	5	$25	$125
Equipment	Cell Phone for Development	2	$400	$800
Total:				$13,000

Appendix B: CAD Model

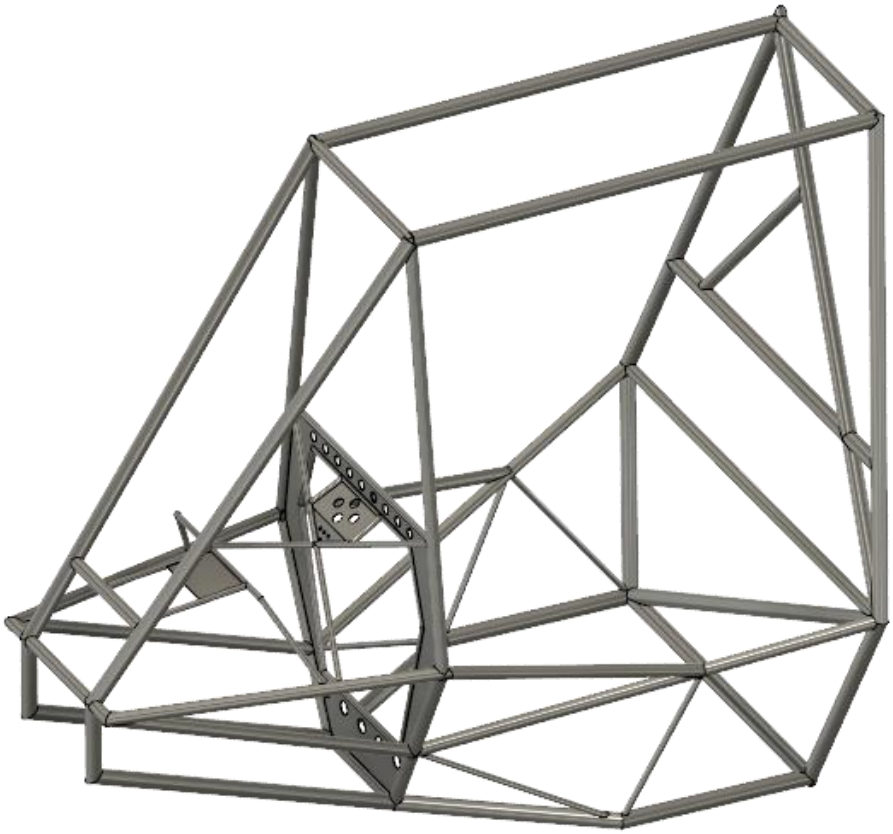
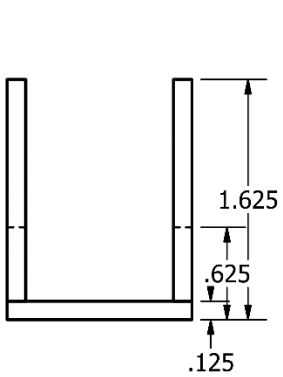
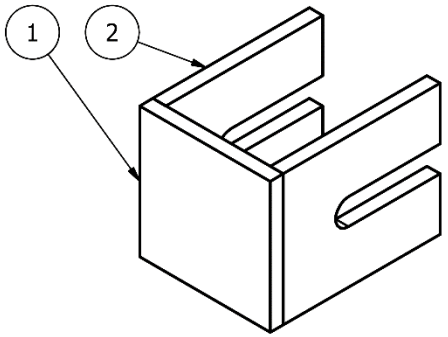


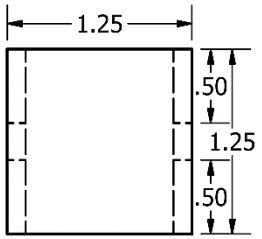
Figure 84: Baja Vehicle Chassis CAD

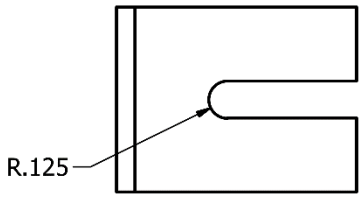
Appendix C: Brakes: Linear Actuator Mount Drawings

PARTS LIST						
ITEM	PART NUMBER	QTY	MATERIAL	VENDOR	STOCK NUMBER	ESTIMATED COST
1	1.25x0.125-bar-stock-1.000	1	Steel, Mild	Home Depot	204225724	$2.50
2	1.25x1.50x0.125-bar-stock-u-shape	2	Steel, Mild	Home Depot	204225724	$2.50

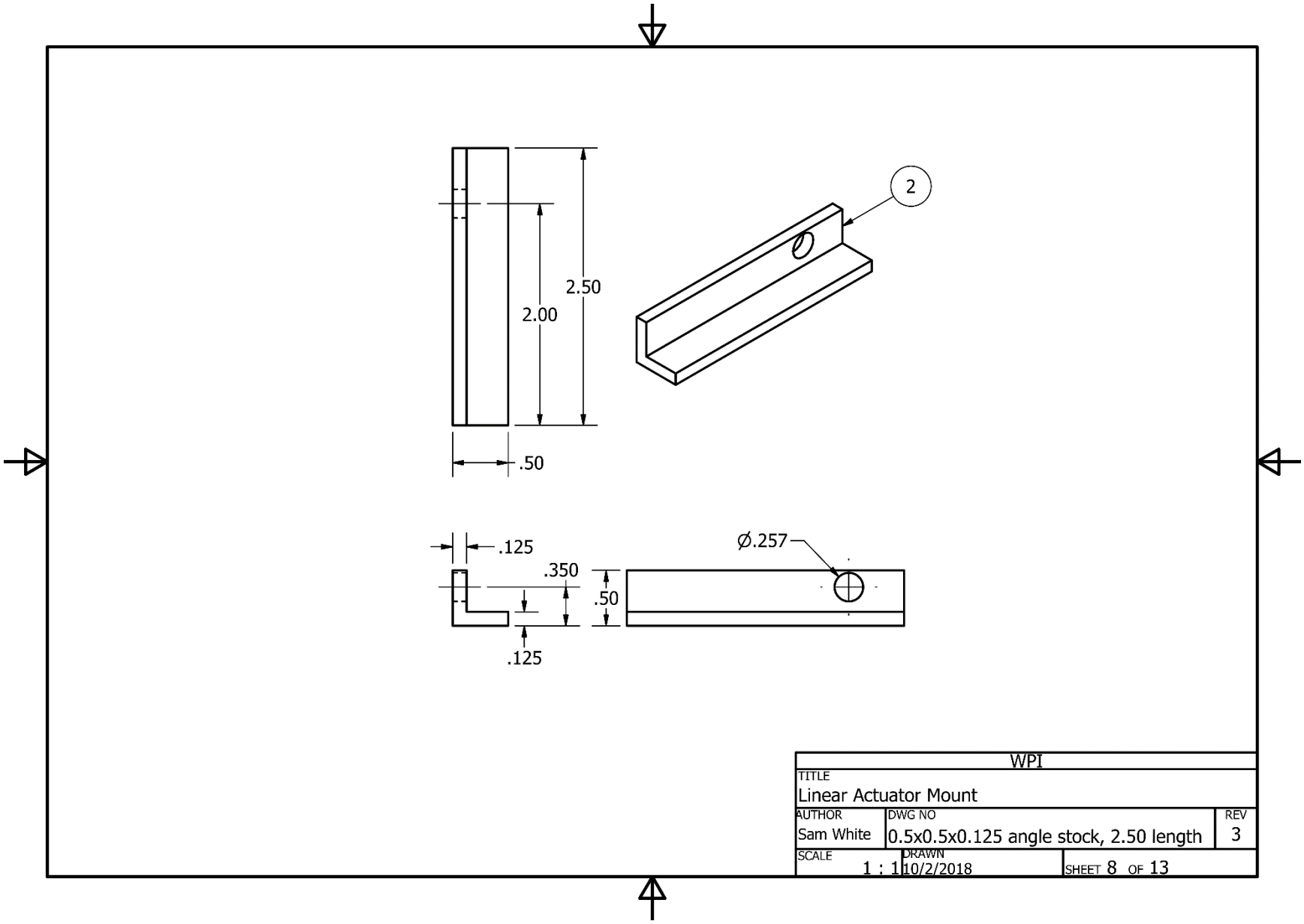


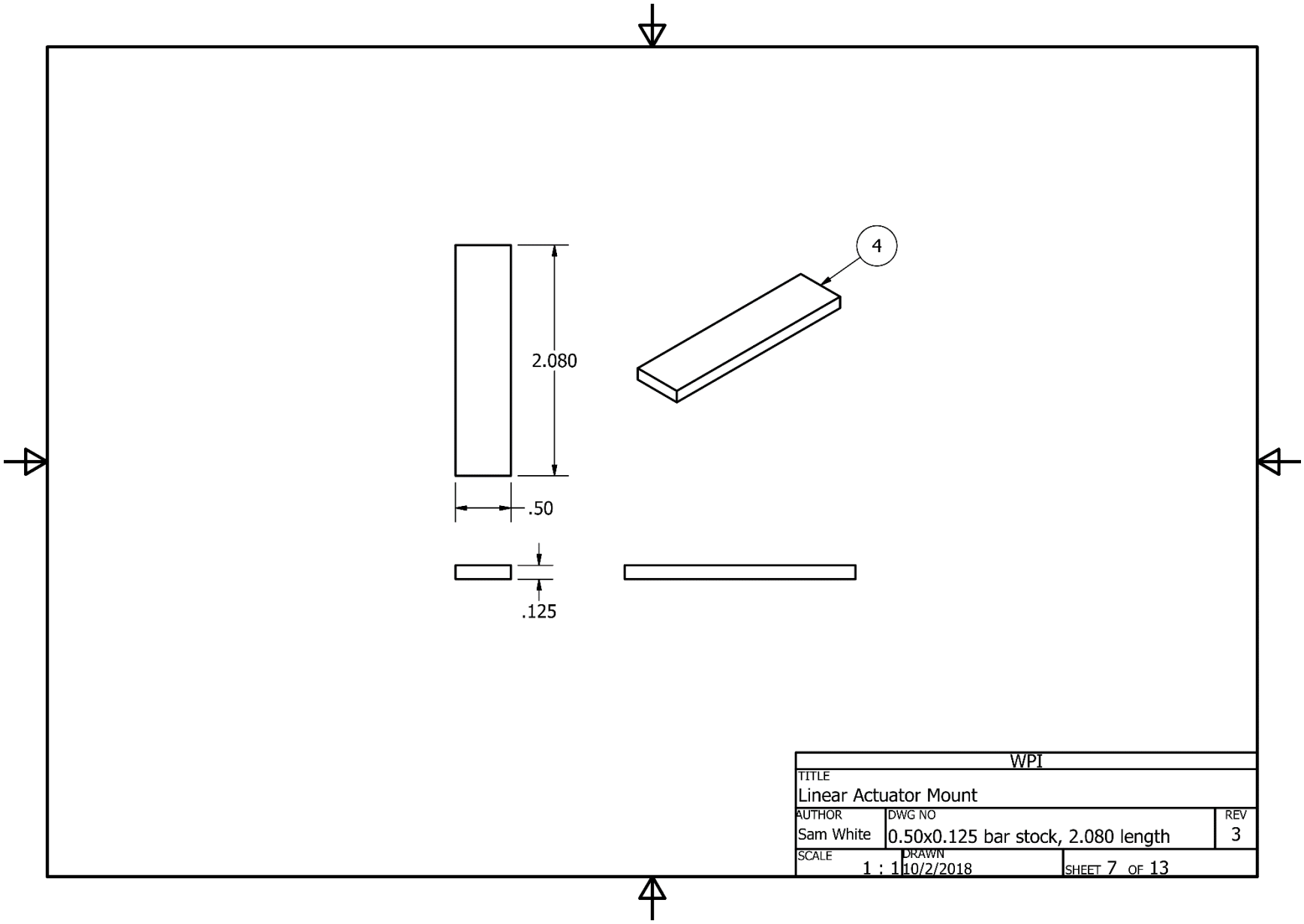


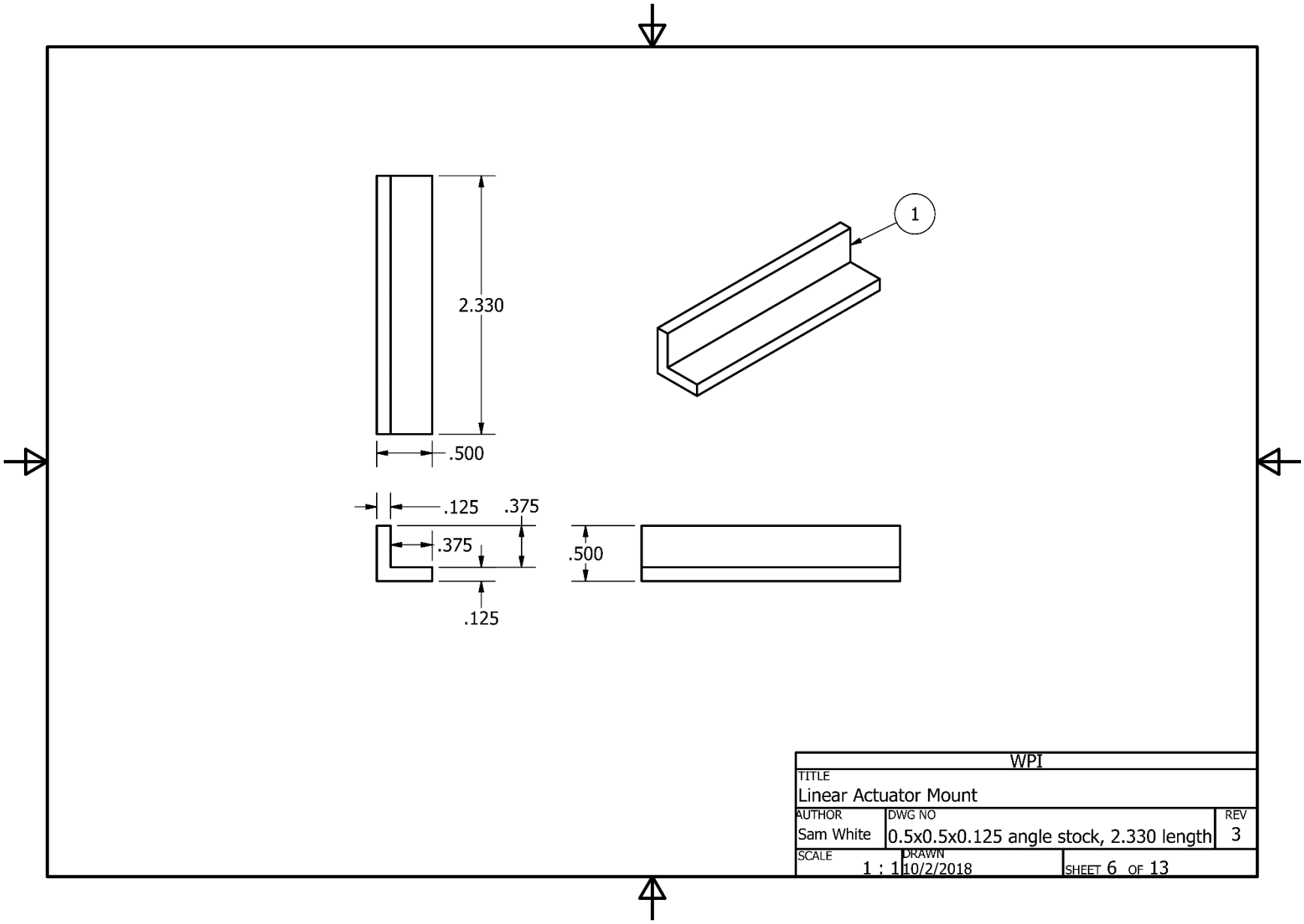




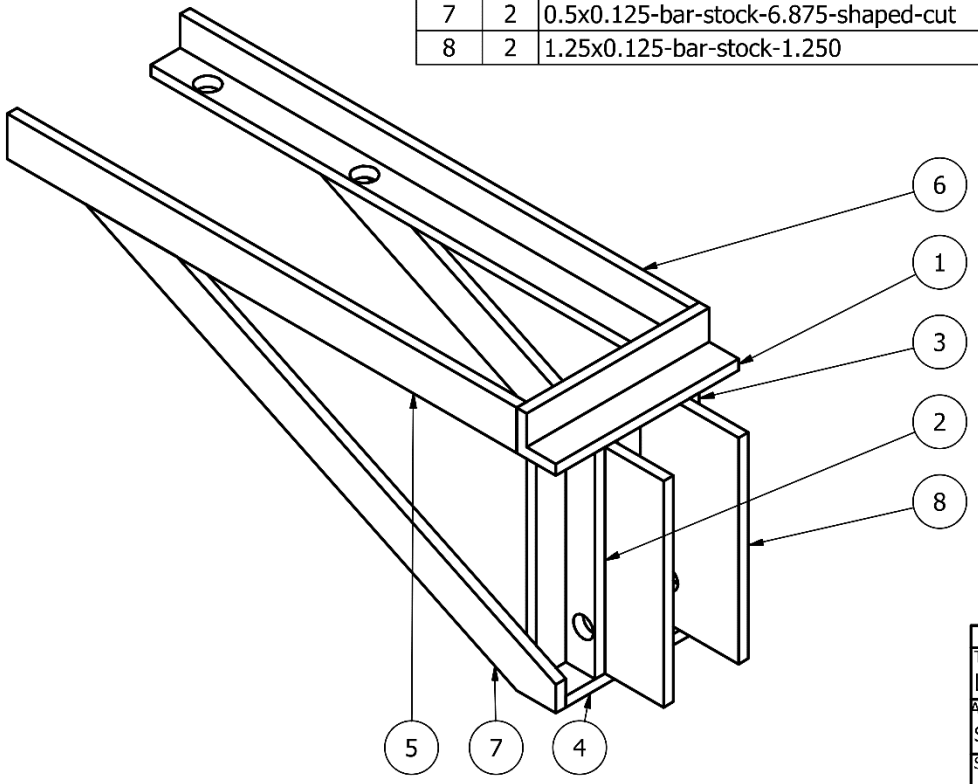
WPI		
TITLE: Linear Actuator Mount		
AUTHOR: Sam White	DWG NO: Brake Pedal Mount Assembly	REV: 3
SCALE: 1 : 1	DRAWN: 10/2/2018	SHEET 1 OF 13


2
2.50
2.00
.50
.125
.350
.50
.125
Ø.257
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.5x0.5x0.125 angle stock, 2.50 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 8 OF 13


4
2.080
.50
.125
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.50x0.125 bar stock, 2.080 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 7 OF 13


2.330
.500
.125
.375
.375
.125
.500
1
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.5x0.5x0.125 angle stock, 2.330 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 6 OF 13

PARTS LIST						
ITEM	QTY	PART NUMBER	MATERIAL	VENDOR	STOCK NUMBER	ESTIMATED COST
1	1	0.5x0.5x0.125-angle-stock-2.330	Steel, Mild	Home Depot	204225743	$2.00
2	1	0.5x0.5x0.125-angle-stock-2.500-hole-a	Steel, Mild	Home Depot	204225743	$2.00
3	1	0.5x0.5x0.125-angle-stock-2.500-hole-b	Steel, Mild	Home Depot	204225743	$2.00
4	1	0.5x0.125-bar-stock-2.080	Steel, Mild	Home Depot	204225773	$2.50
5	1	0.5x0.5x0.125-angle-stock-6.500-hole-b	Steel, Mild	Home Depot	204225743	$2.00
6	1	0.5x0.5x0.125-angle-stock-6.500-hole-a	Steel, Mild	Home Depot	204225743	$2.00
7	2	0.5x0.125-bar-stock-6.875-shaped-cut	Steel, Mild	Home Depot	204225773	$2.50
8	2	1.25x0.125-bar-stock-1.250	Steel, Mild	Home Depot	204225724	$2.50

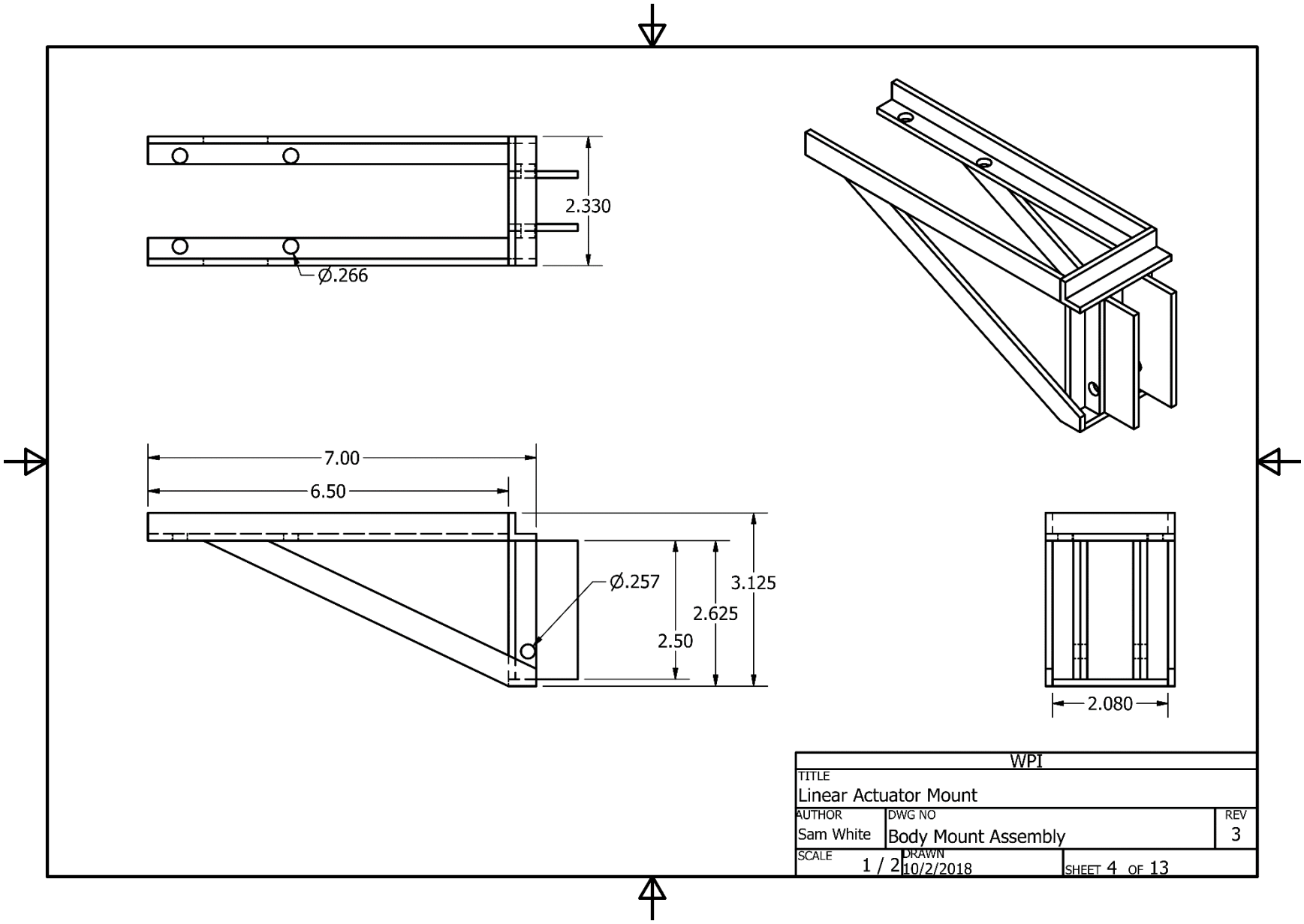


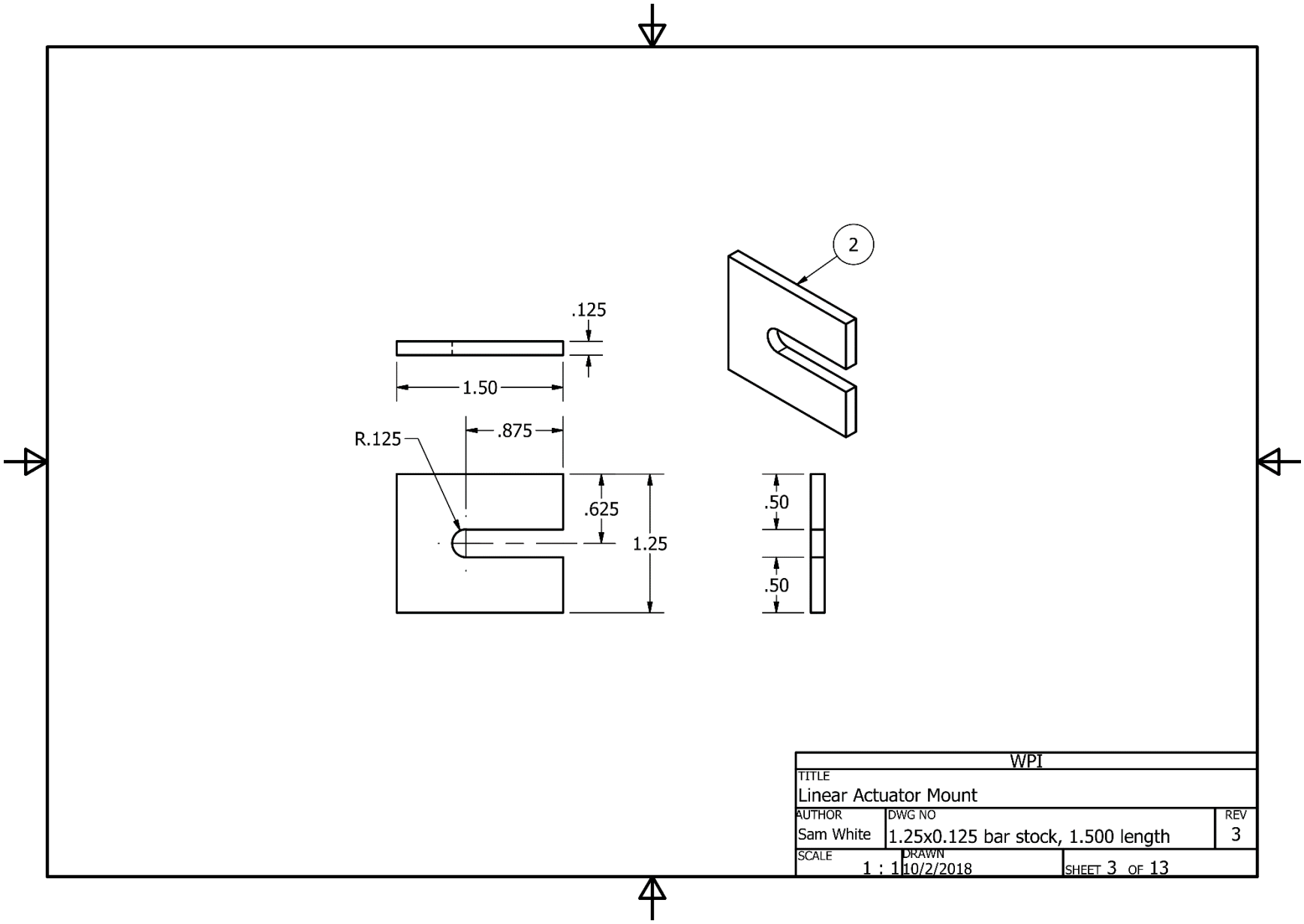
WPI

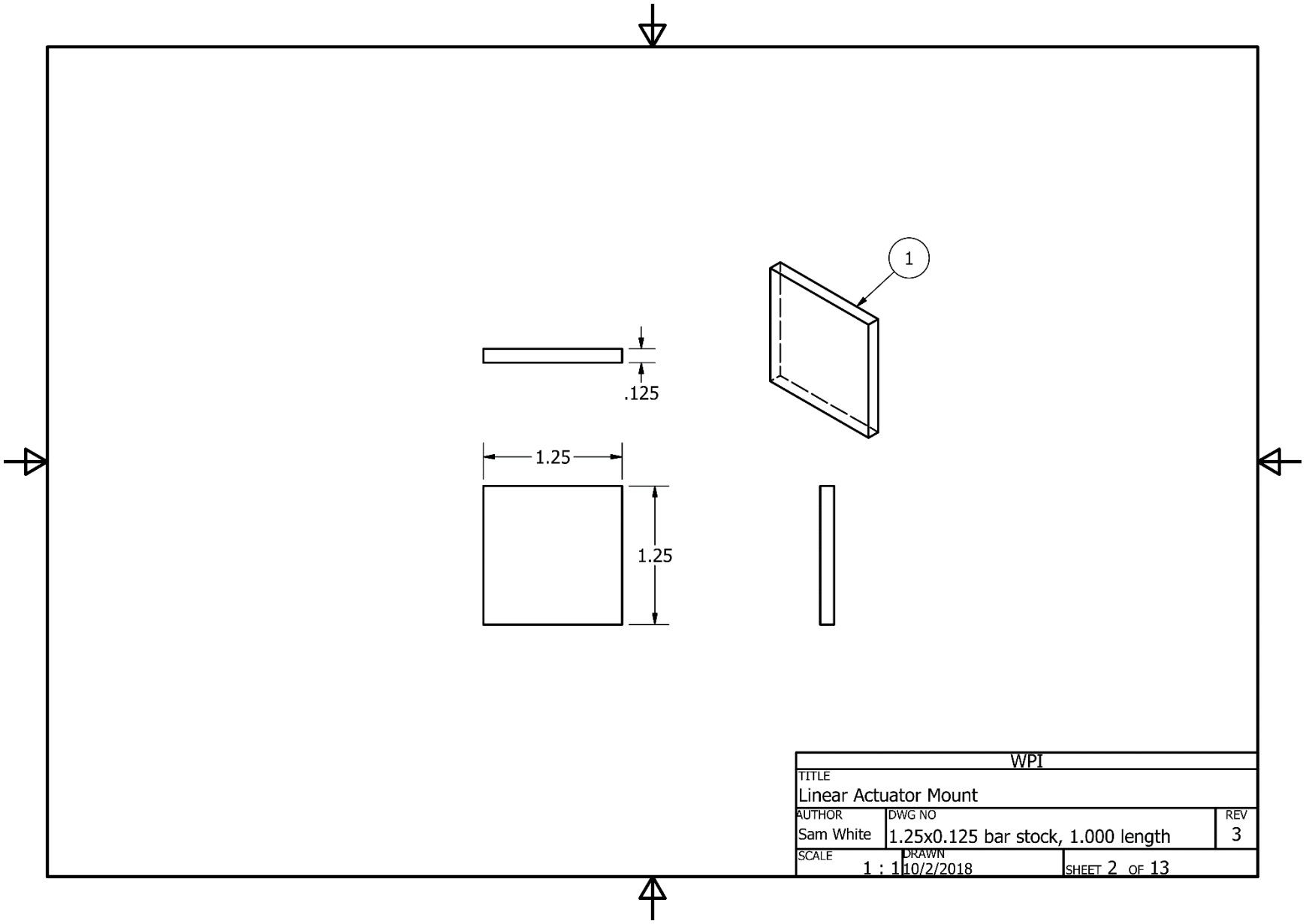
TITLE: Linear Actuator Mount

AUTHOR: Sam White | DWG NO: Body Mount Assembly | REV: 3

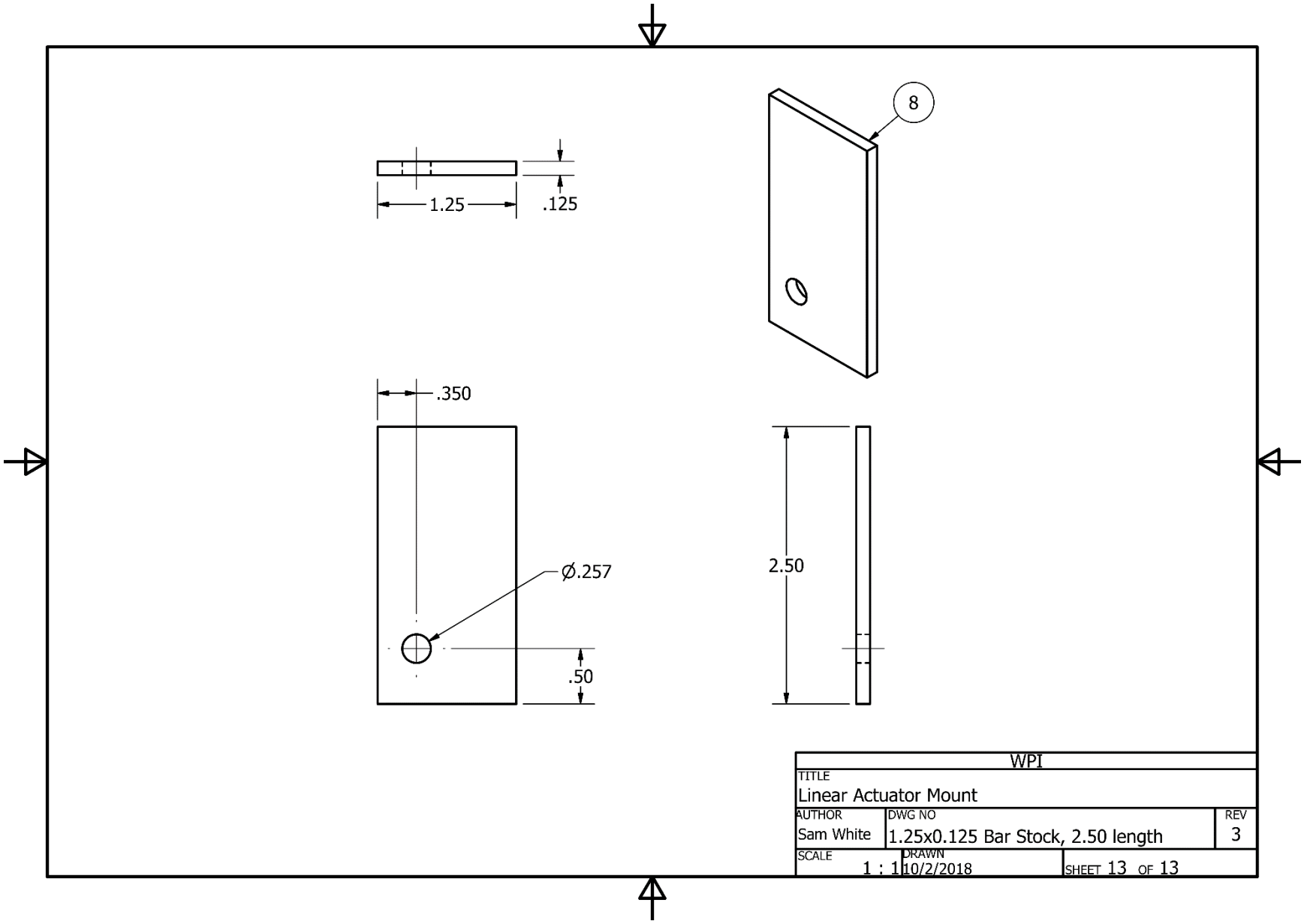
SCALE: 3/4 | DRAWN: 10/2/2018 | SHEET 5 OF 13

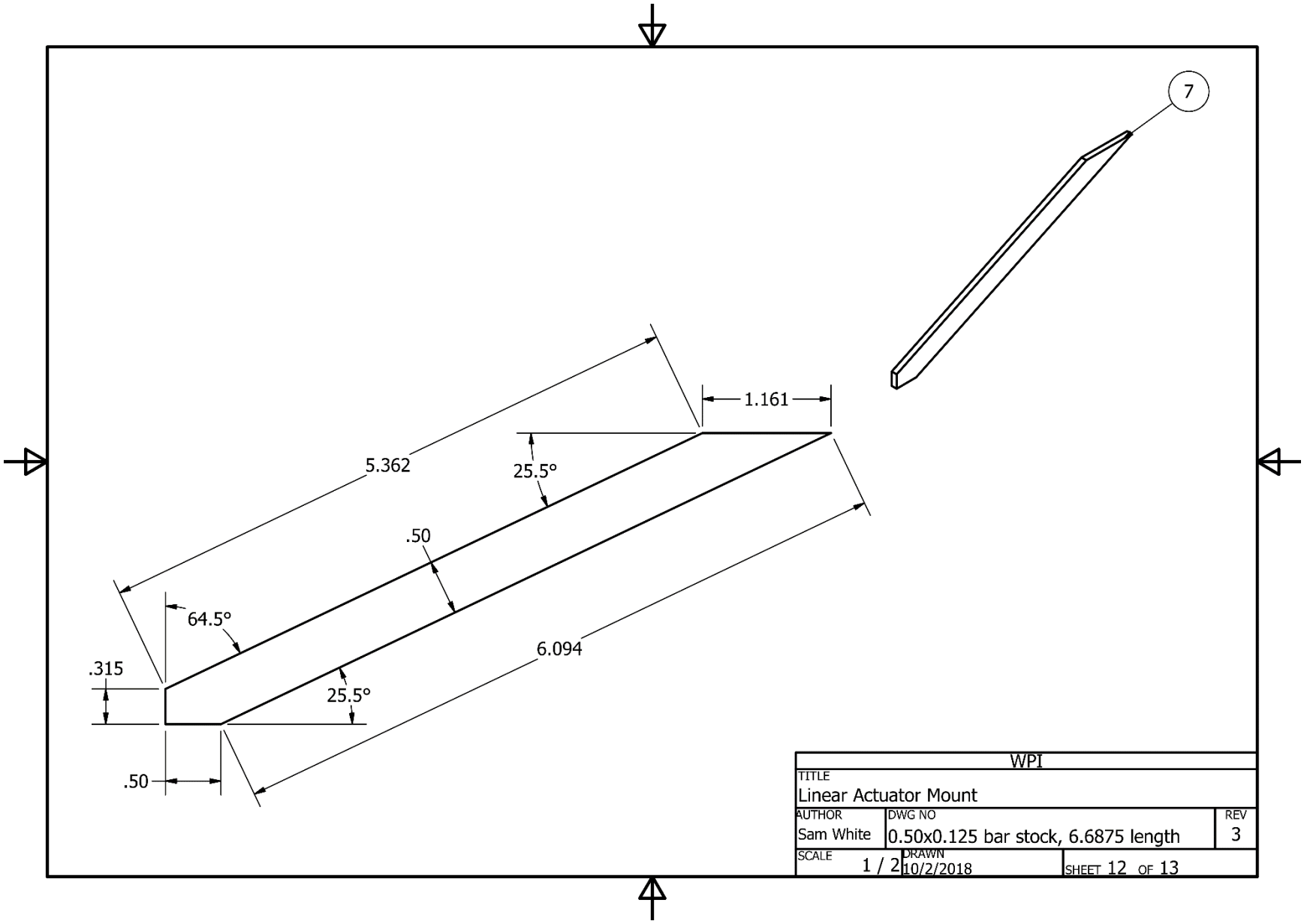

2.330
Ø.266
7.00
6.50
Ø.257
3.125
2.625
2.50
2.080
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
Body Mount Assembly
REV
3
SCALE
1 / 2
DRAWN
10/2/2018
SHEET 4 OF 13

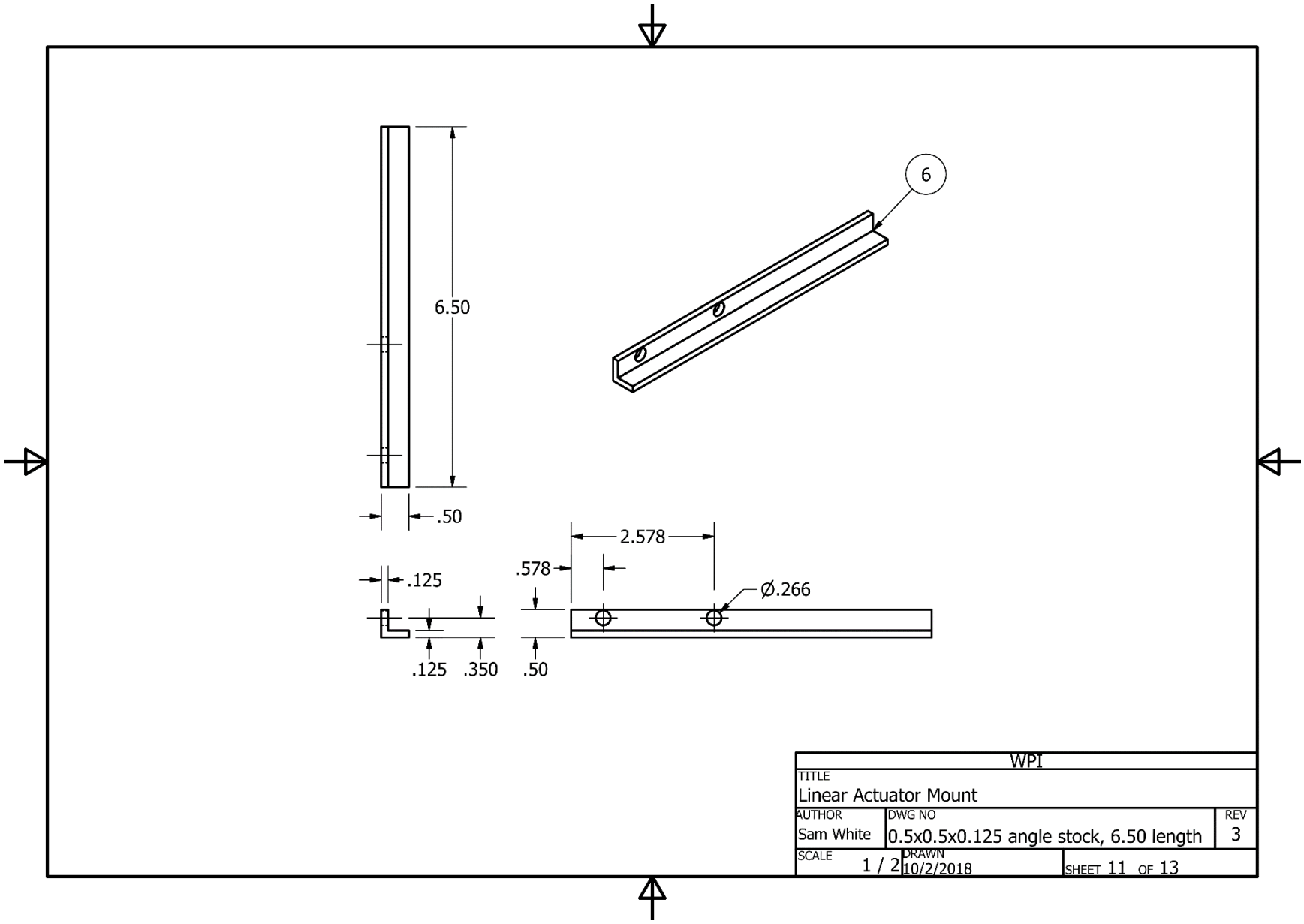

2
.125
1.50
.875
R.125
.625
1.25
.50
.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
1.25x0.125 bar stock, 1.500 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 3 OF 13

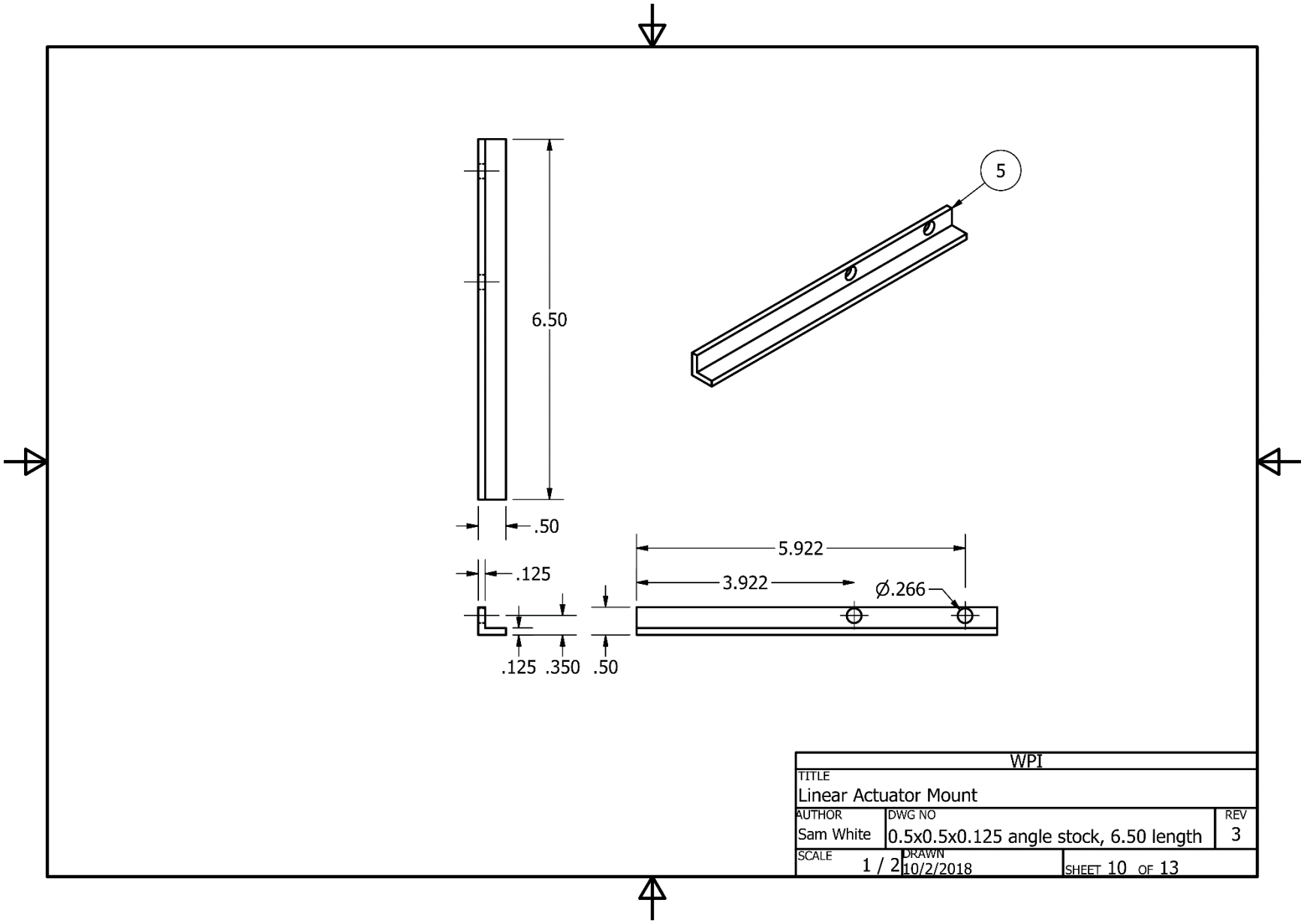


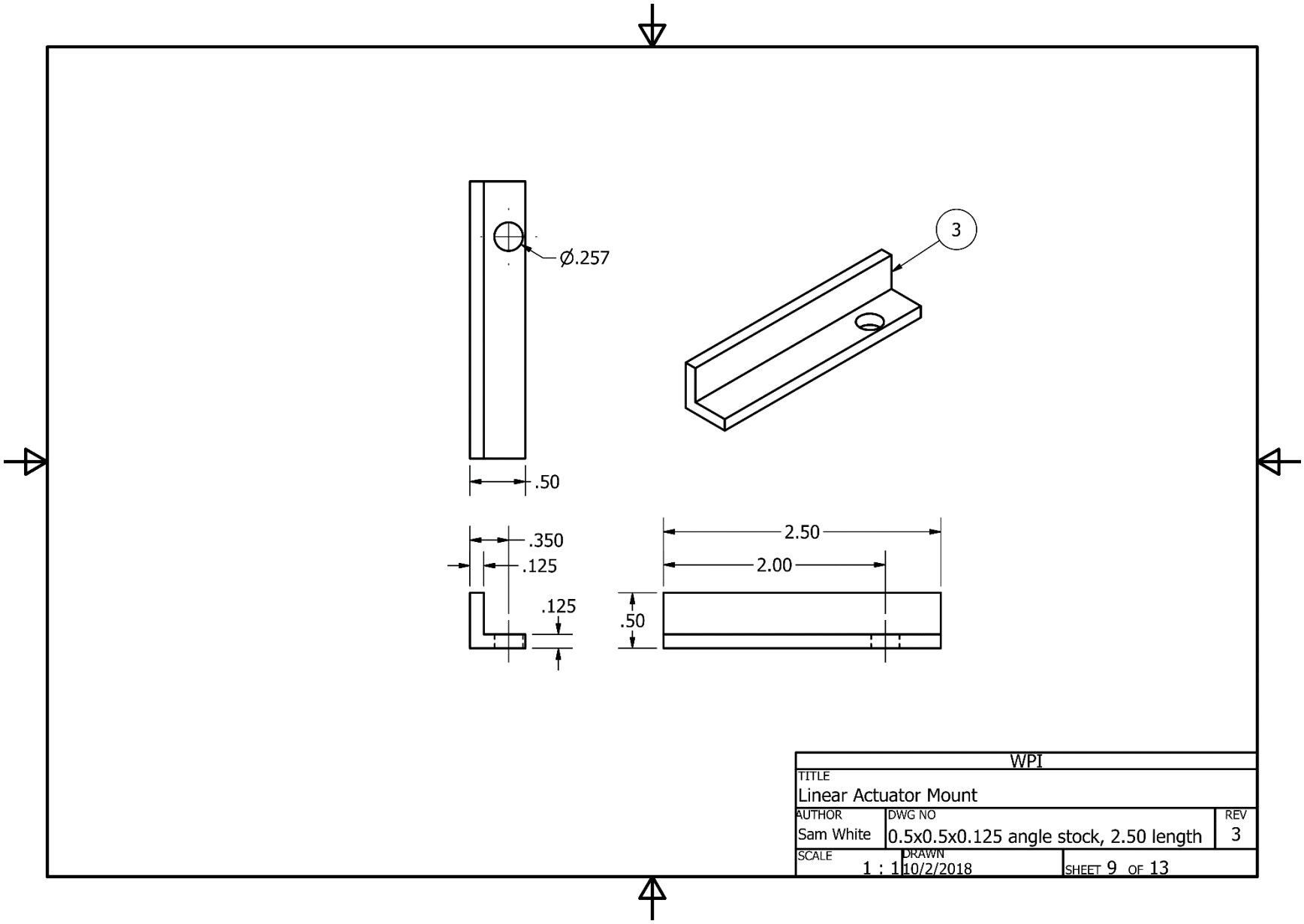
WPI		
TITLE: Linear Actuator Mount		
AUTHOR: Sam White	DWG NO: 1.25x0.125 bar stock, 1.000 length	REV: 3
SCALE: 1 : 1	DRAWN: 10/2/2018	SHEET 2 OF 13


1.25
.125
8
.350
Ø.257
.50
2.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
1.25x0.125 Bar Stock, 2.50 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 13 OF 13


7
1.161
5.362
25.5°
.50
64.5°
6.094
.315
25.5°
.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.50x0.125 bar stock, 6.6875 length
REV
3
SCALE
1 / 2
DRAWN
10/2/2018
SHEET 12 OF 13


6
6.50
.50
2.578
.578
.125
Ø.266
.125
.350
.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.5x0.5x0.125 angle stock, 6.50 length
REV
3
SCALE
1 / 2
DRAWN
10/2/2018
SHEET 11 OF 13


5
6.50
.50
.125
5.922
3.922
Ø.266
.125
.350
.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.5x0.5x0.125 angle stock, 6.50 length
REV
3
SCALE
1 / 2
DRAWN
10/2/2018
SHEET 10 OF 13


Ø.257
3
.50
.350
.125
.125
2.50
2.00
.50
WPI
TITLE
Linear Actuator Mount
AUTHOR
Sam White
DWG NO
0.5x0.5x0.125 angle stock, 2.50 length
REV
3
SCALE
1 : 1
DRAWN
10/2/2018
SHEET 9 OF 13

Appendix D: Brakes: Linear Actuator Mount Finite Element Analysis (FEA)

Stress Analysis Report

AUTODESK.

Analyzed File:	metal frame-v1.iam
Autodesk Inventor Version:	2018 (Build 220112000, 112)
Creation Date:	10/5/2018, 1:22 PM
Study Author:	Sam White
Summary:	simulates maximum force applied by linear actuator (100lbf)

Project Info (iProperties)

Summary

Author	Sam White

Project

Part Number	linear-actuator-to-body-mount-v2
Designer	Sam White

Status

Design Status	WorkInProgress

Physical

Mass	1.08246 lbmass
Area	73.9456 in^2
Volume	3.81686 in^3
Center of Gravity	x=5.15272 in y=-1.43777 in z=8.91299 in

Note: Physical values could be different from Physical values used by FEA reported below.

Static Analysis:1

General objective and settings:

Design Objective	Single Point
Study Type	Static Analysis
Last Modification Date	10/5/2018, 1:01 PM
Detect and Eliminate Rigid Body Modes	No
Separate Stresses Across Contact Surfaces	No
Motion Loads Analysis	No

Mesh settings:

Avg. Element Size (fraction of model diameter)	0.1
Min. Element Size (fraction of avg. size)	0.2
Grading Factor	1.5
Max. Turn Angle	60 deg
Create Curved Mesh Elements	No
Use part based measure for Assembly mesh	Yes

Material(s)

Name	Steel, Mild	
General	Mass Density	0.283599 lbmass/in^3
	Yield Strength	30022.8 psi

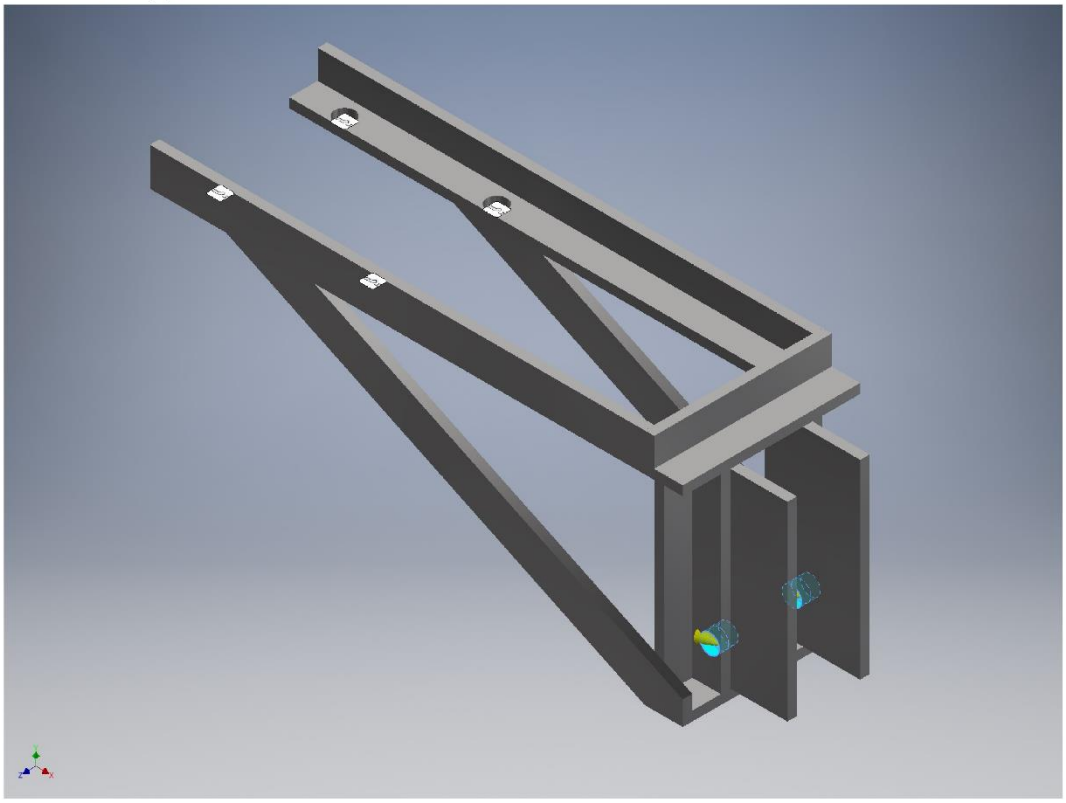
Stress	Ultimate Tensile Strength	50038 psi
	Young's Modulus	31908.3 ksi
	Poisson's Ratio	0.275 ul
	Shear Modulus	12513.1 ksi
Part Name(s)	0.5x0.5x0.125-angle-stock-2.330 0.5x0.5x0.125-angle-stock-2.500-hole-a 0.5x0.5x0.125-angle-stock-6.500-hole-a 0.5x0.5x0.125-angle-stock-2.500-hole-b 0.5x0.125-bar-stock-2.080 0.5x0.125-bar-stock-6.875-shaped-cut 0.5x0.125-bar-stock-6.875-shaped-cut 0.5x0.5x0.125-angle-stock-6.500-hole-b 1.25x0.125-bar-stock-1.250 1.25x0.125-bar-stock-1.250	

Operating conditions

Force:1

Load Type	Force
Magnitude	100.000 lbforce
Vector X	100.000 lbforce
Vector Y	0.000 lbforce
Vector Z	-0.000 lbforce

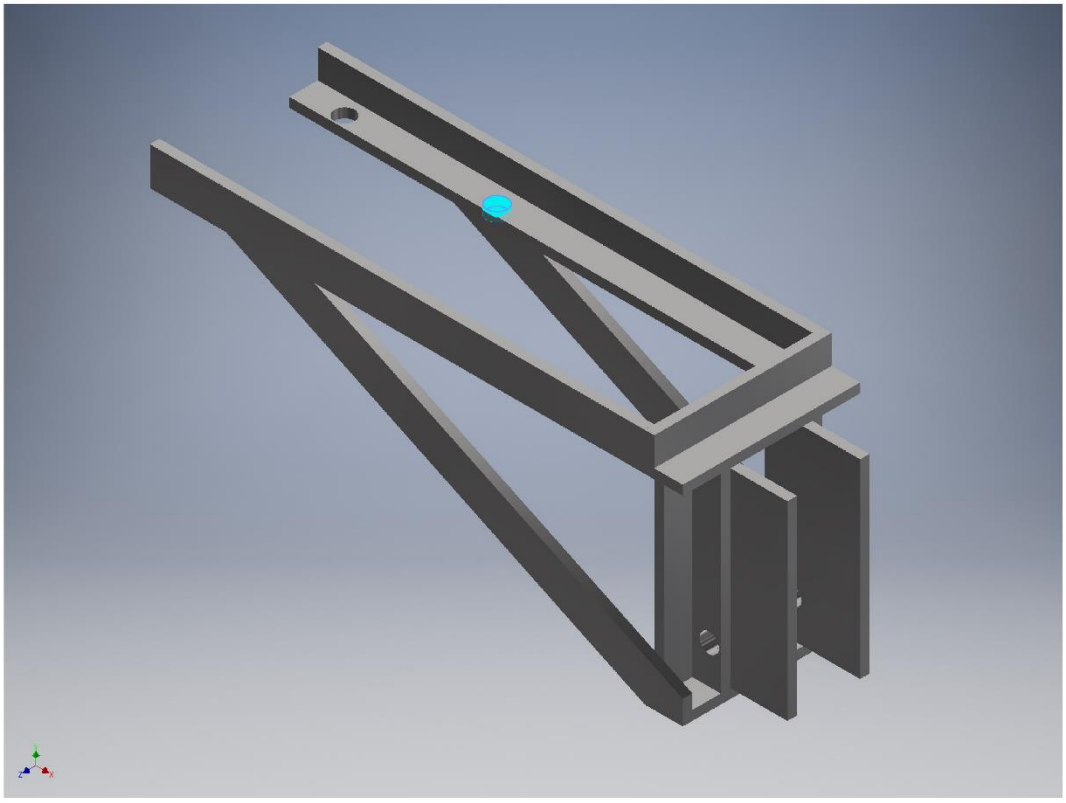
Selected Face(s)



Fixed Constraint:1

Constraint Type	Fixed Constraint

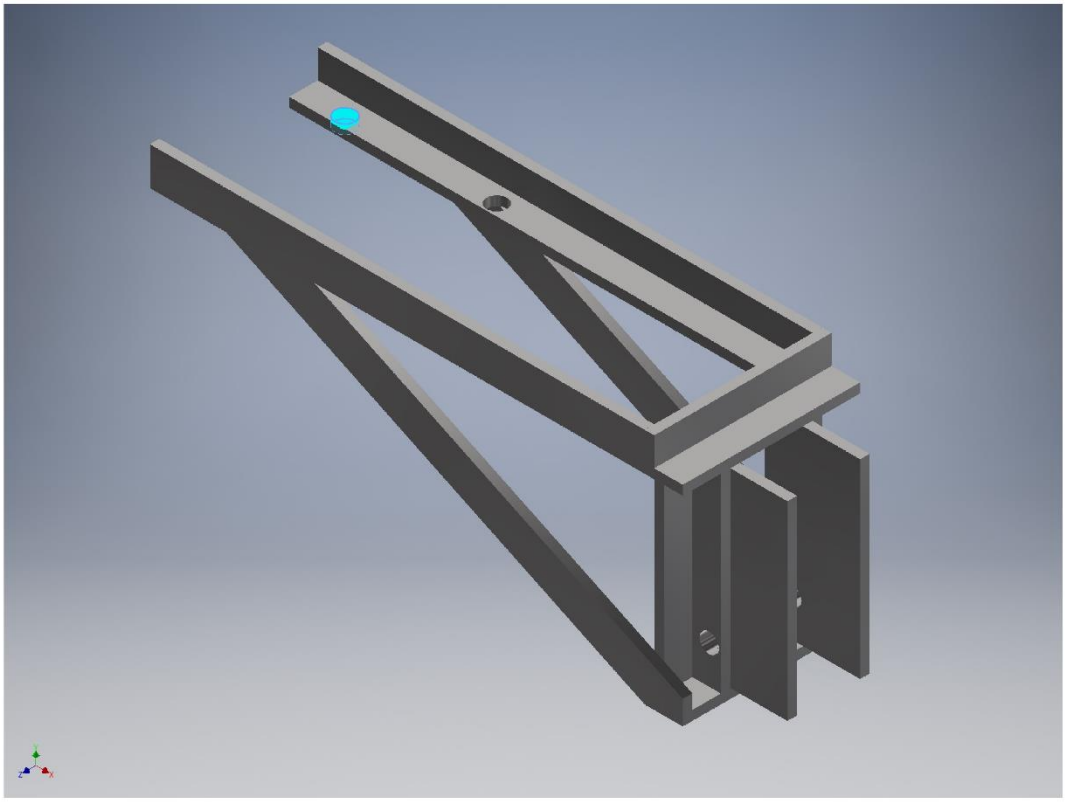
⊟ **Selected Face(s)**



⊟ **Fixed Constraint:2**

Constraint Type	Fixed Constraint

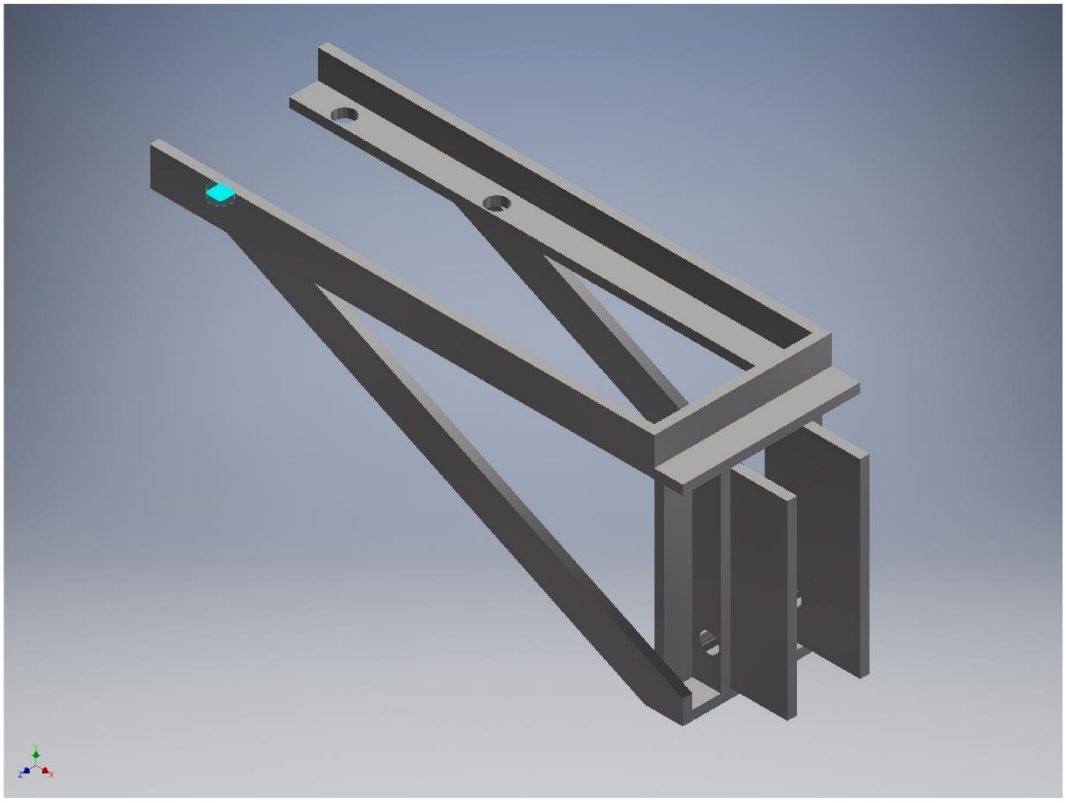
⊟ **Selected Face(s)**



⊟ **Fixed Constraint:3**

Constraint Type	Fixed Constraint

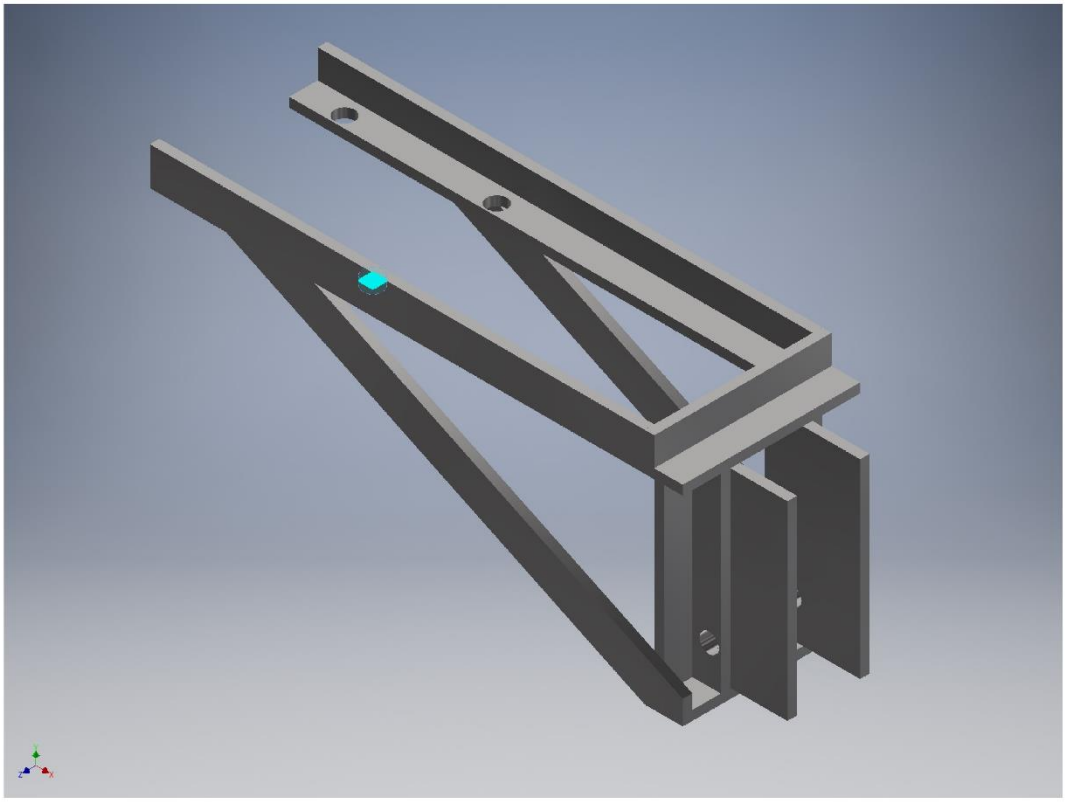
⊟ **Selected Face(s)**



⊟ **Fixed Constraint:4**

Constraint Type	Fixed Constraint

⊟ **Selected Face(s)**



⊟ Results

⊟ Reaction Force and Moment on Constraints

Constraint Name	Reaction Force		Reaction Moment	
	Magnitude	Component (X,Y,Z)	Magnitude	Component (X,Y,Z)
Fixed Constraint:1	91.7707 lbforce	-78.7685 lbforce	1.63847 lbforce ft	-0.873378 lbforce ft
		-45.9741 lbforce		-0.10566 lbforce ft
		10.1862 lbforce		-1.38226 lbforce ft
Fixed Constraint:2	54.3531 lbforce	28.767 lbforce	1.20194 lbforce ft	0.997954 lbforce ft
		45.9896 lbforce		-0.381765 lbforce ft
		-3.41617 lbforce		0.550447 lbforce ft
Fixed Constraint:3	54.3349 lbforce	28.6821 lbforce	1.20544 lbforce ft	-1.00078 lbforce ft
		46.0211 lbforce		0.378381 lbforce ft
		3.41814 lbforce		0.555296 lbforce ft
Fixed Constraint:4	91.7214 lbforce	-78.6806 lbforce	1.64079 lbforce ft	0.883585 lbforce ft
		-46.0288 lbforce		0.108939 lbforce ft
		-10.175 lbforce		-1.37826 lbforce ft

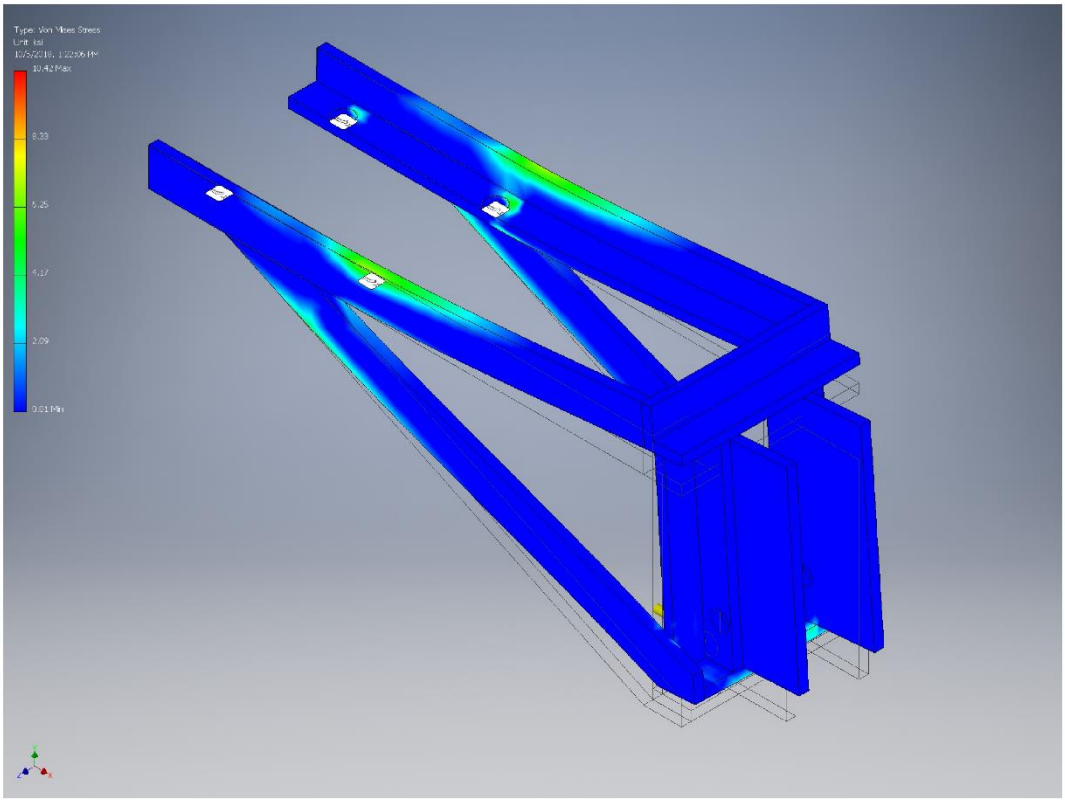
⊟ Result Summary

Name	Minimum	Maximum
Volume	3.81686 in^3	
Mass	1.08246 lbmass	
Von Mises Stress	0.00664597 ksi	10.4159 ksi
1st Principal Stress	-2.15229 ksi	14.7766 ksi
3rd Principal Stress	-9.35213 ksi	4.35114 ksi

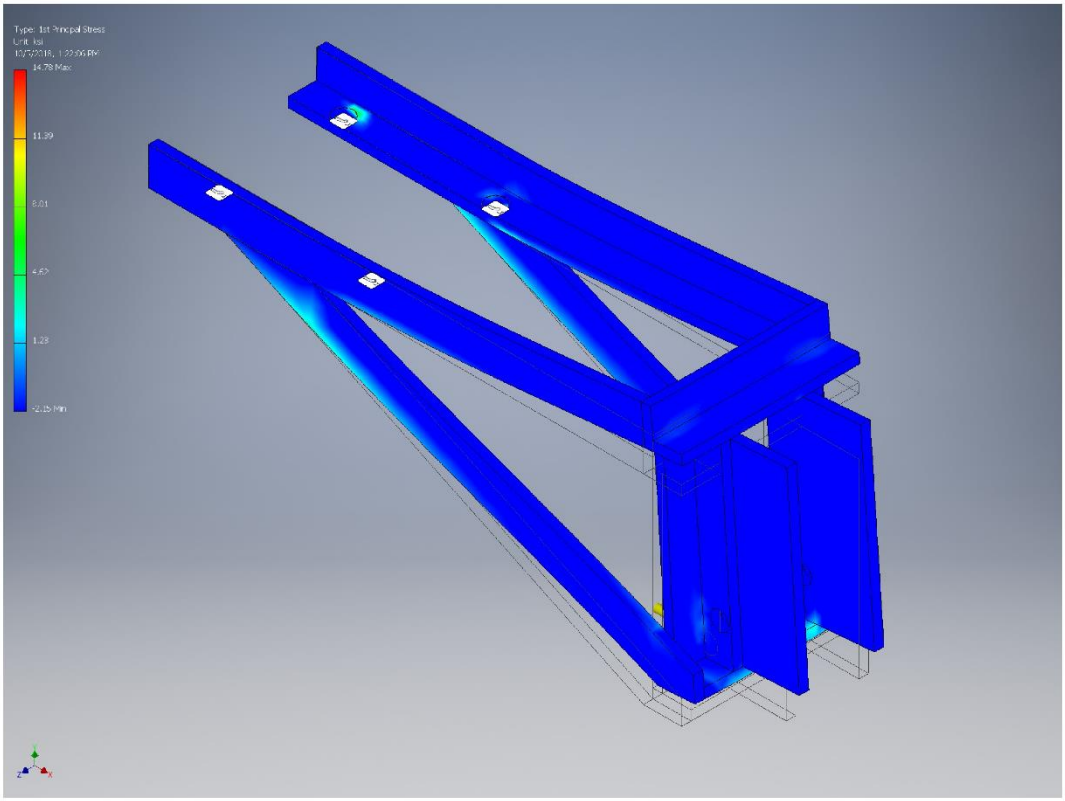
Displacement	0 in	0.00677766 in
Safety Factor	2.88239 ul	15 ul

Figures

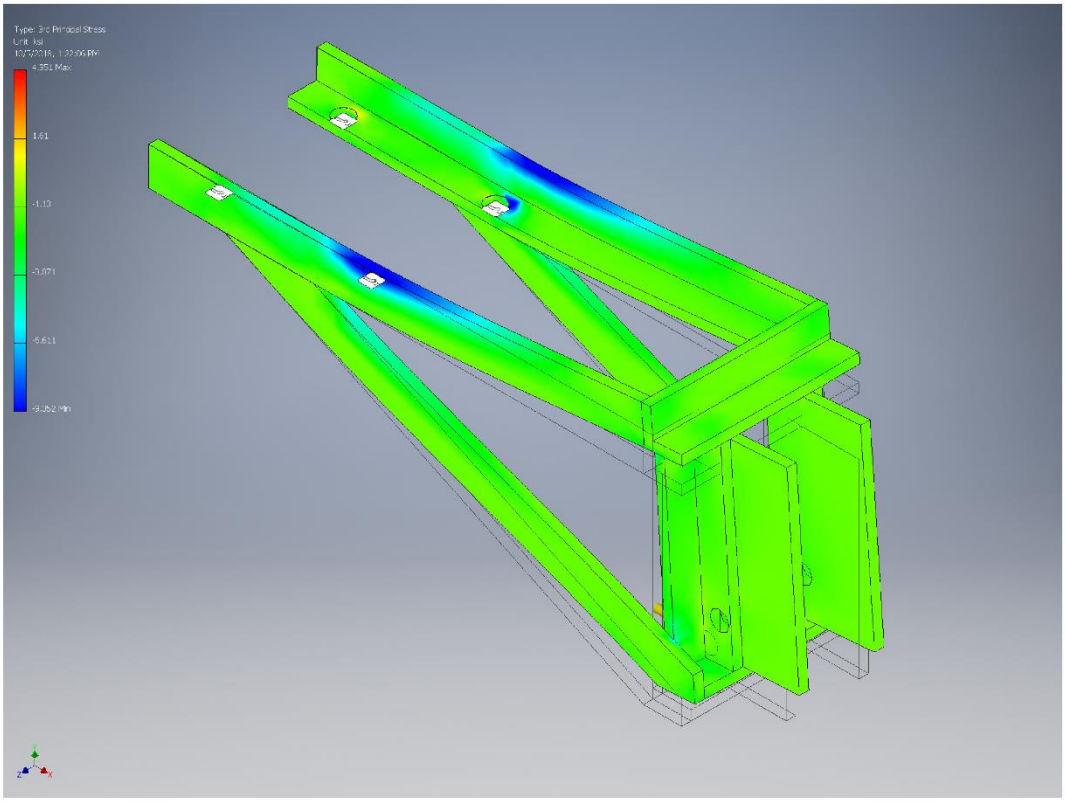
Von Mises Stress



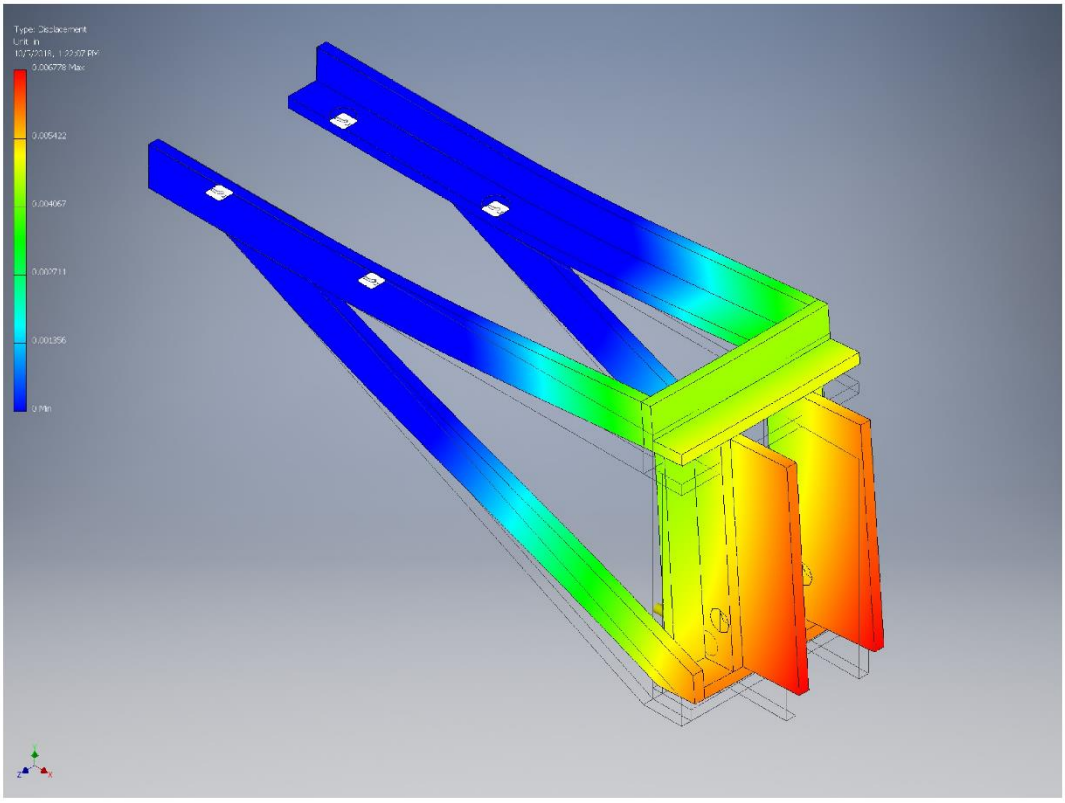
⊟ **1st Principal Stress**



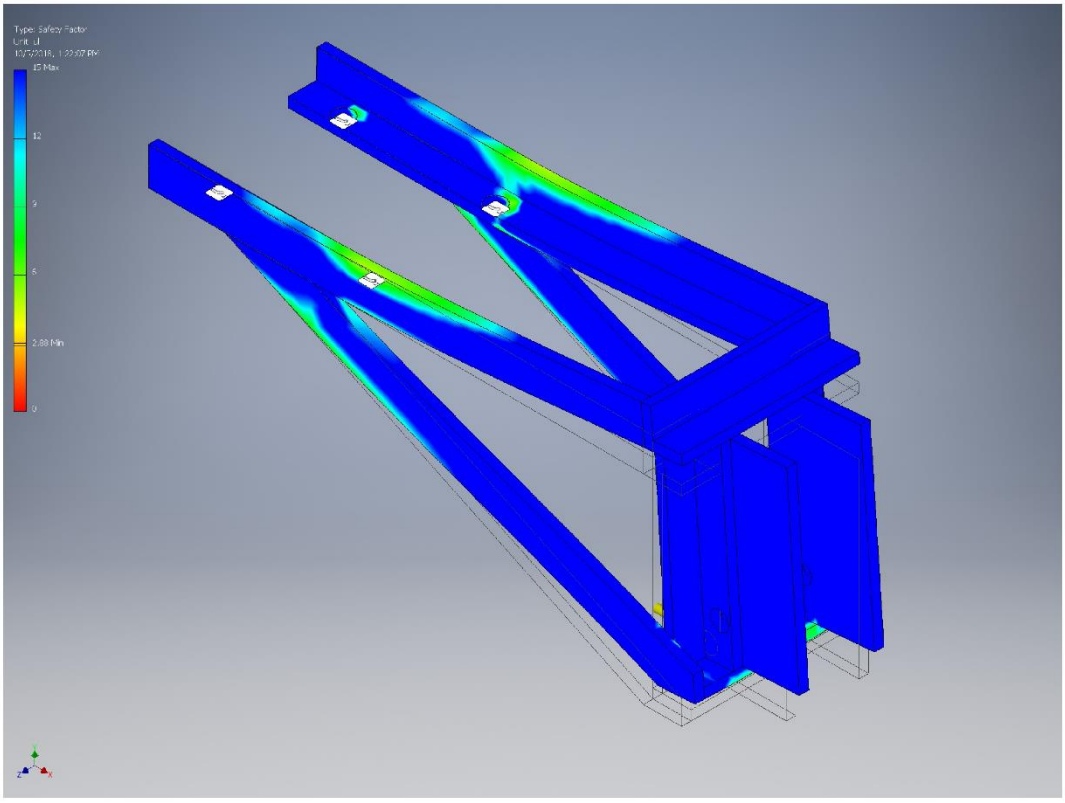
⊟ **3rd Principal Stress**



Displacement



⊟ **Safety Factor**



C:\Users\swhit\Documents\MQP KINISI\Brakes\metal frame-v1.iam

Appendix E: Master-Slave Arduino Communication (master.ino)

```
// Include the required Wire library for I2C<br>#include
#include <Wire.h>
//MAKE SURE TO RUN: rosrun rosserial_arduino serial_node.py _port:=/dev/ttyUSB1
_baud:=57600
#include <ros.h>
#include <std_msgs/String.h>
int x = 0;
//int ok = 0;
float maxSpeed = 7; //this can change
int maxDAC = 65;//145 is maximum speed of vehicle (DAC)
int minimumOnMC = 80;
float dataToSendF = 0;
int dataToSend = 0;
//Adafruit_BNO055 bno = Adafruit_BNO055();
float offset = 21.85;
float gain = 1.11;
bool okToGo = false;
ros::NodeHandle  nh;
void readSerial (const std_msgs::String& str_msg);
ros::Subscriber<std_msgs::String> arduino_sub("motion_command", &readSerial);

void setup() {
  // Start the I2C Bus as Master
  //Serial.begin(115200);
  Serial.begin(57600);
  //Sets the timeout for the serial communication
  Serial.setTimeout(50);
  //begins I2C communication
  Wire.begin();
  pinMode(13, OUTPUT);
  pinMode(8, OUTPUT);
  digitalWrite(8, HIGH);
  //sendCommand(8,0);
//  byte stall = byte(85 + getAvgGrade() - offset);
//  delay(2000);
  //sendCommand(8, speedToDAC(0));
  //delay(1000);
  nh.initNode();
  nh.subscribe(arduino_sub);
}

void loop() {
//  listen for response

//sendCommand(8, speedToDAC(2));
//delay(7000);
//sendCommand(10, 138);
//delay(2000);
//sendCommand(8, speedToDAC(3));
//delay(500);
//sendCommand(10, 108);
//delay(3000);
//hasntRun = false;
```

```
//if(okToGo)
//{
////digitalWrite(13, HIGH);
////  if (Serial.available())
////  {
////   readSerial();
////  }
//digitalWrite(13, HIGH);
//}
//else
//{
//  if(getAvgGrade() < 30)
//  okToGo = true;
//}

nh.spinOnce();
}
void readSerial (const std_msgs::String& str_msg)
{
  //digitalWrite(13, HIGH);
  //read the serial message
  //array allocated for entire serial command
  String cmd;
  //allocate two more arrays, one for actuator data, another for slaveID
  //char data [6];
  char id [3];
  //Read data from serial port
  //cmd = Serial.readString();
  cmd = str_msg.data;
  char cmdChar[cmd.length() + 1];
  //populate the character array cmdChar with contents from cmd
  cmd.toCharArray(cmdChar, (cmd.length() + 1));
  cmdChar[cmd.length()] = '\0';
  //parse the ID of the slave and the data for the slave
  id [0] = cmdChar[0];
  id [1] = cmdChar[1];
  id [2] =  '\0'; //null terminator

  char data[cmd.length() - 2];

  for (int i = 0; i < cmd.length() - 3 ; i++)
  {
    data[i] = cmdChar[i + 3]; //put the data into the data array
  }

  data[cmd.length() - 3] = '\0'; //adda a null terminator

  //convert the two arrays into ints
  int idToSend = atoi(id);
  if (idToSend == 8)
     dataToSendF = atof(data);
  else
     dataToSend = atoi(data);

  if(String(data).equals("stop"))
  {
  dataToSend = 0;
  }
```

```
//  if (strcmp("stop", data) == 1)
//    dataToSend = 0;
  //send the command to the right slave using sendCommand
  //if it's the throttle, convert the speed in m/s to DAC
  if (idToSend == 8)
    sendCommand (idToSend, speedToDAC(dataToSendF));
  else
    sendCommand (idToSend, dataToSend);
  // end of  readSerial
}
float getAvgGrade()
{
  float avg = 0;
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  for (int i = 0; i < 40; i++)
  {
    euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
    avg += euler.z() + 90;//(accel.z() - accAvg);
  }
  return (float)(avg / 40.0);
}

byte speedToDAC(float new_speed)
{
  //take in a speed (m/s)
  float conversion = (float) (maxDAC / maxSpeed);
  //output a voltage
  return (byte) ((new_speed * conversion) + minimumOnMC);// + (getAvgGrade()*gain) -
offset);
  //check if it gets below 80, this means we are going downhill, we should start
applying the brakes
}

void sendCommand(int deviceID, byte data)
{
  //open communication, send the data, close communication
  Wire.beginTransmission(deviceID);
  Wire.write(data);              // sends x
  Wire.endTransmission(); // stop transmitting
}



//this function gets the response from the slave if necessary.
//We will need this for the snapshot. Not yet implemented
int getResponse(int deviceID, int bytes)
{
  int ok = 0;
  Wire.requestFrom(deviceID, bytes);    // request 4 bytes from slave device #9
  if (Wire.available()) { // slave may send less than requested
    ok = Wire.read(); // receive a byte as character
  }

  return ok;
}
```

Appendix F: Acceleration-Slave Arduino Code (slaveAccel.ino)

```
#include <Wire.h>
//B is negative, A is positive
// Include the required Wire library for I2C<br>#include <Wire.h>
int LED = 13;
int x = 0;
int reading = 0;
int percent = 0;


void setup() {
  // Define the LED pin as Output
  pinMode (LED, OUTPUT);
  // Start the I2C Bus as Slave on address 8
  Wire.begin(8);
  // Attach a function to trigger when something is received.
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
  pinMode(5, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(LED, OUTPUT);
  digitalWrite(7, HIGH);
  analogWrite(9, 87);
}
void receiveEvent(int bytes) {
  int voltage = Wire.read();    // read the voltage from the master (DAC PWM value)
  if(voltage == 80)
  digitalWrite(LED, HIGH);
  if(voltage == 0)
  {
    digitalWrite(7, LOW);
  }
  analogWrite(9, voltage); //output that voltage on pin 5
}
void requestEvent()
{
  //here the snapshot will go
}
void loop() {
}
```

Appendix G: Braking-Slave Arduino Code (slaveLinAct.ino)

```
//#include <LinActuator.h>
#include <Wire.h>
//white is ground on pot linear actuator
//B is negative, A is positive on motorcontroller

// Include the required Wire library for I2C<br>#include <Wire.h>
int LED = 13;
int x = 0;
int reading = 0;
int percent = 0;
bool newCommand = false;
int tolerance = 1;
int brakePot = A2;
int dir = 4;
int PWM = 9;
int speed = 255;
void linActControl(int percent);
void setup() {
  // Define the LED pin as Output
  pinMode (LED, OUTPUT);
  // Start the I2C Bus as Slave on address 9
  Wire.begin(9);
  // Attach a function to trigger when something is received.
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
  pinMode(9, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(A2, INPUT);
  Serial.begin(9600);
}
void receiveEvent(int bytes) {
  percent = Wire.read();    // data from master
  //call linActControl function to go to percent indicated
  newCommand = true;
}
void requestEvent()
{
  //interim solution for heartbeat. Will Need to change for snapshot
  reading = analogRead(A2);
  reading = (int)((reading - 26) / (9.7));
  if (abs(percent - reading) <= 2)
    Wire.write(1);

  else
    Wire.write(0);
}
void loop() {
  linActControl(percent);
}
void linActControl(int percent)
{
  if (percent == 0)
  {
    analogWrite(PWM, 0);
```

```
    return;
  }
  int potValue = analogRead(A2);
  potValue = (int)((potValue - 26) / (9.7));
  int error = potValue - percent;
  while (abs(error) > tolerance && (newCommand == false))
  {
    if (error < 0)
      digitalWrite(dir, HIGH);
    else
      digitalWrite(dir, LOW);
    analogWrite(PWM, speed);
    potValue = analogRead(brakePot);
    potValue = (int)((potValue - 26) / (9.7));
    error = potValue - percent;
  }
  //WHY DO WE NEED THIS? DOESN'T LISTEN OTHERWISE BY WHY?????
  if (newCommand == true)
  {
    newCommand = false;
  }
  analogWrite(PWM, 0);
}
```

Appendix H: Steering-Slave Arduino Code (slaveSteering.ino)

```
//#include <Steering.h>
#include <Wire.h>
//A is right, B is left
//B is negative, A is positive
//Left is 0, Right is 1023
//A0:D A1:C A2:B A3:A
// Include the required Wire library for I2C<br>#include <Wire.h>
int LED = 13;
int x = 0;
int reading = 0;
int pos = 0;
int PWM = 5;
int led_on = 0;
bool newCommand = false;
int tolerance = 25;
int dir = 4;
int speed = 0;
int kp = 10;
int error; //Set once pot is fixed
//int new_pos = 0;
//int avgPot1 = 0;
//int avgPot2 = 0;
//extern int prev1G;
//extern int prev2G;
void steeringControl(int pos);
void setup() {
  // Define the LED pin as Output
  pinMode (LED, OUTPUT);
  // Start the I2C Bus as Slave on address 10
  Wire.begin(10);
  // Attach a function to trigger when something is received.
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
  //pinMode(9, OUTPUT);
  pinMode(PWM, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(A1, INPUT);
  //  pinMode(A2, INPUT);
  //  pinMode(A3, INPUT);
  //  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  //  int avgPot1 = ((analogRead(A2) + analogRead(A1))/2);
  //  prev1G = avgPot1;
  //  prev2G = avgPot2;
  //int avgPot2 = ((analogRead(A3) + analogRead(A0))/2);
  Serial.begin(9600);
}
void receiveEvent(int bytes) {
  pos = Wire.read();    // read one character from the I2C
  pos = (int)((pos * 2.156) + 300);
  newCommand = true;
}
void requestEvent()
```

```
{
  // I suspect this will have some logic here for the snapshot, nothing here yet
  //will need to change for steering
  //  reading = analogRead(A2);
  //  reading = (int)((reading - 26)/(9.7));
  //  if (abs(percent - reading) <= 2)
  //    Wire.write(1);
  //
  //  else
  //    Wire.write(0);
}
void loop() {
  steeringControl((int)(pos));
}
void steeringControl(int pos)
{ if (pos == 0)
  {
    analogWrite(PWM, 0);
    return;
  }
  int avgPot = analogRead(A2);
  error = avgPot - pos;
  //newCommand = false;
  while ((abs(error) > tolerance) && (newCommand == false)) //while a new command
hasn't come in yet
  {
    if (error < 0)
    {
      digitalWrite(dir, HIGH);
    }
    else
    {
      digitalWrite(dir, LOW);
    }
    speed = kp * (abs(error));
    if (speed > 255)
      speed = 255;
    analogWrite(PWM, speed);
    int avgPot = analogRead(A2);
    error = avgPot - pos;
  }

  if (newCommand == true)
  {
    analogWrite(PWM, 0);
    delay(30); //delay needed for motor control to not throw a         hissy-fit.
Capacitor?
    newCommand = false;
  }
  analogWrite(PWM, 0);
}
```