

# Autonomous Scale Car Using Computer Vision and Neural Networks

Major Qualifying Project

Submitted to the faculty  
of  
WORCESTER POLYTECHNIC INSTITUTE

By

Mitchell Curbelo (ECE, ME)

Ryan Darnley (ECE, ME)

Ava Karet (ME)

Krishna Madhurkar (RBE, ECE)

Dylan McKillip (RBE)

Ethan Schutzman (CS)

Christian Scillitoe (CS)

## Advisors

Prof. Kaveh Pahlavan (ECE, CS)

Prof. Pradeep Radhakrishnan (ME, RBE)

*This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the project program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>*

## **Abstract**

This project explores the feasibility of creating an autonomous 1:10 scale car capable of racing against RC cars on any indoor track. Prior to conducting autonomous testing, a custom made scale car equipped was designed and manufactured. Comprised mainly of 3D printed parts, the car used rear-wheel drive with independent and passive front and rear suspensions. Additionally, tachometers, strain gages, temperature sensors, and an inertial measurement unit were integrated to determine vehicle performance. The data from all sensors were displayed in real time on a webpage. For autonomous navigation, the car leveraged Artificial Neural Networks to produce optimal driving outputs. Taking grayscale pixel input from a single, front-facing camera, the car down-sampled and masked the input data to locate brown walls. Without relying on mapping or localization, the car collected high quantities of training data to navigate variable track conditions. Ultimately, the training produced a car that could perform partial laps under its own control.

## **Acknowledgments**

First, we would like to thank our two advisors: Professor Pradeep Radhakrishnan and Professor Kaveh Pahlavan. Without their support and direction, we would not have been able to achieve as much as we did. We would also like to thank Worcester Polytechnic Institute, most importantly all of the faculty and staff who have helped us in all forms of planning, space reservation, and advice. Among them, Ms. Payton Wilkins, Ms. Barbara Furhman, Professor Cagdas D. Onal, and Mr. Peter Hefti were very influential at providing the resources and structure necessary to work on this project.

# Executive Summary

## Introduction

Worcester Polytechnic Institute (WPI) offers a course titled ME4320, Advanced Engineering Design. Within the course, students are tasked with the construction of a remote control 1:10 scale car with a custom gearbox and steering assembly. After designing, analyzing, and manufacturing the vehicle, the class then culminates in a race. Here, the cars are raced around a wooden track and ultimately, the winners receive significant grade boosts.

While the course is very successful and regularly produces high quality cars, there are three fundamental aspects of the course which could be improved. First, in light of the 7-week time constraints of the course, the students are often unable to pursue more elaborate car designs. Second, the students, although required to conduct mechanical analysis, are unable to collect sensory data to substantiate the performance of their vehicles. Finally, the race at the end of the class is often won by the best driver rather than the best car.

Based on the three aforementioned flaws, this project looks to improve further the quality of the Advanced Engineering Design course. In order to create a more complicated car design, this project first explores the feasibility of creating a custom, 1:10 scale car equipped with passive front and rear suspension. Next, in order to provide sensory data, this project examines the prospect of creating a modular sensor package which can be readily equipped to any of the produced scale cars. Finally, in order to eliminate driver bias, this project tests the implementation of a modular Artificial Intelligence (AI) package which allows for autonomous navigation of the vehicles.

## Background

A radio controlled (RC) car is a car that is operated using a receiver-transmitter pair. While the RC car is typically much smaller than an actual vehicle (often 1:10 scale), it possesses many of the same fundamental driving features as does a normal vehicle. Among them, the RC car has suspension, drivetrain, chassis and steering mechanisms. As RC cars are normally cheap and used for hobbyist applications, these mechanisms, however, are typically created in different manners.

Seen in Figure ES-1 (reproduced as is from [1]) is an RC car. Of particular importance with the car's mechanical design are the circled elements such as the front and rear suspension, chassis, steering assembly, and drivetrain. Conventionally, the front suspension of a scale vehicle is independent. Furthermore, the rear suspension is typically dependent utilizing a four-bar mechanism. The chassis, normally a single plate, connects the front and rear suspensions while also holding all appropriate electronics including the battery and motor. Meanwhile, the steering assembly is prone to vary across different RC models. Among the variations, a parallelogram linkage with a servo motor typically is used to control the rotation of the front wheels. Finally, the drivetrain consists of a brushed or brushless electric motor connected to a gearbox transmission. The output of the gearbox is then translated to an output shaft which is fitted with universal joints to allow for rear suspension articulation. This energy is

ultimately transferred to the rear differential, which then splits and rotates the energy to the wheels.

In conjunction with the mechanical design, the RC car is normally equipped with the following electrical components: battery, servo, ESC, motor, and receiver. Either Lithium Polymer or Nickel Metal Hydride, RC cars typically use batteries to power their equipment. For steering, RC cars also utilize servo motors which connect to the steering assemblies. In order to accelerate and decelerate the vehicle, a motor and compatible ESC are also fitted to the car. Typically brushed or brushless, this combination is comparable to the internal combustion engine of a full-sized vehicle. Finally, the receiver communicates with the controller used by the operator.

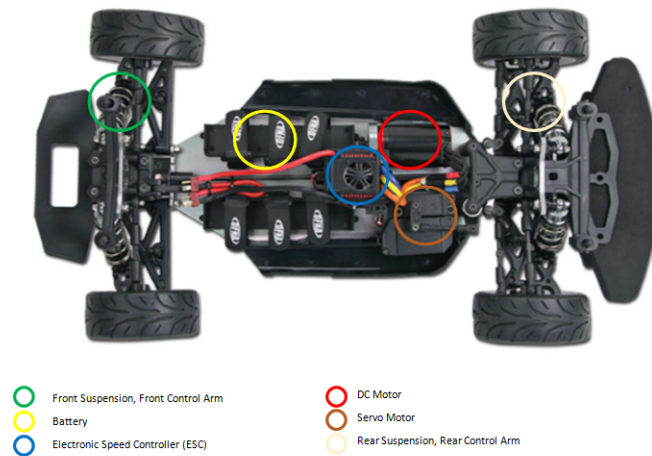


Figure ES-1: Labeled top view of the GT-2Ve RC Car. Reproduced as is from [1].

In order to make the RC car autonomous, there are two fundamental additions to the electrical and mechanical structure: hardware and software. The hardware comes in the form of sensors and computing, while the software involves some form of artificial intelligence. Among the many potential sensor systems, some include RADAR, LiDAR, and cameras. With RADAR and LiDAR, both systems transmit and receive a signal (electromagnetic wave and laser pulse), which contains information pertaining to the sensor's immediate surroundings. This influx of rapidly produced data can then be arranged into point clouds to create a representation of the surrounding environment. Meanwhile, cameras, much like human eyes, do not produce a transformation of reality. Instead, cameras capture the actual representation of the scene using a matrix of pixels with appropriate RGB (Red, Green, Blue) values.

In order to handle the influx of data collected by the vehicle's sensors, autonomous scale cars must also utilize an onboard computer. While the specific computational power is dependent on the application, the RC car must be able to integrate hardware such as a Raspberry Pi [2] or a micro-computer into its mechanical and electrical structure. This computer must then function as the brain of the RC vehicle when appropriately programmed. Ultimately, among the many subdisciplines of AI, the car must be capable of accepting sensory data, processing said data, and then producing an optimal vehicular response based on the output.

Over the previous decade, an abundance of work has been documented in the autonomous car spectrum. Seen most frequently with automotive giants, car manufacturers like Tesla harness 8 external cameras, 12 ultrasonic sensors, and a radar to collect the sensory data necessary for autonomous navigation [3]. While the race for Level 5 autonomy is most common among major enterprises, the quest has also expanded into the hobbyist and academic sectors. Seen in the hobbyist product, Donkey Car [4], cameras and open-source neural networks have provided cheap and attainable means to creating simple autonomous navigation vehicles. Ultimately, Donkey Car, Tesla, and all other autonomous solutions generate data to form an AI capable of producing autonomous navigation controls. While open-source platforms like Donkey Car rival the concept, however, no system appears to be fully modular for scale vehicles. Consequently, the creation of a modular AI package for scale vehicles will be a fundamental objective examined in this report.

### Car Design

Prior to conducting any autonomous testing, a scale vehicle was created. As illustrated in Figure ES-2, the vehicle was 16.38 inches long by 8.85 inches wide by 4.71 inches tall and 6.3 pounds. For the front suspension, the car followed a typical RC car convention and used a modified independent single wishbone. Additionally for the rear suspension, the car used a similarly modified independent wishbone. Using a servo controlling mechanism, the steering assembly was based on the parallelogram design. The drivetrain of the vehicle consisted of a brushless DC motor connected to a gearbox transmission with a 1:9 step down. The rotary output from the gearbox was then transmitted to the rear differential which had a 1:3 step ratio. This power was then delivered to the two rear wheels of the car. Finally, as the car had to hold many electronics and sensors, the chassis of the car was appropriately designed to carry many mounting features. Ultimately, while Figure ES-2 illustrates the final car design, the vehicle underwent countless iterations and modifications. Not only was the car design dependent on mechanical performance, the car also had to satisfy the many requirements of the AI.



Figure ES-2: Final iteration of 1:10 scale car.

In order to monitor the physical performance of the car while driving, the car was also fitted with a modular sensor dashboard. Capable of producing real-time data readouts, the car was equipped with a tachometer, temperature sensors, and an inertial measurement unit (IMU). To produce real-time updates, these sensors were connected to an Arduino Mega which relayed the sensory data to a Raspberry Pi. The Raspberry Pi was then able to send the sensory data to a webpage, producing graphics similar to the ones depicted in Figure ES-3. While data flow in the dashboard can be seen in Figure ES-4.

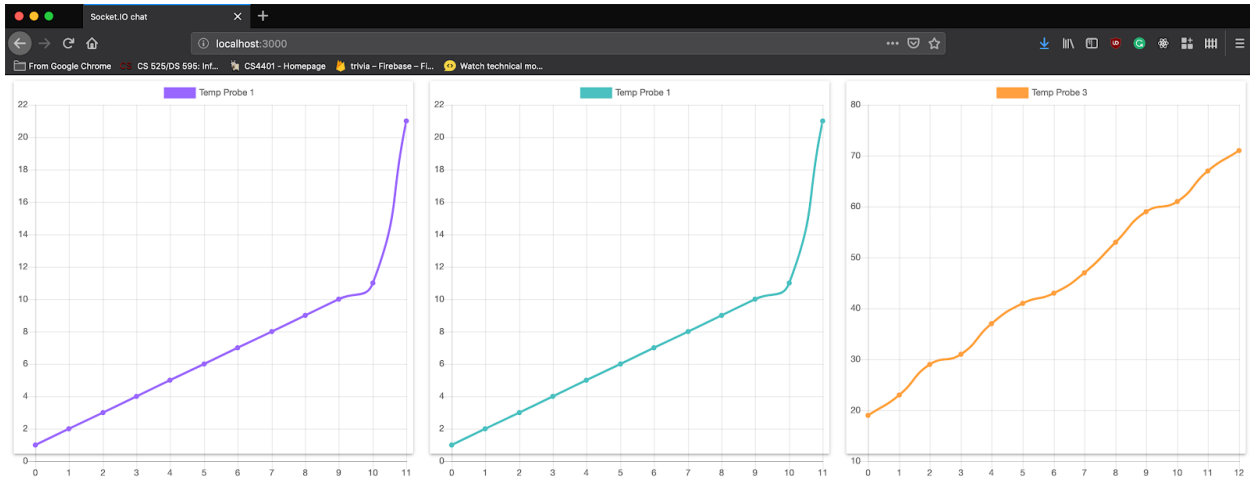


Figure ES-3: Example of graphs on webpage. These graphics were produced using mock data.

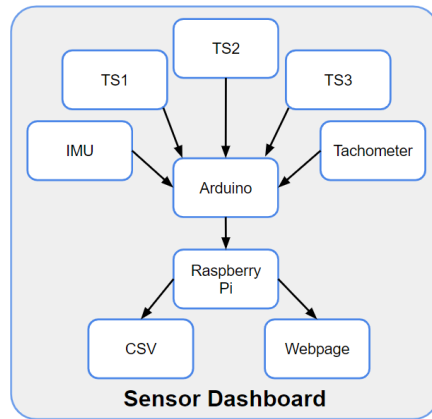


Figure ES-4: Standard dashboard layout for sensor module.

For autonomy, the car was fitted with a single, front-facing camera for sensory data collection. Using a Raspberry Pi for computation, the Red-Green-Blue (RGB) data was then downsampled and masked into a 640x480x1 matrix of grayscale valued pixels. This matrix was then fed to a convolutional neural network which produced appropriate driving instructions. These instructions were ultimately sent to an Arduino Mega which appropriately controlled the servo and brushless motor of the car.

## Results

Having produced a car and an autonomous sensor package, the project culminated in the performance of its autonomous navigation module. Tested on numerous tracks and even more convolutional neural network models, the car ultimately produced variable results. For its best performance, the car autonomously navigated approximately 50% of a track (Figure ES-5). Figure ES-5.a shows the car driving down a straight after a gradual left turn. As seen in Figure ES-5.b, the car begins to drift right down the straight, until it performs a strong correction. Re-centered, the car encounters a left hair-pin turn seen in Figure ES-5.c. The car successfully navigates the turn, however, it does oversteer slightly. The car continues down a new straight until it fails at a right hair-pin turn. The test showed that the car was capable of performing gradual and hair-pin turns, while also showing an effort to make turning corrections when placed in precarious positions. Ultimately, while the car was unable to autonomously navigate a full track, it illustrated the potential of the system. Therefore, the car's performance has indicated the feasibility of producing a 1:10 scale car that can autonomously navigate using solely a front-facing camera.

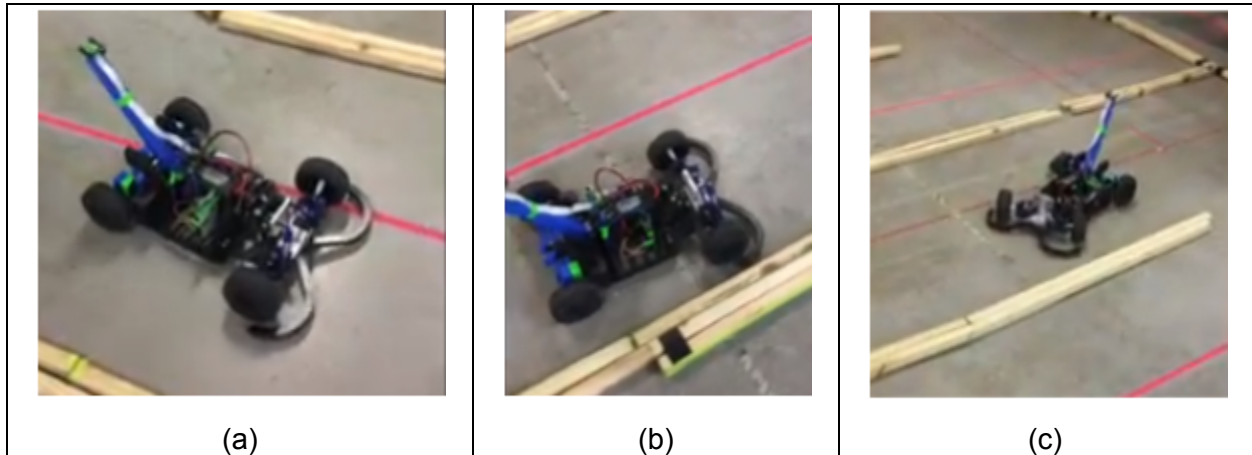


Figure ES-5: Pictures of car driving autonomously around approximately 50% of the track.



# Authorship

<b>Executive Summary</b>	Ryan
<b>1.0 Introduction</b>	
1.1 Driving Technology	Ryan, Krishna
1.2 ME 4320: Advanced Engineering Design	Ryan
1.3 Project Statement	Ryan
1.4 Format of Report	Mitchell
<b>2.0 Background</b>	
2.1 What is an RC Car?	Ryan
2.2 Components of an RC Car	Ava, Dylan, Mitchell, Ryan
2.3 Sensors, Artificial Intelligence and Neural Networks	Christian, Ethan, Krishna, Ryan
2.4 Existing Autonomous Solutions	Ryan
2.5 ME 4320 Advanced Engineering Design	Ava
<b>3.0 General Methodology</b>	Mitchell
<b>4.0 The Test Car</b>	Mitchell, Ryan
<b>5.0 Custom Car</b>	
5.1 Early Stages	Ava, Dylan, Mitchell, Ryan
5.2 Design Issues	Mitchell, Ryan
5.3 Last Stage	Mitchell, Ryan
5.4 Conclusion	Mitchell
<b>6.0 Mechanical Analysis</b>	
6.1 Static Analysis - Front Suspension	Ryan
6.2 Bond Graph Analysis	Mitchell
6.3 Finite Element Analysis for Von Mises Stress	Ava
6.4 Static and Dynamic Analysis for Steering System	Dylan
<b>7.0 Sensor Dashboard</b>	
7.1 Sensors	Krishna, Ryan
7.2 Implementation	Krishna, Ryan
7.3 Modularization of Sensor System	Krishna
7.4 Data Collection	Krishna
7.5 Sensor Dashboard	Ethan
7.6 Conclusion	Krishna
<b>8.0 Autonomous Driving</b>	
8.1 Implementation	Ethan
8.2 Collecting Data	Christian, Ethan
8.3 Formatting Data	Christian, Ethan
8.4 Issuing Commands	Christian, Ethan
8.5 Conclusion	Ethan
<b>9.0 Conclusion</b>	All

# Table of Contents

<b>Abstract</b>	<b>1</b>
<b>Acknowledgments</b>	<b>2</b>
<b>Executive Summary</b>	<b>3</b>
<b>Authorship</b>	<b>8</b>
<b>Table of Contents</b>	<b>9</b>
<b>List of Figures</b>	<b>13</b>
<b>List of Tables</b>	<b>18</b>
<b>List of Nomenclature</b>	<b>20</b>
<b>1.0 Introduction</b>	<b>21</b>
1.1 Driverless Technology	21
1.2 ME 4320: Advanced Engineering Design	22
1.3 Project Statement	22
1.4 Format of Report	22
<b>2.0 Background</b>	<b>24</b>
2.1 What is an RC Car?	24
2.2 Components of an RC Car	25
2.2.1 Mechanical Components	25
Motor	25
Transmission	26
Driveshaft	28
Differential	28
Suspension	29
Steering	30
Chassis	32
2.2.2 Electrical Components	34
Motor	34
Battery	34
Servo Motor	35
Electronic Speed Controller	35
Voltage Regulator	36
2.3 Sensors, Artificial Intelligence and Neural Networks	39
2.3.1 Sensors	39
Accelerometers	42
Gyroscope	42

Magnetometer	42
Determining Orientation with Inertial Measurement Unit Sensors	42
2.3.2 Artificial Intelligence	43
2.3.3 Neural Network	43
2.4 Existing Autonomous Solutions	45
2.5 ME 4320 Advanced Engineering Design	45
<b>3.0 General Methodology</b>	<b>49</b>
<b>4.0 The Test Car</b>	<b>51</b>
4.1 Donation of a Test Car	51
4.2 Preliminary Hardware	52
4.2.1 Sensor	52
4.2.2 Car Control	53
4.2.3 AI Computation	53
4.3 Mounts	53
4.4 Communication	56
4.5 Power	63
4.6 Implementation Problems	63
4.7 Proof of Concept Results	64
<b>5.0 Custom Car</b>	<b>66</b>
5.1 Early Stages	66
5.1.1 Steering Assembly	69
5.1.2 Front Suspension	71
5.1.3 Drivetrain	72
5.1.4 Rear Suspension	80
5.1.5 Other Components	81
Battery	81
Camera	82
Chassis	84
Mounts	84
5.1.6 Electronics	87
Power	87
Control Systems	87
5.2 Design Issues	87
5.2.1 Front Suspension	88
5.2.2 Steering Assembly	88
5.2.3 Gearbox	89
5.2.4 Additional Issues	89
5.3 Last Stage	90

5.3.1 Component Changes	90
5.3.2 Design Alterations	91
5.3.3 Controls	93
5.4 Conclusion	94
<b>6.0 Mechanical Analysis</b>	<b>95</b>
6.1 Static Analysis - Front Suspension	95
6.2 Bond Graph Analysis	102
6.3 Finite Element Analysis for Von Mises Stress	106
6.4 Static and Dynamic Analysis for Steering System	110
<b>7.0 Sensor Dashboard</b>	<b>111</b>
7.1 Sensors	111
7.1.1 Thermal System	111
Temperature Sensor	111
Cooling Fan	112
7.1.2 Inertial Measurement Unit	113
7.1.3 Tachometer	114
7.2 Implementation	115
7.2.1 Thermal System	116
Temperature Sensors	116
Cooling Fan	118
Thermal Control	120
7.2.2 Inertial Measurement Unit	122
Circuit	122
Gyroscope	122
Complementary Filter	126
7.2.3 Tachometer	129
7.3 Modularization of the Sensor System	130
7.4 Data Collection	134
7.5 Sensor Dashboard	136
7.5.1 Display Server	136
7.5.2 Graphics	138
7.6 Conclusion	138
<b>8.0 Autonomous Driving</b>	<b>140</b>
8.1 Implementation	140
8.2 Collecting Data	140
8.2.1 Collection System	141
8.3 Formatting Data	142
8.4 Issuing Commands	145

8.4.1 Arduino Serial Events	145
8.4.2 Neural Network Design	146
8.5 Conclusion	149
<b>9.0 Conclusion</b>	<b>150</b>
<b>References</b>	<b>154</b>
<b>Appendices</b>	<b>161</b>
Appendix A: Finite Element Analysis	161
Appendix B: Design for Manufacturability Manual	165
Appendix C: Arduino Scripts	182
Appendix C: Bond Graph State Equations and Derivations	196
Appendix D: Arduino script: Multiple DHT22 Temperature Sensors	200
Appendix E: Arduino script: IMU	201
Appendix F: Bill of Materials	206
Appendix G: Arduino Script: Sensor System	209
Appendix H: Dynamical Analysis of Steering Assembly	214

## List of Figures

### Executive Summary

Figure ES-1: Labeled top view of the GT-2Ve RC Ca	4
Figure ES-2: Final iteration of 1:10 scale car	5
Figure ES-3: Example of graphs on webpage	6
Figure ES-4: Standard dashboard layout for sensor module	6
Figure ES-5: Pictures of car driving autonomously around approximately 50% of the track	7

### Chapter 2

Figure 2-1: 1/10 <sup>th</sup> scale RC car designed after a Ford Mustang	24
Figure 2-2: Top view of the GT-2Ve RC Car	25
Figure 2-3: Standard Hobbyist Electric Motor	26
Figure 2-4: Driveshaft for Jeep Wrangler	28
Figure 2-5: Open differential with pinion and ring gears	28
Figure 2-6: Suspension Components	29
Figure 2-7: Suspension Components II	29
Figure 2-8: Model of rack and pinion steering linkage	31
Figure 2-9: Drawing of Haltenberger linkage	31
Figure 2-10: Drawing of parallelogram linkage	32
Figure 2-11: Example of chassis on a larger car	33
Figure 2-12: Example of chassis for RC car	33
Figure 2-13: Standard RC Servo Motor	35
Figure 2-14: Brushless Electronic Speed Controller	36
Figure 2-15: Voltage Divider Circuit	37
Figure 2-16: Zener Diode Voltage Regulator	37
Figure 2-17: Switching power supplies	38
Figure 2-18: GPS Trilateration with Uncertainty	40
Figure 2-19: Camera-enabled computer vision in Tesla Autopilot	41
Figure 2-20: Internodal connections within an Artificial Neural Network	44
Figure 2-21: A car designed by Vinve Lucca, Joe Stapleton, Peter Nash, Colin Saunders, Zach Fischer, Jake Chiudina featuring a billet machined chassis and laser cut gearbox housing.	46
Figure 2-22: A steering linkage designed by Derek Kruzan, Tim Esworthy, Walter Kwiecinski, Jessica Elder, Nolan Bell and Xavier Little, featuring entirely 3D printed components.	46
Figure 2-23: A gearbox designed by Amanda Richards, Michael Wilkinson, Tim Bill, Jamie Stephen, and Chris Mayforth, constructed out of mostly VEX gears and axles.	47

### Chapter 3

Figure 3-1: General Methodology Flow Chart	50
--	----

## Chapter 4

Figure 4-1: Test RC Scale Car. created by Alan Hunt, Colin Maschack, Aimilios, Tachiaos, Jake Halverson, Jesse Kaufman, Marissa Ford, Kyle Whittaker, Howard Vance, Frank Bucknor, and Edward Crofts.	52
Figure 4-2: Pegboard-style cover so the camera can be mounted at several locations	54
Figure 4-3: Camera mount assembly	54
Figure 4-4: Arduino and Raspberry Pi pegboard mounts	55
Figure 4-5: ESC and Receiver mount and Battery mount	55
Figure 4-6: Test car modifications	56
Figure 4-7: Voltage of receiver input signal	57
Figure 4-8: Arduino and Receiver Schematic	57
Figure 4-9: Method one duty cycle measurement with one pin measurement	58
Figure 4-10: Method one duty cycle measurement with two pin measurements	59
Figure 4-11: Method two duty cycle measurement (two pins)	59
Figure 4-12: Time period of receiver input signal	60
Figure 4-13: Duty cycle measurement time of one pin	60
Figure 4-14: Conductor, measured duty cycle of the input signal	61
Figure 4-15: Conductor, the output signal from the Arduino	62
Figure 4-16: Conductor circuit schematic	62
Figure 4-17: Steering assembly failures of test car	64

## Chapter 5

Figure 5-1: Initial Car Design	66
Figure 5-2: Initial layout of custom car	67
Figure 5-3: Initial custom car dimensions	67
Figure 5-4: Vehicle Subsystem connections	68
Figure 5-5: Kinematic diagram of steering assembly	69
Figure 5-6: Steering System	70
Figure 5-7: Selected Servo	70
Figure 5-8: SolidWorks Drawing of Servo	71
Figure 5-9: Front Suspension	72
Figure 5-10: Selected Motor	73
Figure 5-11: SolidWorks Drawing of Motor	74
Figure 5-12: Selected ESC	74
Figure 5-13: SolidWorks Drawing of ESC	75
Figure 5-14: Transmission Design	76
Figure 5-15: Gear Mesh Within Gearbox	77
Figure 5-16: Driveshaft Design	78
Figure 5-17: Differential and Differential Housing	79
Figure 5-18: Selected Differential	79
Figure 5-19: SolidWorks Assembly of Powertrain	80
Figure 5-20: 2D Drawing of SolidWorks Assembly of Powertrain	81

Figure 5-21: Dependent Rear Suspension	81
Figure 5-22: Selected Battery	82
Figure 5-23: SolidWorks Drawing of Battery	83
Figure 5-24: Chassis	84
Figure 5-25: Raspberry Pi, Arduino, and PCB Mount	85
Figure 5-26: Temperature Sensor Mounts for Raspberry Pi	85
Figure 5-27: Temperature Sensor Mount for Motor	86
Figure 5-28: Battery Mount	86
Figure 5-29: ESC and Receiver Mount	86
Figure 5-30: Control of Servo and Motor During ANN Training	87
Figure 5-31: Control of Servo and Motor During ANN Testing	87
Figure 5-32: New Servo Motor	90
Figure 5-33: Isometric view of steering assembly solution	91
Figure 5-34: Top view of steering assembly solution	92
Figure 5-35: Model of the custom car with the bumper	92
Figure 5-36: Independent Rear Suspension	93
Figure 5-37: Finalized Car	93

## Chapter 6

Figure 6-1: SolidWorks Model of Half of Front Suspension	95
Figure 6-2: Kinematic Outline of Front Suspension	95
Figure 6-3: Wheel Assembly	96
Figure 6-4: Free Body Diagram of Wheel Assembly	96
Figure 6-5: Wheel Mount Holder	97
Figure 6-6: Free Body Diagram of Wheel Mount Holder	97
Figure 6-7: Camber Link	98
Figure 6-8: Free Body Diagram of Camber Link	98
Figure 6-9: Wishbone	99
Figure 6-10: Free Body Diagram of Wishbone	99
Figure 6-11: Shock Absorber	99
Figure 6-12: Free Body Diagram of Shock Absorber	100
Figure 6-13: Shock Tower	101
Figure 6-14: Free Body Diagram of Shock Tower	101
Figure 6-15: Labeled Gearbox Diagram	103
Figure 6-16: Gearbox Bond Graph	103
Figure 6-17: Gearbox matrix equations	104
Figure 6-18: Half car model for suspension	105
Figure 6-19: Suspension Bond Graph	105
Figure 6-20: Suspension Bond Graph Matrix	106
Figure 6-21: SolidWorks FEA of Servo Horn	107
Figure 6-22: Von Mises Stress of Servo Horn	108
Figure 6-23: Von Mises Stress of Steering Pivot	109
Figure 6-24: Von Mises Stress of Steering Arm	110



Figure 6-25: Steering Linkage System	110
--------------------------------------	-----

## Chapter 7

Figure 7-1: DHT22 Temperature Sensor	112
Figure 7-2: Noctua NF-A4x10 cooling fan	113
Figure 7-3: Pololu MiniIMU-9 v3.	114
Figure 7-4: KY-003 Hall-Effect Sensor	115
Figure 7-5: Neodymium Magnets	115
Figure 7-6: Arduino and sensor communication	116
Figure 7-7: Arduino communication with the thermal system	116
Figure 7-8: DHT22 Temperature sensor circuit	117
Figure 7-9: Multiple DHT22 temperature sensors circuit	117
Figure 7-10: Average temperature reading output to the Arduino's serial monitor	118
Figure 7-11: Temperature sensor locations	118
Figure 7-12: NF-A4x10 circuit	119
Figure 7-13: Cooling Fan Placement	120
Figure 7-14: Circuit diagram of integrated temperature sensors and cooling fan	120
Figure 7-15: Temperature sensors and cooling Fan Integrated into the Thermal System	121
Figure 7-16: Output to fan from the Arduino based on the average temperature	122
Figure 7-17 : IMU Circuit Schematic	122
Figure 7-18: Calibration readings	123
Figure 7-19: Resulting output from Gyro_minimu.ino	124
Figure 7-20: Base case, in resting position (Nose up)	124
Figure 7-21: Test 1, rotate X axis counterclockwise by 90	125
Figure 7-22: Test 2, rotate X Clockwise by 90 and rotate Z Clockwise by 90	126
Figure 7-23: Terminal Showing Readings from Complementary Filter	127
Figure 7-24: Comparing errors in Gyro vs Complementary filter for test 1	128
Figure 7-25: Comparing errors in Gyro vs Complementary filter for test 2	129
Figure 7-26: Hall Effect Sensor	130
Figure 7-27: Flowchart of Sensor Dashboard Data	130
Figure 7-28: Combined sensor circuit schematics	131
Figure 7-29: Temperature sensor components in Altium schematics Library	132
Figure 7-30: Final PCB schematics in Altium	132
Figure 7-31: DHT22 sensor components on top layer in PCB design	133
Figure 7-32: 2-D design, 3-D view of final PCB design	133
Figure 7-33: Final Printed Circuit Board for Sensor Dashboard	134
Figure 7-34: Data collected by the tachometer	135
Figure 7-35: Data collected by Temperature sensors 1, 2 and 3	135
Figure 7-36: Data collected by the IMU	136
Figure 7-37: Data flow from Arduino to CSV and Webpage	137
Figure 7-38: Mockup displays of sensor dashboard data	138

## **Chapter 8**

Figure 8-1: AI Data Flow	140
Figure 8-2: Sample of data written to file format FRAME_NUM: ACCEL,TURN	142
Figure 8-3 Front positioned camera data	143
Figure 8-4: Raw versus Masked Image Data	143
Figure 8-5: Example switchback track	144
Figure 8-6: Example rounded track	145
Figure 8-7: Simple Fully Connected Neural Network	147
Figure 8-8: Neural Network Model Inspired by EuroPilot	148

## **Appendix A**

Figure A-1: Von Mises Stress Analysis for Front Suspension Lower Link	161
Figure A-2: Displacement Analysis for Front Suspension Lower Link	162
Figure A-3: Von Mises Stress for Servo Horn (Bottom View)	163
Figure A-4: Displacement Analysis for Servo Horn (Top View)	163
Figure A-5: Displacement Analysis for Servo Horn (Side View)	164

## **Appendix B**

Figure B-1: Square vs. Rounded Internal Corners	167
Figure B-2: Fillets vs. Chamfers	168
Figure B-3: Part Fixturing	169
Figure B-4: Fixture Matching	169
Figure B-5: Tap Drill Chart	171
Figure B-6: SolidWorks Hole Wizard	172
Figure B-7: Internal Part Features	174
Figure B-8: Simple vs. Hard Parts	174
Figure B-9: Unsupported vs. Supported Cuts	176
Figure B-10: 3D Printing Orientation	178
Figure B-11: 3D Printing Overhangs	179
Figure B-12: 3D Printing Bridging	179
Figure B-13: 3D Printing Vertical Towers	180
Figure B-14: 3D Printing Infill	180

## **Appendix D**

Figure D-1: Temperature Sensor Arduino Code	200
Figure D-2: IMU Arduino Code	205

## List of Tables

### Chapter 2

Table 2-1: Benefits and Drawbacks of RC Transmission Systems	27
Table 2-2: Common Vehicle Suspension Systems	30
Table 2-3: Brushed vs. Brushless Electric Motors	34
Table 2-4: NiMH vs. LiPo Batteries	35
Table 2-5: Measurement and Orientation Contribution of each IMU Component	43

### Chapter 4

Table 4-1: Proof of Concept Questions for Test Car	51
--	----

### Chapter 5

Table 5-1: Servo Motor Specifications	71
Table 5-2: Motor Specifications	73
Table 5-3: ESC Specifications	75
Table 5-4: Gear Specifications	76
Table 5-5: Differential Specifications	79
Table 5-6: Battery Specifications	82
Table 5-7: Camera Specifications	83
Table 5-8: New Servo Motor Specifications	90

### Chapter 6

Table 6-1: Static forces of Front Suspension	102
Table 6-2: Simulation Parameters for Von Mises Stress	109

### Chapter 7

Table 7-1: DHT22 vs. DHT11 Temperature Sensors	112
Table 7-2: Cooling Fan Specifications [73]	113
Table 7-3: Pololu MinIMU-9 v3 specifications [75]	114
Table 7-4: PWM pin values and corresponding duty cycles	115
Table 7-5: Arduino PWM output based on measured temperature	119
Table 7-6 : Minimum and Maximum IMU sensor headings	121
Table 7-7: Difference between the Calculated and Actual readings from test 1	125
Table 7-8: Difference between the Calculated and Actual readings from test 2	126
Table 7-9: Complementary filter measurements vs Gyroscope measurements in test 1	128
Table 7-10: Complementary filter vs Gyroscope readings obtained in test 2	129

### Appendix A

Table A-1: Parameters for FEA on Front Lower Suspension Link	161
Table A-2: Parameters for FEA on Servo Horn	162

**Appendix F**

Table F-1: Bill of Materials

207

## List of Nomenclature

### Abbreviations

RC: Radio Controlled  
AI: Artificial Intelligence  
R&D: Research and Development  
POC: Proof of Concept  
ANN: Artificial Neural Network  
CNN: Convolutional Neural Network  
WPI: Worcester Polytechnic Institute  
ESC: Electronic Speed Controller  
RGB: Red-Green-Blue  
PWM: Pulse Width Modulation  
IMU: Inertial measurement unit  
LiPo: Lithium Polymer  
NiMH: Nickel-Metal Hydride  
SAE: Society of Automotive Engineers  
FEA: Finite Element Analysis  
SDA: Serial Data  
SCL: Serial Clock Line  
PCB: Printed Circuit Board  
CSV: Comma Separated Values  
ES: Executive Summary  
MQP: Major Qualifying Project

### Bond Graph Analysis Variable Notation

$J_i$ : Angular mass moment of inertia i  
 $k_i$ : Spring constant i  
 $D_i$ : Damping constant i  
 $\omega_i$ : Angular velocity i  
 $v_i$ : Velocity i  
 $F_i$ : Force i  
 $h_i$ : Angular momentum i  
 $p_i$ : Translational momentum i  
 $x_i$ : Position i  
 $h_i'$ : Derivative of angular momentum i  
 $p_i'$ : Derivative of angular momentum i  
 $x_i'$ : Derivative of position i  
 $N_A$ : Number of teeth on gear A  
 $\theta_i$ : Derivative of angular position i  
 $\theta_i$ : Angular position i  
 $\tau_i$ : Torque i  
A: Gear A in gearbox assembly  
 $L_i$ : Shaft i the gearbox assembly

## 1.0 Introduction

### 1.1 Driverless Technology

The quest for fully autonomous vehicles has sparked rapid growth in research and development (R&D) among automotive giants. Evidenced by Ford's \$1 billion investment in ArgoAI and SoftBank Vision Fund's \$2.25 billion investment in GM Cruise Holdings, the influx of wealth devoted to the creation of autonomous vehicles has created unprecedented demand for engineers and advancements in technologies like artificial intelligence (AI) and computer vision [5, 6]. While the effects may ultimately be for the better, or for the worse, the race to create fully autonomous vehicles contains global implications.

As per the guidelines set by the Society of Automotive Engineers (SAE), there are six levels to autonomy. The seemingly unanimous system by which automotive producers rank vehicles, the SAE guidelines illustrate the spectrum between no automation (Level 0) and full automation (Level 5) [75]. Despite the relatively subjective nature by which companies rank cars, the end goal for the creation of autonomous cars is clear: cars that can safely operate without any instance of user input. In order to achieve such independence, autonomous cars leverage a plethora of sensors and supercomputers. Integrating big data analytics with AI, car manufacturers must focus not only on keeping a vehicle on the road, but also the abundance of indefinable variables that characterize human action and inaction.

In an attempt to scale down the enormous financial, moral, and technological implications of driverless technology, consider the Radio Controlled (RC) car. A cheap and simple means of replicating the dynamic behavior of a full-sized vehicle, the RC car can replicate full-scale conditions without severe life-threatening or financial consequences. Furthermore, in an effort to expand on the standard complications of autonomous vehicles, consider the following: the creation of a modular AI system for autonomous cars. Inherently "plug and play," this system will not only drive one RC car in volatile conditions, but instead, it can be readily adapted to drive any RC car in volatile conditions.

Over the previous decade, an abundance of work has been documented in the autonomous car spectrum. Seen most frequently with automotive giants, car manufacturers like Ford regularly report work showing their progress towards creating a Level 5 autonomous vehicle by the year 2021 [8]. The quest for autonomous vehicles, however, has also expanded into the hobbyist and academic sectors. Seen in the creation of autonomous scale vehicle kits by hobbyist companies such as Donkey Car and the work on an autonomous golf cart by academic scholars at Worcester Polytechnic Institute, the challenge in creating autonomous vehicles has produced an immense quantity of scholarly work [4,9]. While the design of an autonomous vehicle has become a global challenge, however, the modular AI system intricacy appears to be a relatively novel concept.

## **1.2 ME 4320: Advanced Engineering Design**

ME 4320, Advanced Engineering Design, is a capstone design class for mechanical engineers at Worcester Polytechnic Institute (WPI). The course requires students to work in teams to design and fabricate a 1:10th RC car. RC cars are the subject of the final project in the course because they have many complex subsystems that need to work together in order for the car to perform well. Each sub system such as the gearbox, transmission, and steering needs to be well designed, well fabricated, and analyzed to ensure that it will work well in the situation outlined in the competition rules. The cars were then evaluated in a race at the end of the term.

One of the problems that arose in the race was the issue of driver skill. It became apparent that the biggest factor in who won the race was the amount of practice the driver had with the car, not the car itself. Often poorly designed cars would out race well designed cars because they had more experienced drivers. In order to address this problem, Professor Radhakrishnan tasked us with creating a proof of concept car to test whether self driving cars would be feasible on this scale. He hopes that in the future, all of the cars created in the course will be able to be autonomously race each other, remove the drive from the equation entirely.

Additionally, during the course, students are tasked with analyzing the mechanical systems present on their cars. In an effort to validate their theoretical calculations and provide increased exposure to electronic circuits, specifically sensors, Professor Radhakrishnan tasked us with creating a modular sensor dashboard. The sensor dashboard is intended to be compatible with ME4320 cars. Furthermore, the dashboard should be both scalable and customizable to suit each team's individual requirements.

## **1.3 Project Statement**

This report explores the feasibility of creating a modular AI system for autonomous cars. Using a 1/10 scale RC car as the testing platform, this project will have five major examinations. First, the project will test the ability of manufacturing a 1:10 scale car equipped with front and rear suspension. Second, the project will test the performance of a modular sensor package capable of producing live data updates. Third, the project will test the ability of a trained RC car to drive itself around an indoor track without the use of mapping. Fourth, the project will test the ability of the RC car to navigate around the track while racing against other, human controlled RC cars. Finally, the project will test the feasibility of creating a modular AI system. Specifically, it will test whether or not an AI system can be readily fitted to other cars so that they can navigate autonomously.

## **1.4 Format of Report**

The remainder of the paper will be formatted as follows. The second chapter will contain background discussing RC cars, sensors, autonomous solutions, and neural networks. The third chapter will provide a brief overview of the project's procedure and timeline. The fourth chapter will discuss the proof of concept work that was done to validate the team's ability to accomplish the project goals. The fifth chapter will detail the design and construction of the custom car built

by the team. The sixth chapter will discuss the mechanical analysis of the systems on the custom car. The seventh chapter will detail the modular sensor system made by the team. The eighth chapter will detail the process of selecting and implementing a neural network. The last chapter, chapter eight, will conclude the paper, wrapping up the discussions in the previous chapters and conveying final thoughts.



## 2.0 Background

First, this chapter defines the RC car and the major mechanical and electrical subassemblies of which it is comprised. Following, the chapter provides an introduction to artificial intelligence (AI), most specifically emphasizing the basic software and hardware requirements needed to create an autonomous vehicle. This chapter ends with a brief discussion of the Worcester Polytechnic Institute (WPI) course, ME4320, Advanced Engineering Design. Improving this course is a primary objective to the work performed in this project.

### 2.1 What is an RC Car?

An RC, or Radio Controlled car is a car operated by a receiver-transmitter pair [10]. Commonly owned by hobbyists, RC cars are often scaled-down versions of popular cars. For example, Figure 2-1 (reproduced as is from [10]) shows a 1/10<sup>th</sup> scale RC car of the Ford Mustang.



Figure 2-1: 1/10<sup>th</sup> scale RC car designed after a Ford Mustang. Reproduced as is from [11].

Often modeling full-sized vehicles, RC cars contain many of the electrical and mechanical components found within normal street vehicles. Their more compact, simplistic, and cheap nature, however, make them a great tool for hobbyists and autonomous vehicle testing. Shown in Figure 2-2 is a top view of a GT-V2e, a relatively standard RC car. Specifically, Figure 2-2 (reproduced as is from [1]) and its corresponding legend highlight the major mechanical and electrical components found within conventional scale RC cars.

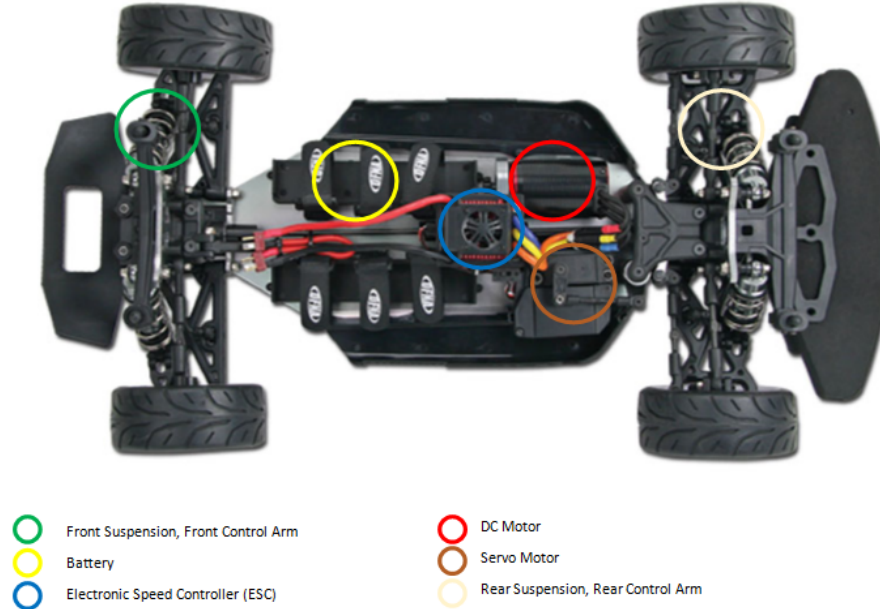


Figure 2-2: Top view of the GT-2Ve RC Car. As per the legend, the circled components of the car illustrate many of the important subassemblies fitted to the vehicle. Reproduced as is from [1].

The cars built by the students in ME 4320 share many similarities to commercially available cars, such as the one pictured in Figure 2-2. The typical cars built by students have many similar components including a battery, DC motor, gearbox, servo motor, and steering linkage, but vary in other ways by not having suspension systems. By not including suspension in their car design, the steering and transmission linkages are dramatically simplified. These cars are discussed in more detail in Section 2.5. It is important to consider the design of the cars made by students in ME 4320 as the long term goal of the project is to make the cars designed and built by students in ME 4320 self drive.

## 2.2 Components of an RC Car

### 2.2.1 Mechanical Components

As seen in Figure 2-2 (reproduced as is from [1]), the scale RC car contains many mechanical components. For simplicity, the most important components include: the motor, transmission, driveshaft, differential, suspension, steering, and chassis.

#### Motor

While the majority of full-sized cars rely on the internal combustion engine, the RC car normally relies on electric motors. Drastically easier to scale and much more appropriate for the physical requirements of the RC car, the electric motor for RC applications comes in two general options: brushed and brushless. While the specifics of the brushed and brushless motors will be

discussed in Section 2.3, it is essential first to note the general mechanics and function of the electric motor.

A typical electric motor used in hobbyist activities can be seen in Figure 2-3 (reproduced as is from [12]). In its simplest sense, the electric motor receives electrical energy via electric current and ultimately converts said electrical energy into rotational, mechanical motion. Leveraging the inherent relationship between electricity and magnetism to convert energy, the electric motor is comprised of a rotor and a stator. While the specific construction of the rotor and stator varies among the many electric motor types, at least one of the components utilizes coiled wire to behave as an electromagnet. Consequently, the introduction of current to the electromagnet creates a magnetic field which interacts with the magnetic field of the complementary component. Inevitably, the proper manipulation of these two magnetic fields creates a rotational force. Based on this theoretical foundation, the electric motor is ultimately capable of creating very high instantaneous torque values. As a result, the electric motor produces rapid acceleration forces [13].



Figure 2-3: Standard Hobbyist Electric Motor. Reproduced as is from [12].

### **Transmission**

The main purpose of the transmission is to convert the high frequency (rotations per minute) rotational power generated by the electric motor to a more controlled and optimal rotational power at the vehicle wheels [14]. While road cars are typically known to possess variations of automatic or manual transmissions, the RC car typically utilizes one of five options: direct drive, compound gearing, belt drive, chain drive, and planetary gearing [15]. For simplicity, Table 2-1 illustrates many of the benefits and drawbacks of each transmission system.

Table 2-1: Benefits and Drawbacks of RC Transmission Systems [15]

Benefits	Drawbacks
<p>Direct Drive:</p> <ul style="list-style-type: none"> <li>- Lightweight</li> <li>- Simple</li> <li>- Highly efficient</li> </ul>	<p>Direct Drive:</p> <ul style="list-style-type: none"> <li>- No gear reduction</li> <li>- Gears easily damaged by debris</li> </ul>
<p>Compound Gearing:</p> <ul style="list-style-type: none"> <li>- Easy to assemble</li> <li>- Easy to maintain</li> <li>- Non-slip</li> <li>- Large constant velocity ratios</li> <li>- Can handle high-end loads</li> <li>- Can transmit large amounts of power</li> <li>- High transmission efficiency (85-95%)</li> <li>- Compact</li> <li>- Can transmit power between non-parallel shafts</li> </ul>	<p>Compound Gearing:</p> <ul style="list-style-type: none"> <li>- Not ideal for long distance power transmission</li> <li>- Not flexible</li> <li>- Not ideal for higher speeds</li> <li>- Heavy</li> <li>- High noise and vibration</li> </ul>
<p>Belt Drive:</p> <ul style="list-style-type: none"> <li>- Easy to assemble</li> <li>- Easy to maintain</li> <li>- Extremely high transmission efficiency (95-98%)</li> <li>- Overload protection</li> <li>- Jam protection</li> <li>- Good for long distance power transmission</li> <li>- Quiet and low vibration</li> <li>- Shock absorption for load fluctuation</li> </ul>	<p>Belt Drive:</p> <ul style="list-style-type: none"> <li>- Not compact</li> <li>- Belts fatigue</li> <li>- Lower power transmission</li> <li>- High load on shafts and bearings</li> <li>- Belt slip</li> </ul>
<p>Chain Drive:</p> <ul style="list-style-type: none"> <li>- Non-slip</li> <li>- Constant angular velocity between sprockets</li> <li>- High-velocity ratios</li> <li>- Strong with power transmission for most shaft distances</li> <li>- Limited maintenance</li> <li>- High transmission efficiency</li> <li>- Relatively compact</li> <li>- Relatively low shaft load</li> </ul>	<p>Chain Drive:</p> <ul style="list-style-type: none"> <li>- Expensive</li> <li>- Difficult to scale down</li> <li>- High noise and vibration</li> <li>- Heavy</li> <li>- Less compact than compound gears</li> <li>- Heavier shaft load than compound gears</li> </ul>
<p>Planetary Gearing:</p> <ul style="list-style-type: none"> <li>- Extremely compact</li> <li>- Input and output shafts in-line</li> <li>- Non-slip</li> <li>- Fully enclosed</li> <li>- High load capabilities</li> </ul>	<p>Planetary Gearing:</p> <ul style="list-style-type: none"> <li>- Not ideal for high speeds</li> <li>- Not flexible</li> <li>- Heavy</li> <li>- Hard to maintain</li> </ul>

## Driveshaft

Using the transmission to optimize the motor output, the next stage of power transmission is the driveshaft. Typically a cylindrical shaft that connects the output of the transmission to the drive axle, the driveshaft translates the optimized motor torque to the axle holding the drive wheels. Shown in Figure 2-4 (reproduced as is from [16]) is a full-sized drive shaft for a Jeep Wrangler. While many design variations of the driveshaft exist, it is best understood as the component which translates motor torque [17].



Figure 2-4: Driveshaft for Jeep Wrangler. Reproduced as is from [16].

## Differential

Using the driveshaft to translate the motor torque across the length of the car to the drive axle, the final stage of power transmission is the differential. Due to the 90 degree angle between the drive shaft (motor rotation) and the drive axle (wheel rotation), the drivetrain must also rotate the motor output. Known as a differential, this component of the drivetrain can rotate the torque of the driveshaft to the rear axle, while simultaneously allowing for independent wheel rotation. While the differential has many different design variations, its most basic design consists of a pinion gear from the driveshaft and a ring gear which rotates about the drive axle [18]. For further clarification, Figure 2-5 (reproduced as is from [19]) illustrates the components of the open differential design.



Figure 2-5: Open differential with pinion and ring gears. Reproduced as is from [19].

## Suspension

Vehicle suspension serves three main purposes: (i) to allow the vehicle to ride undisturbed over rough roads, (ii) to keep the tires in contact with the ground, and (iii) to minimize the body roll experienced by the car. Although there are a variety of different suspension systems, each system is based on the following two components: springs and dampers. When compressed or expanded, springs store mechanical energy and exert a force. Subsequently, this inherent property of springs promotes smoother vehicle travel. Unfortunately, while springs may capture some energy, they will not dissipate said energy. Consequently, springs are most effective when coupled with dampers. Devices which convert kinetic energy into thermal energy, dampers help reduce the movement experienced by the vehicle [20]. For further clarification, Figures 2-6 and 2-7 (reproduced as is from [21-26]) illustrate some examples of both springs and dampers commonly used in vehicles.

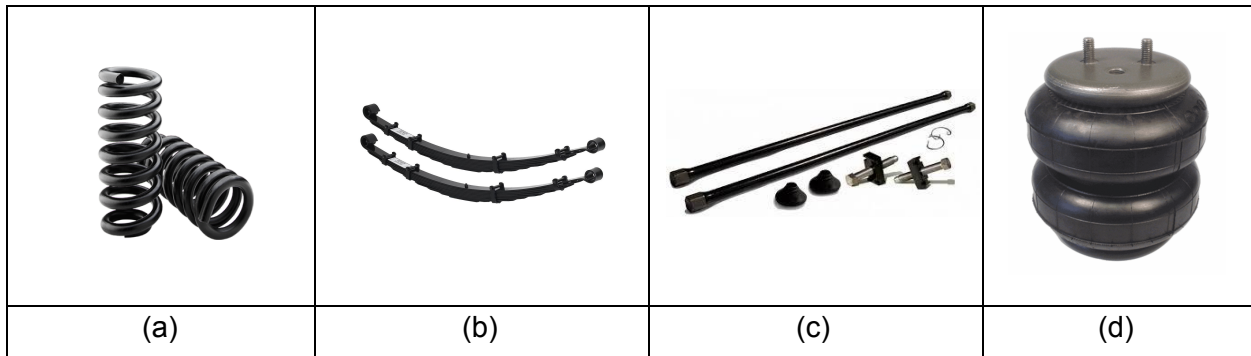


Figure 2-6: Suspension Components (a) Coil spring, (b) leaf spring, (c) torsion bars, and (d) air springs. Reproduced as is from [21-24].

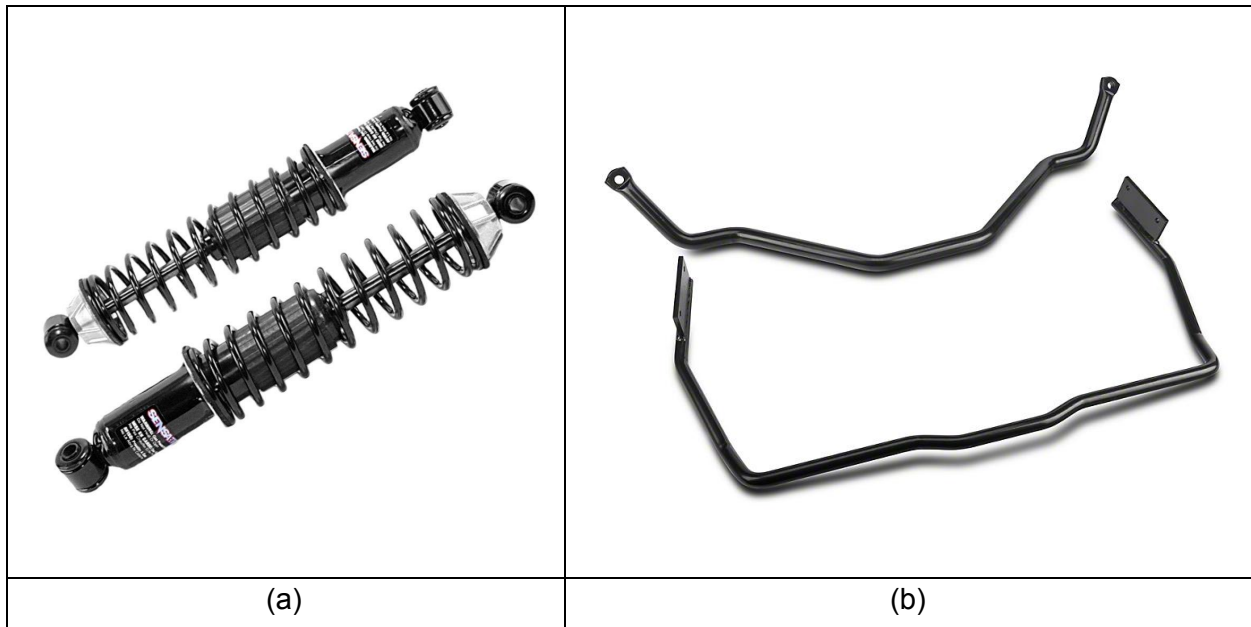


Figure 2-7: Suspension Components II (a) shock absorbers and struts, and (b) anti-sway bar. Reproduced from [25, 26].

In addition to the various combinations of springs and dampers, suspensions can also be dependent or independent. Known for the rigid axle that binds its wheels, dependent suspension is commonly used for the rear suspension of modern full-size vehicles. While this suspension allows for axle articulation, however, any movement experienced by the one wheel will be mirrored by the wheel opposite [20, 27].

Although the rear suspension is commonly dependent, the front suspension is typically independent. Among its many strengths, the front suspension is normally independent for three main reasons: (i) dependent suspension is prone to swaying by virtue of amplification of inertia from one wheel to the other, (ii) dependent suspension increases unsprung weight, and (iii) dependent suspensions make wheel alignment difficult. Furthermore, as the front suspension must also contain the steering mechanism, the following three reasons illustrate the importance of an independent front suspension [20, 27].

Due to the many variables related to suspension design, modern vehicles use a variety of different suspension systems for both the front and rear of the vehicle. For simplicity, Table 2-2 lists several independent front suspension systems and several dependent rear suspension systems.

Table 2-2: Common Vehicle Suspension Systems [20]

Independent Front Suspension Systems	Dependent Rear Suspension Systems
MacPherson Strut	Solid Axle
Double-Wishbone	Beam Axle
Trailing Arm	4-Bar
Twin I-Beam	De Dion
Moulton Rubber	
Transverse Leaf spring	

## Steering

Integrated with the front suspension, the steering system allows the vehicle to change direction. In most cars the steering systems works by changing the angle of the wheel relative to the body of the car. By rotating the wheels, the direction of travel changes so that the car can make turns. Some steering systems work by only rotating the front two wheels, while other systems rotate all four wheels. Given the smaller scale of RC cars, however, the two wheel steering is best suited to simplify the design of the car, as well as, the control scheme.

In order to rotate the front wheels, the car uses a steering linkage. Among the many variations, three of the more standard steering linkage systems include: rack and pinion, parallelogram, and Haltenberger [28]. As will be discussed, each style of linkage has its own advantages and disadvantages for implementation on a 1/10th scale vehicle.

Rack and pinion linkages are commonly among the easiest steering linkages to implement in vehicles. As the steering wheel rotates, a pinion gear rotates against a rack gear causing the linkage to slide. While the physical design is very simple, the disadvantage to using this steering system at a 1/10 scale is that the steering input is controlled by a servo rather than a person turning a steering wheel. As servos typically have a limited range of motion, it would be difficult to effectively slide the linkage [28]. For further illustration, an example of a rack and pinion steering linkage can be seen in Figure 2-8 (reproduced as is from [29]).

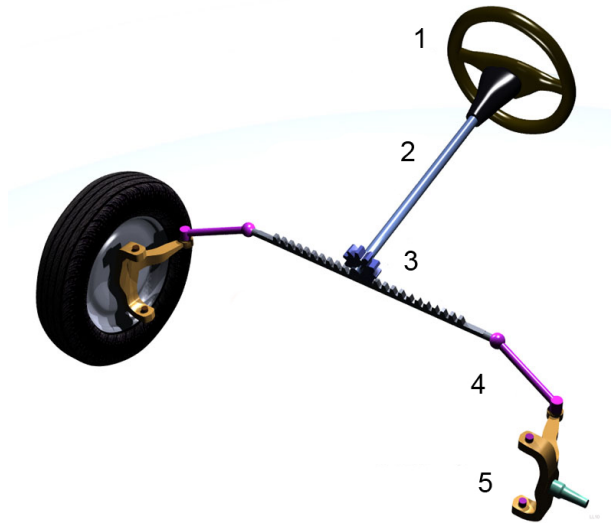


Figure 2-8: Model of rack and pinion steering linkage. Reproduced as is from [29].

The Haltenberger linkage is the most complex of the three as it requires precise geometries and tolerances that would be hard to implement in an RC car. In practice, this system works by rotating a link called the Pitman arm. When the Pitman arm rotates, the inner and outer tie rods move laterally. Consequently, as the tie rods move, the wheels move with them [28]. Figure 2-9 (reproduced as is from [30]) shows the Haltenberger linkage.

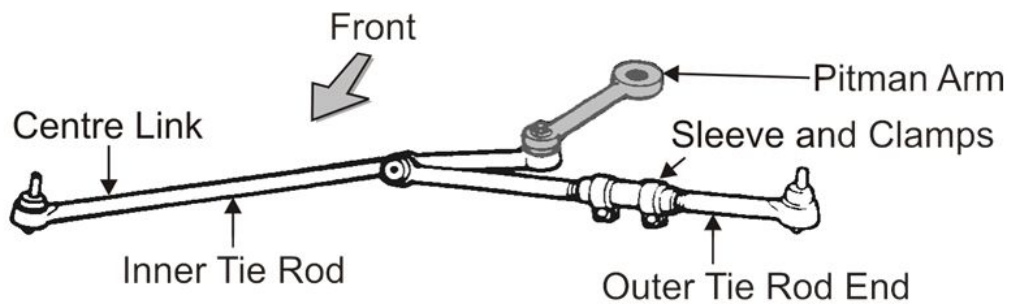


Figure 2-9: Drawing of Haltenberger linkage. Reproduced as is from [30].



In order to prevent binding within the system, the Haltenberger linkage requires very tight tolerances. As a result, it is a very difficult system to implement on a 1:10 scale [28].

The final type of linkage, as shown in Figure 2-10 (reproduced as is from [31]), is a parallel or parallelogram linkage. Sharing many similarities with the Haltenberger linkage, the parallelogram linkage also uses a rotating Pitman arm to provide input to the system. As the Pitman arm rotates, the center link moves, which then connects to the tie rods and finally the wheels [28].

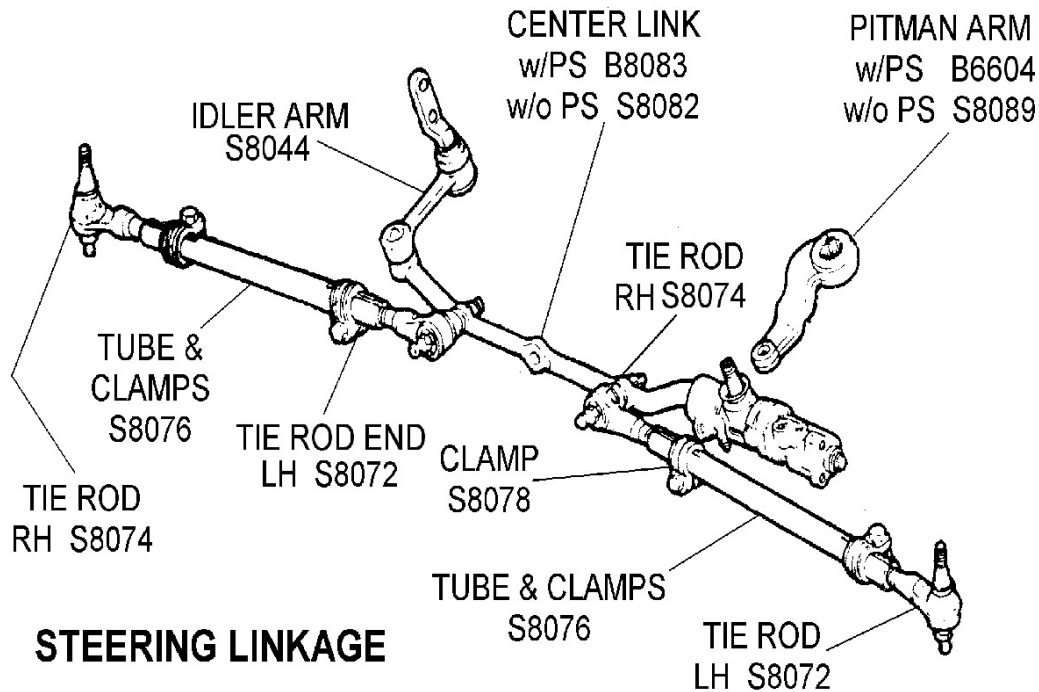


Figure 2-10: Drawing of parallelogram linkage. Reproduced as is from [31].

The disadvantage to this system is the greater number of components than present in the rack and pinion or Haltenberger linkages. While this disadvantage makes manufacturing more complex, the parallelogram linkage can scale down more easily [28].

### Chassis

The chassis serves as the structural foundation of the car. Designed to handle longitudinal, lateral, torsional and bending stresses and strains, the chassis creates a rigid vehicular structure. On a full-size vehicle, the chassis normally forms an exoskeleton. As illustrated in Figure 2-11 (reproduced as is from [32]), interconnecting metal rods create the vehicle frame and roll cage [33]. While the general principle of the chassis extends to scale cars, the design is typically different. Given the smaller, lighter and more compact nature of the scale RC car, the chassis does not normally have to be a series of interconnected metal rods. Furthermore, as toy scale cars are not explicitly designed to handle crashes, the chassis need not be as elaborate. Consequently, the chassis of an RC scale car is often designed as seen in Figure 2-12 (reproduced as is from [34]). Typically a single metal plate that connects the front

and rear axle, this chassis satisfies the significantly simpler mechanical requirements of a scale car.

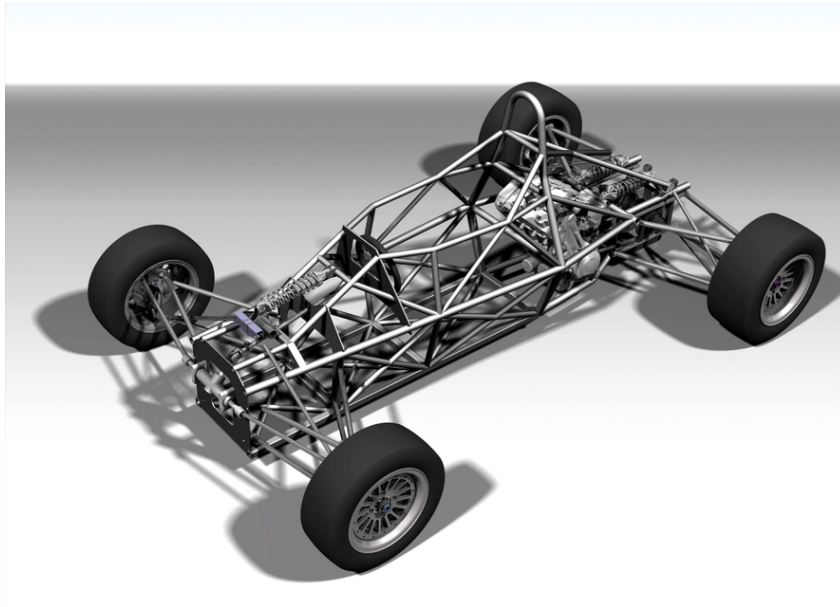


Figure 2-11: Example of chassis on a larger car. Reproduced as is from [32].



Figure 2-12: Example of chassis for RC car. Reproduced as is from [34].

## 2.2.2 Electrical Components

In addition to the mechanical components outlined in Section 2.2, the RC car contains many electrical components. Among the many, the components that are most important are the motor, battery, servo motor, electronic speed controller (ESC), and power regulators.

### Motor

Among the many variations of the electric motor, the two most relevant for RC cars are the brushed DC motor and the brushless DC motor. As outlined in Section 2.2.1, both variations of the electric motor operate according to the same electromagnetic principles. The fundamental difference between the two motors, however, is their means of interaction between the stator and rotor. For simplicity, the strengths and weaknesses of each electric motor are listed in Table 2-3.

Table 2-3: Brushed vs. Brushless Electric Motors [35]

Strengths	Weaknesses
Brushless Motor: <ul style="list-style-type: none"> <li>- Highly efficient</li> <li>- Very powerful</li> <li>- Limited maintenance</li> <li>- Limited wear</li> <li>- Low electromagnetic interference</li> </ul>	Brushless Motor: <ul style="list-style-type: none"> <li>- Expensive</li> <li>- More complicated</li> <li>- Requires Electronic Speed Controller</li> <li>- More susceptible to malfunction</li> </ul>
Brushed Motor: <ul style="list-style-type: none"> <li>- Simple</li> <li>- Rugged</li> <li>- Cheap</li> </ul>	Brushed Motor: <ul style="list-style-type: none"> <li>- Brushes wear out</li> <li>- Less powerful</li> <li>- Mechanical restrictions</li> <li>- Brush arcing</li> </ul>

As highlighted by their listed strengths and weaknesses, the performance characteristics of both the brushless and brushed DC motors are byproducts of their designs. As the brushed DC motor relies on physical connection using contacts (commutators) to energize the rotor windings, mechanical flaws such as inertia and friction compromise the efficiency of the motor. Meanwhile, as the brushless DC motor relies on well-timed stator winding energization, the brushless DC motor achieves more optimal performance but at an additional cost. In conclusion, each motor satisfies specific mechanical needs [35].

### Battery

The battery is an energy storage device which allows for continued operation of an RC car. Utilizing chemical bonds to store electrical energy, the RC car battery powers the motor, electronic speed controller (ESC), servo, and all other peripherals. Among the many different chemical compositions of batteries, the RC community most commonly uses two variations: Nickel-Metal Hydride and Lithium Polymer. Better known as NiMH and LiPo respectively, these two battery types satisfy the same requirement: to power the equipment of the RC car [36]. As illustrated in Table 2-4, however, there are fundamental differences between the performance levels of NiMH and LiPo batteries.

Table 2-4: NiMH vs. LiPo Batteries [36]

Strengths	Weaknesses
NiMH Battery: <ul style="list-style-type: none"> <li>- Rugged</li> <li>- Cheap</li> <li>- Simple</li> </ul>	NiMH Battery: <ul style="list-style-type: none"> <li>- Less energy dense</li> <li>- Heavy</li> <li>- Steadily decreasing voltage</li> </ul>
LiPo Battery: <ul style="list-style-type: none"> <li>- Very energy dense</li> <li>- Light</li> <li>- Maintains constant voltage longer</li> </ul>	LiPo Battery: <ul style="list-style-type: none"> <li>- Expensive</li> <li>- Special maintenance restrictions</li> </ul>

### Servo Motor

While the brushed and brushless DC motors induce motion within RC cars, the servo motor controls steering. As illustrated in Figure 2-13 (reproduced as is from [37]), the servo motor consists of a DC motor, a potentiometer, a gearing system, and a control system. Unlike the DC motors that drive the RC car, the servo does not continuously spin. Instead, the servo receives a Pulse-Width Modulated (PWM) signal from a microcontroller which corresponds to a specific angular rotation. Upon attaining the desired orientation, the servo remains motionless until it receives the next signal instruction [38].



Figure 2-13: Standard RC Servo Motor. Reproduced as is from [37].

### Electronic Speed Controller

The electronic speed controller, or ESC, has two typical configurations: brushed and brushless. Acting as a medium between the battery and the DC motor, the brushed ESC is compatible with brushed DC motors while the brushless ESC is compatible with brushless DC motors. For brushed DC motors, the two-wire ESC acts solely as a voltage regulator for the electric motor. Meanwhile, for the brushless DC motor, the three-wire ESC acts as a voltage regulator and a throttle control. Consequently, the brushless ESC, such as the one in Figure 2-14 (reproduced as is from [39]), utilizes a PWM signal to provide variable speed control [40].

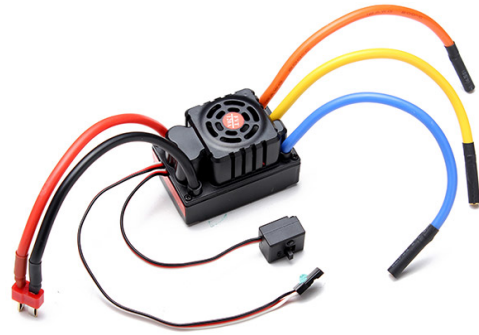


Figure 2-14: Brushless Electronic Speed Controller. Reproduced as is from [39].

### Voltage Regulator

A voltage controller is an electronic circuit designed to provide a constant voltage independent of current requirements [41]. Among the many variations, three types of power regulators were considered for this project: (i) voltage dividers, (ii) zener diode voltage regulators, and (iii) switching power supplies. In order to evaluate each regulator's performance, the important parameters were determined to be: (i) circuit simplicity, (ii) power efficiency, and (iii) voltage consistency for both constant current draw and modulating current draw.

The first power regulating circuit, a voltage divider, was the simplest of the voltage regulators. Shown in Figure 2-15, the voltage divider uses two resistors in series to modify the output voltage across the load,  $R_L$ . Under constant current,  $R_2$  is chosen so that the equivalent resistance of  $R_2$  and  $R_L$  in parallel creates a voltage divider with  $R_1$  that produces the desired output voltage. Under condition two, the current through  $R_2$  must be much larger than the current through  $R_L$  so that fluctuations in the current to the load do not cause large fluctuations in the output voltage. Because constant voltage is required under modulating current, if a voltage divider is used to regulate the voltage, condition two must be used. The constant current through  $R_1$  and  $R_2$  that is significantly greater than the current to the load, makes the voltage divider very inefficient, however, if the current through  $R_1$  and  $R_2$  is large enough, the output voltage can be considered to be constant.

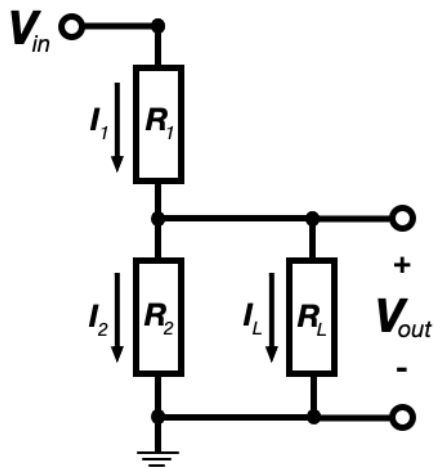


Figure 2-15: Voltage Divider Circuit

The zener diode voltage regulator, as seen in Figure 2-16, has a resistor in series with a reverse biased zener diode. The zener diode is selected with a breakdown voltage equal to the desired  $V_{out}$ . This circuit has two states of operation based on if the  $V_{out}$  that would be produced by the  $R_1$  and  $R_L$  voltage divider is (i) less than the zener diode breakdown voltage or (ii) greater than the zener diode breakdown voltage. In case one,  $V_{out}$  is determined by the  $R_1$  and  $R_L$  voltage divider, with the diode acting as an open circuit. In case two,  $V_{out}$  is equal to the zener diode breakdown voltage because the current that does not go through  $R_L$ , but is needed to maintain  $V_1$  at  $V_{in}$  minus  $V_{BR}$ , goes through the zener diode. Ideally the circuit is always operating under case two so that the voltage is maintained constant independent of  $I_L$  or  $V_{in}$ . Similar to the voltage divider, this circuit is inefficient partially due to the power lost at  $R_1$ , but it is also inefficient because of power lost through  $D$  [42].

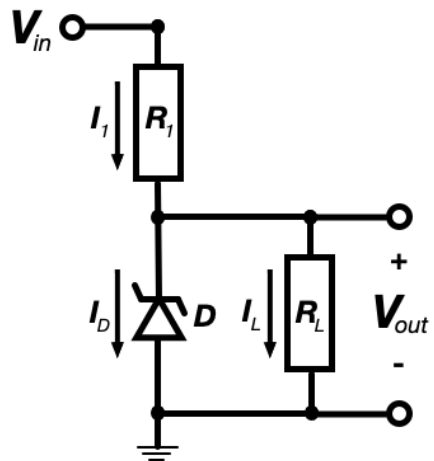


Figure 2-16: Zener Diode Voltage Regulator

Switching power supplies operate by switching the  $V_{in}$  on and off, resulting in an average voltage equal to the desired voltage. Inductors and capacitors are used to minimize the shock to the system between the on and off switching periods. Because there is no series resistor, these switching power supplies can be very power efficient with power losses predominantly coming from the switching circuit. Drawbacks for these circuits include voltage ripples caused by switching, large inductors and capacitors to reduce the voltage ripple, and circuit complexity caused by the switching system. There are three switching power supplies that were considered: (i) the buck converter (Figure 2-17a), which is a step down DC to DC converter, (ii) the buck-boost converter (Figure 2-17b), which is a step up or step down DC to DC converter and (iii) the Cuk converter (Figure 2-17c), which is a constant power DC to DC converter [43].

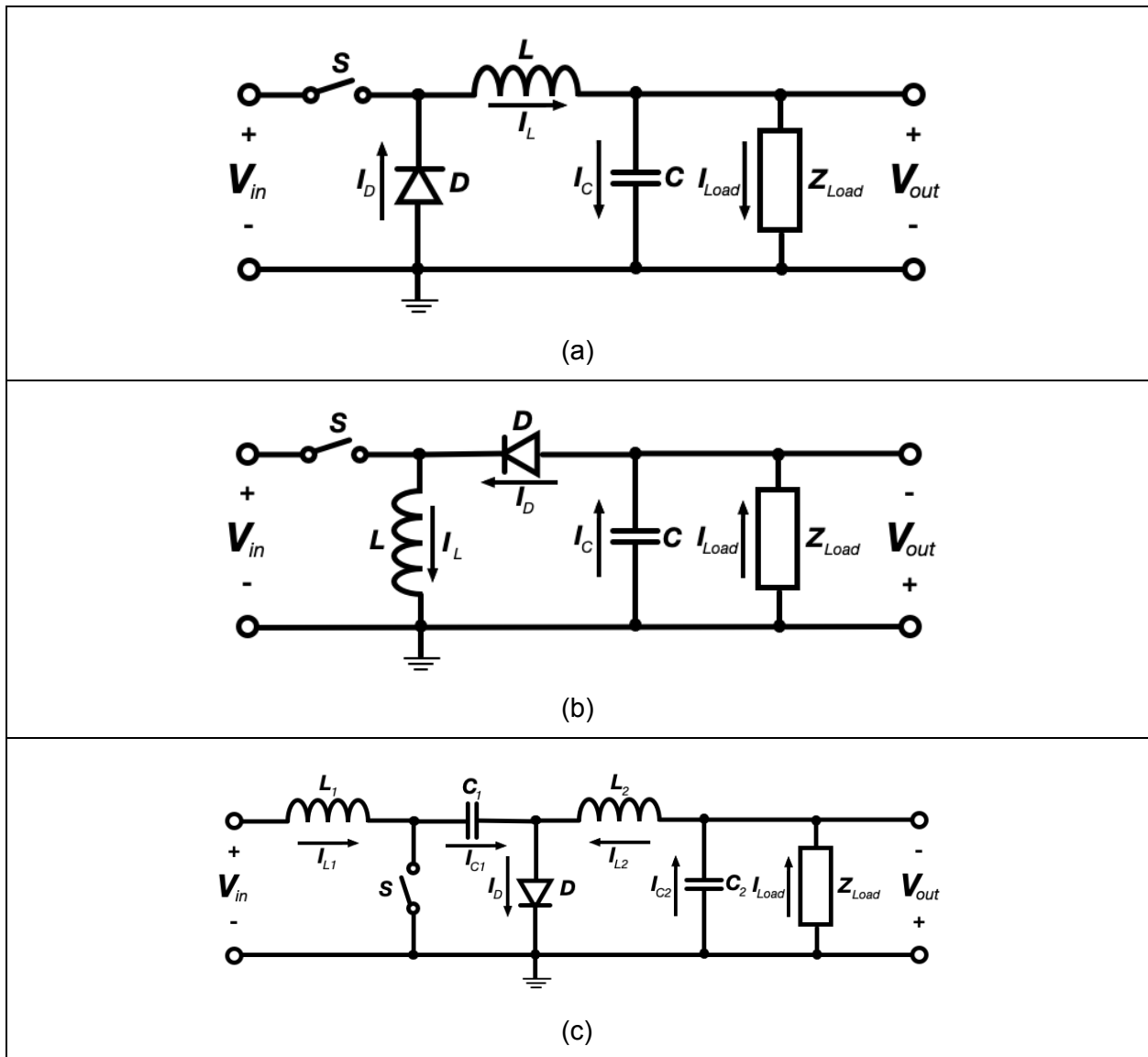


Figure 2-17: Switching power supplies, (a) Buck Converter, (b) Buck-Boost Converter, and (c) Cuk Converter

## **2.3 Sensors, Artificial Intelligence and Neural Networks**

The aforementioned mechanical and electrical components create an RC car capable of motion. In order to induce motion autonomously, however, the vehicle must also leverage sensors and artificial intelligence.

### **2.3.1 Sensors**

Sensors are sophisticated devices capable of converting physical data into electrical signals. For a car to behave autonomously, it must collect data describing its environment. Concerned with other cars, the shape of the road, the presence of pedestrians and the countless other variables that define driving, these sensors must produce responsive and accurate results. While there are many different sensors and variations, the following are the most commonly used in autonomous cars.

#### **RADAR**

RADAR, an acronym for Radio Detecting and Ranging, is a technology based on the transmission and receiving of radio waves. Sent from an antenna transmitter, an electromagnetic wave propagates through space. In the event that the electromagnetic wave comes across an obstacle, some of the electromagnetic wave will reverse directions. Heading into the direction of the receiver (or transceiver), the electromagnetic wave is then analyzed and manipulated. Based on the amount of pulse returned and the time duration of its transmission and reception, the shape and velocity of the surroundings can be determined [44].

#### **LiDAR**

LiDAR, or Light Detection and Ranging, uses measured laser pulses to determine distances of surrounding objects. Very similar in principle to RADAR, LiDAR uses a laser which acts as a transmitter. When laser pulses are sent, some pulses encounter objects which force the pulsed particles to change direction. As a portion of the pulse returns to the laser, it is then read by a scanner. By measuring the quantity of the original pulse that returned and the time duration, the shape and distance of the surrounding environment can be predicted.

In order to generate valuable LiDAR data, the system operates very quickly and creates point clouds. These point clouds, when containing sufficient data, can then be calibrated and constructed into estimated representations of the surroundings. Ultimately, while very similar in principle to RADAR, LiDAR units have experienced greater success and abundance in autonomous driving systems. In light of their abundance, however, LiDAR units are far more expensive than their RADAR counterparts [45].

#### **GNSS**

A Global Navigation Satellite System (GNSS) consists of several constellations of satellites in space continuously transmitting data. Small chip receivers located on Earth can receive said data, and ultimately use those data to calculate its position on Earth. While orbiting the globe, each satellite specifically transmits information describing itself and the whole



constellation of satellites. Providing data such as its unique ID, almanac, ephemeris, and time of transmission, the satellites help GNSS receivers perform direct geolocation. The datum that ultimately enables GNSS receivers to determine their location is the time of transmission.

The receiver compares the time of transmission to its internal time and calculates the total transmission time from satellite to receiver. In this calculation there are many variables taken into consideration such as the effects of the atmosphere on the signal transit time. With the total transmission time, the GNSS receiver can then determine the distance to each satellite because all radio signals travel in a vacuum at the speed of light, 300,000 km per second.

The calculated distance is a pseudo-range representing the receiver's knowledge of the satellite's location. This pseudo-range then creates a sphere around the satellite of possible locations for the GNSS receiver. When a receiver calculates four or more pseudo-ranges from different satellites, the receiver can then perform a process known as trilateration to determine its location on the Earth. Shown in Figure 2-18 (reproduced as is from [46]), the trilateration method is used to determine a GNSS receiver's location by finding an overlapping region of the potential spherical locations [47, 48].

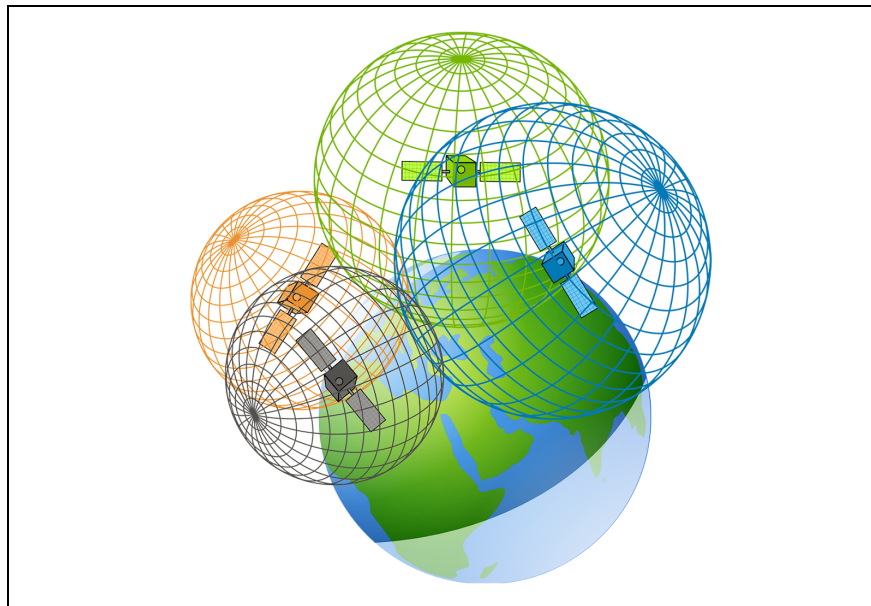


Figure 2-18: GPS Trilateration with Uncertainty [46]

Shown in Figure 2-18 (reproduced as is from [46]) is an example of trilateration with four different satellites. Alone, no satellite provides sufficient data for a location solution. When a GNSS receiver calculates the distance from a single satellite, the receiver is unable to determine its orientation and therefore could be anywhere on the surface of each sphere. As the GNSS receiver collects more satellite data from different satellites, however, an overlapping area is found where the surface of these spheres all intersect and thus an estimated location is determined. Ultimately, with the data provided by four satellites, a GNSS receiver is able to estimate its location anywhere on Earth.

## Camera-enabled Computer Vision

Cameras are a popular choice of sensors for autonomous exploration. A cheap alternative to RADAR and LiDAR, cameras capture images using a matrix of pixels, each with a specific red, blue, green (RGB) value. A direct representation of the ambient environment, a camera with the help of 'Computer Vision' or sophisticated image recognition can view and collect data from the surrounding environment just the way we do with our eyes. [49] As defined by Elon Musk, the CEO of Tesla on the importance of cameras for autonomy - "The whole road system is meant to be navigated with passive optical, or cameras, and so once you solve cameras or vision, then autonomy is solved. If you don't solve vision, it's not solved." [50] Figure 2-19 (reproduced from [51]) shows the application of camera-enabled computer vision by Tesla's autopilot learning feature for one of its self driving cars. In the image, the camera-enabled computer vision indicates which portion of the image is most important for the car to make its next decision.

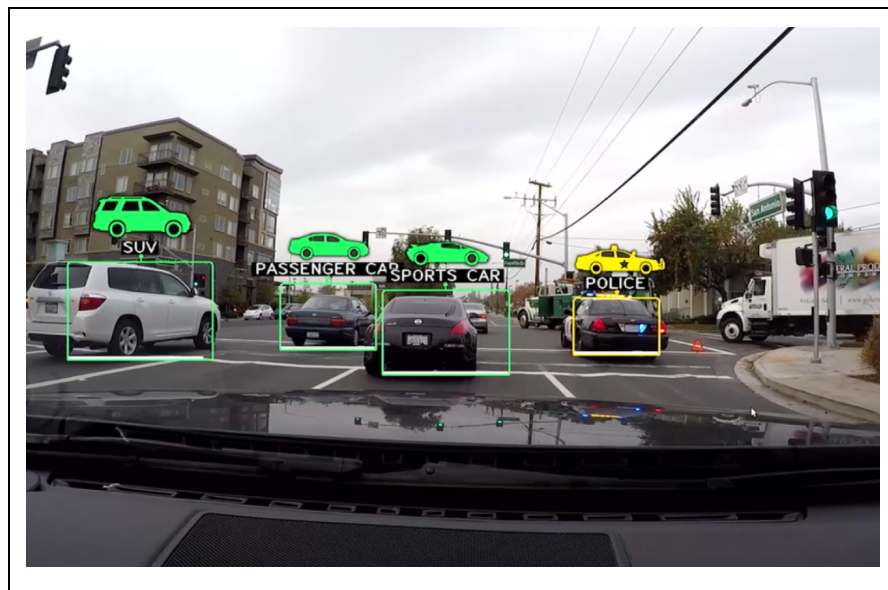


Figure 2-19: Camera-enabled computer vision in Tesla Autopilot [51]

## Inertial Measurement Unit

The inertial measurement unit (IMU) is a device capable of producing independent orientation references. While the modern Inertial Measurement Unit (IMU) has many different designs, it typically contains three sensors each with a 3-axis configuration. Collectively representing 9 Degrees of Freedom (DOF), the 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer allow the IMU to create a location and orientation estimate given initial conditions.

In general, there are two main categories of IMU designs: gimballed and microelectromechanical systems (MEMS). Gimballed systems utilize gimbals to isolate the IMU from the movement of the attached system. Largely mechanical, these systems offer simplicity and accuracy at the cost of size and weight [52]. On the contrary, MEMS IMUs take an intrinsically mechanical action, such as rotation, and generate a corresponding electrical signal.

Often packaged into systems a few millimeters in size, MEMS technology allows for compact sensor implementations, such as those within smartphones.

### **Accelerometers**

MEMS accelerometers have two main categories: mechanical and vibrating element. Mechanically based accelerometers measure the change in position of a known mass to determine the forces and acceleration on that mass. Meanwhile, vibrating element accelerometers, such as surface acoustic wave (SAW) accelerometers, utilize a lever arm with a mass on it. When the mass has a force applied to it, the lever arm's resonant frequency changes. Subsequently, by measuring the change in frequency, the force applied to that mass can be determined.

### **Gyroscope**

A MEMS gyroscope, often called a Coriolis vibratory gyroscope (CVG), contains a vibrating mass that allows for the measurement of rotational acceleration using the conservation of momentum. When undergoing rotation, the gyroscope measures the Coriolis Effect, which states that a mass moving within a rotating system experiences an external force called the Coriolis force. This force is perpendicular to the direction of motion and to the axis of rotation. [53] Ultimately, one IMU typically consists of multiple gyroscopes or multiple-axis gyroscopes to capture a movement with 3 degrees of freedom.

### **Magnetometer**

Most MEMS magnetometers utilize the Lorentz force, which is the force exerted by an electromagnetic field on a moving charge, or current. When a magnetic field is applied across the central beam, the beam moves proportionally in the z-axis. This movement produces a change in capacitance across the finger electrodes on both sides of the beam. The strength of the magnetic field is a function of this change in capacitance [54].

### **Determining Orientation with Inertial Measurement Unit Sensors**

Each sensor within an IMU—accelerometer, gyroscope, and magnetometer—is typically seen in a 3-axis configuration, taking measurements about the device's x, y, and z direction. Using these measurements, the IMU's orientation can be found, most often in terms of roll, pitch, and yaw [55].

Each sensor within the IMU provides data that can be used to calculate the device's pitch, roll, and yaw. A gyroscope measures the device's angular rate. Integrated over time, this measurement can determine the device's angle with respect to an initial orientation [47]. An accelerometer can be used to measure translational acceleration. This measurement can be integrated once with respect to time to produce a 3-axis change in velocity, or twice to produce a 3-axis change in position. As the 3-axis accelerometer also measures the constant force of gravity, it can be also used to determine the direction of Earth's gravity vector. By knowing the direction of Earth's gravity vector, or which way is down, the device's roll and pitch can be calculated [56]. A magnetometer can be used to measure the device's relative magnetic field strength and can be calibrated to determine the direction of magnetic North. The direction of

North can be used to determine the device’s yaw compared to North, also commonly referred to as azimuth or heading [47]. Figure 7-15 below summarizes the measurement and orientation contribution of each IMU component, with  $\Delta$  meaning, “change in”.

Table 2-5: Measurement and Orientation Contribution of each IMU Component

<b>IMU Component</b>	<b>Measurement Uses (3 Axes)</b>	<b>Orientation Contribution(s)</b>
<b>Gyroscope</b>	$\Delta$ Angle	Roll, Pitch and Yaw
<b>Accelerometer</b>	Acceleration $\Delta$ Velocity $\Delta$ Position	Roll and Pitch
<b>Magnetometer</b>	$\Delta$ Angle from Magnetic North	Yaw

### **Tachometer**

A tachometer is a device which measures the rotational speed of a spinning shaft. Used in a wide variety of applications, tachometers often utilize very different design variations. Among the variations, tachometers typically count rotations using either electromagnetic, electronic, or optics principles. Furthermore, tachometers also have fundamental differences including contact, non-contact, frequency-based calculations, and time-based calculations.

Due to the wide variety of system designs, tachometers range in complexity. The most specific tachometers, often interfaced with Arduino microcontrollers, utilize a Hall-Effect, or magnetic field sensor and a magnet. Using the sensor to monitor the location of the magnet placed on the shaft, the rotational speed of the shaft can ultimately be deduced [57].

### **2.3.2 Artificial Intelligence**

With a steady source of reliable data, the next step is to process that data and make decisions. This typically involves the use of artificial intelligence (AI). As defined by Merriam-Webster, artificial intelligence is the capability of a machine to imitate intelligent human behavior [58]. A capability increasingly prevalent in the modern world, AI plays fundamental roles in search engines, email spam rejection, and countless other technological applications. For the purposes of this project, AI is implemented with autonomous vehicles. As it is a relatively high-level concept, AI can be attained using several different technological phenomena. Given the general mission of creating a small scale autonomous vehicle, however, neural networks will be used.

### **2.3.3 Neural Network**

Created to loosely model human neural function, Artificial Neural Networks (ANN) are computational programs comprised of several layers possessing interconnected nodes, or

neurons. Consisting of an input layer, an output layer, and at least one hidden layer, the ANN attempts to generate a rational output based on a large influx of input data.

Seen in Figure 2-20 (reproduced as is from [59]), each layer of the ANN is comprised of interconnected nodes. Acting in a binary state (e.g. “On” or “Off”), each node relies on an activation function which determines whether or not the node should fire. Propagating throughout the network, activation and inactivation of nodes on the input layer influences the activation or inactivation of nodes within the first stage of the hidden layer. Continuing throughout all of the hidden layers, the ANN then concludes with an output layer with appropriately activated neurons. Reflective of the ANN’s “decision”, this output layer serves as the system’s response to the aforementioned input data. In its simplest form, neural networks are interconnected linear equations that result in reducing complex data down to relatively simple outputs.

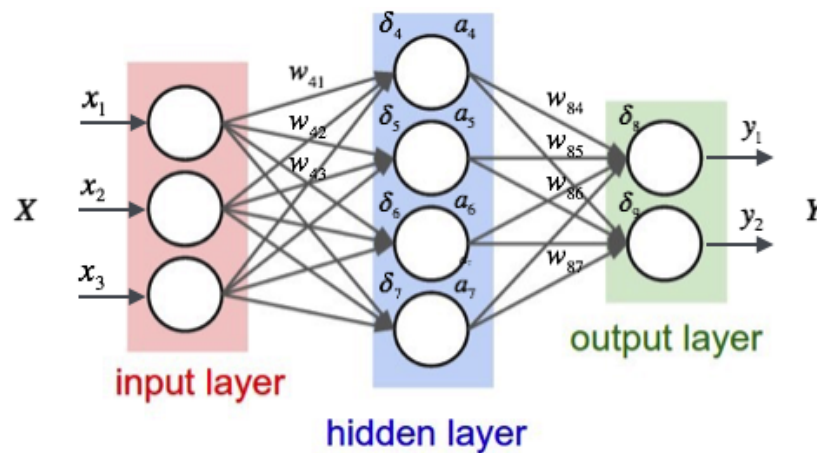


Figure 2-20: Internodal connections within an Artificial Neural Network. Reproduced as is from [59].

When creating a neural network, the program must be “trained.” Provided input data and the corresponding ground truth, the network tries different combinations of weights between each of the nodes to identify which configuration arrives at the most accurate conclusions. Once run through the training data, a set of validation data is used to test how well the network is trained. While the idea of training occurs within all neural networks, there are distinctions based on the two types of networks used: supervised and unsupervised. For example, a “backpropagation neural network” (BPNN) [60], the most common supervised network design, begins with an initial guess of the weights. As per its name, it then propagates back through the calculation, determining errors and reweighting values on certain nodes. Continuously iterating through the nodes, the BPNN gradually optimizes the weight parameters. Ultimately by using a gradient descent algorithm, the BPNN determines the global minimum along the steepest error curve which theoretically depicts the lowest error rate of weights [60].

Careful to avoid overfitting, the fundamental goal of training a network is to produce output appropriately responsive to input data. Once the training is completed, the network can be switched to a forward only propagation state where it is used as a purely analytical tool to generate correct solutions to the given problem. In the case of this project, the AI will be trained

to react to situations that it could encounter on a race track in order to teach the network to drive a scale car.

## **2.4 Existing Autonomous Solutions**

One existing autonomous solution for 1:10 scale RC cars is the Donkey Car [4]. An open source module, the Donkey Car can be calibrated to drive any RC car. The system design heavily influenced the design used by this team. The Donkey Car module consists of an Arduino, Raspberry Pi, and a Raspberry Pi Camera Module. With these pieces of hardware and an incredible amount of open source contribution from enthusiasts, the Donkey Car system has been incredibly versatile for getting RC cars to self drive. The Donkey Car uses several Python packages for creating, training and running an ANN onboard the Raspberry Pi to drive any car at a reasonable speed. While the Donkey Car exists as a template to prove that self driving RC cars are possible and attainable, it is, however, unable to produce autonomous solutions for any track.

Another prevalent self-driving solution for cars comes from Tesla. A major proponent for self driving capabilities over the previous decade, Tesla utilizes large sensor suites to collect environmental information. With current designs using 8 external cameras, 12 ultrasonic sensors, and a radar, Tesla performs high scale sensor fusion to produce autonomous driving capabilities. Ultimately, while Tesla produces work far beyond the scope of this project, it acts as an additional reference for the goal of autonomous navigation.

Lastly, several previous MQPs have create autonomous vehicles. The three of the most relevant projects were the Semi-Autonomous Wheelchair Navigation Project and the Autonomous Campus Modular Platform. The Semi-Autonomous Wheelchair Navigation Project, involved creating an autonomous wheelchair for people who are unable to use a traditional joystick controller [61]. The team used several sensors, including LiDAR, ultrasonic distance sensors and IR sensors to make the wheelchair autonomous. For the autonomous campus mobility platform, the team designed a long board that used LiDAR to autonomously navigate around campus [62].

## **2.5 ME 4320 Advanced Engineering Design**

Currently, Worcester Polytechnic Institute (WPI) offers a course within the Mechanical Engineering department titled Advanced Engineering Design. Known by its course number, ME 4320, this course is instructed by Professor Pradeep Radhakrishnan. Among many topics, the course challenges students to create a 1:10 scale RC car. Specifically, as per the course design guidelines, the students create a rear-wheel drive car with a servo-controlled steering mechanism and a custom gearbox.

Aside from the fundamental design requirements, the students are allowed to construct their scale cars, however, they please. The cars designed by students vary vastly in construction, component selection, overall size, linkage design, gearbox design, and material used. Figures 2-21 to 2-23 below, illustrate some of the cars that have been produced.

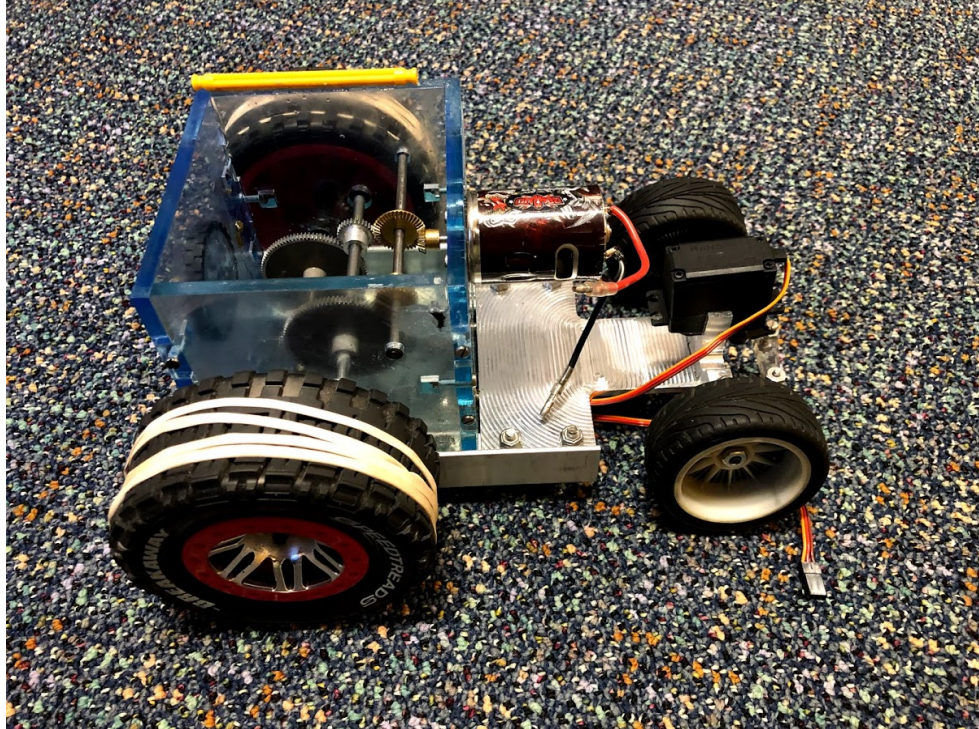


Figure 2-21: A car designed by Vinve Lucca, Joe Stapleton, Peter Nash, Colin Saunders, Zach Fischer, Jake Chiudina featuring a billet machined chassis and laser cut gearbox housing.

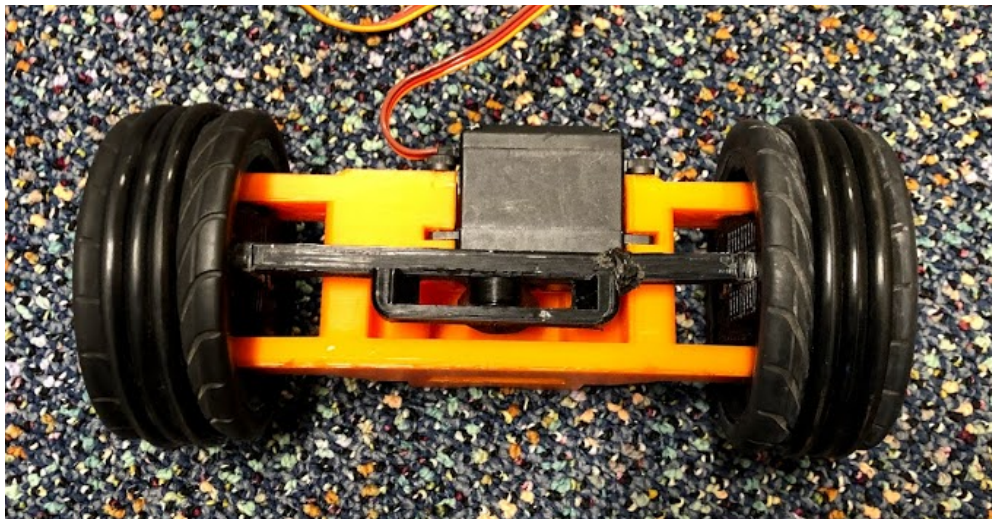


Figure 2-22: A steering linkage designed by Derek Kruzan, Tim Esworthy, Walter Kwiecinski, Jessica Elder, Nolan Bell and Xavier Little, featuring entirely 3D printed components.

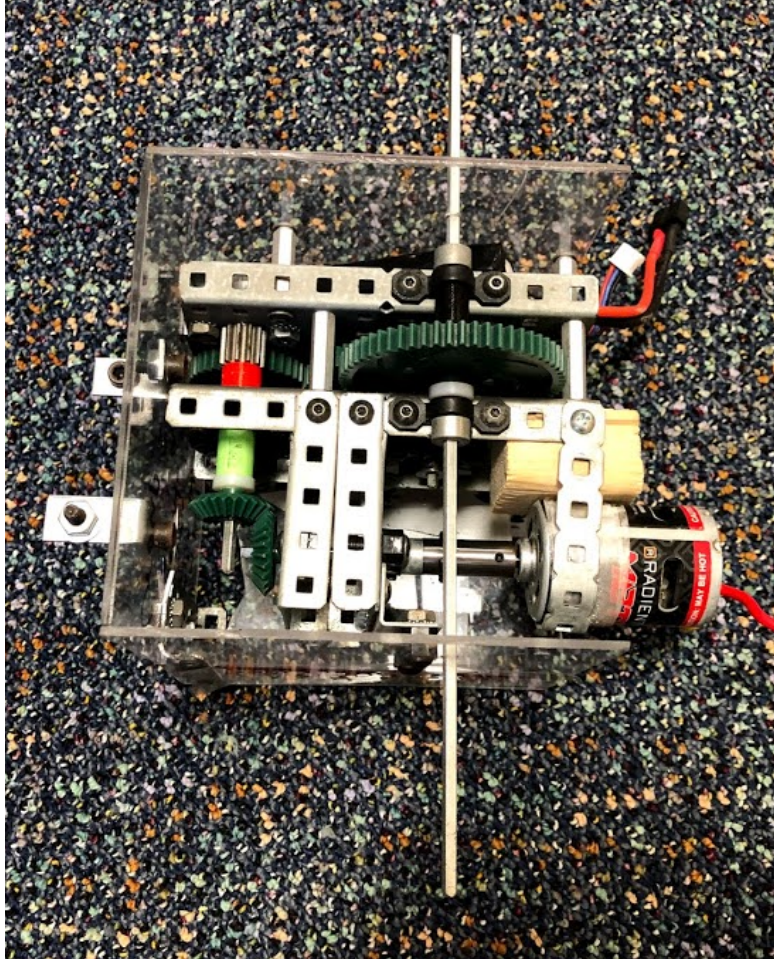


Figure 2-23: A gearbox designed by Amanda Richards, Michael Wilkinson, Tim Bill, Jamie Stephen, and Chris Mayforth, constructed out of mostly VEX gears and axles.

As seen in the figures above, the cars vary dramatically between teams. For example, in the transmission systems, some teams use plastic gears, while some teams will use metal gears. Some teams will have multiple stages in their gear reductions while some use a single stage. In the steering linkages, most linkages, although typically planar, vary dramatically in size, design and performance. Furthermore, some linkages have large dead zone areas where the linkage will not respond despite movement of the input linkage, while in other linkages the steering is responsive. Additionally, the range of motion of the steering linkages varies as well.

As discussed above, the students are responsible for designing, analyzing, and manufacturing their cars. Once the students have completed these steps, they compete in a race to evaluate the performance of the car. The race is designed to act as a tool to differentiate the high performing cars from the low performing cars. The race track typically consists of ramps, gradual turns, hair-pin turns, and tunnels. Unfortunately, while the race may appear to be a great way of determining car performance, its results can be clouded by the various skill levels of the drivers. In order to combat this, the implementation of an autonomous module could mitigate driver bias and provide more parity in determining the capabilities of the designed cars.



Ultimately, this chapter provided an overview of Radio Control cars and their assembly. Additionally, this chapter served as a literature review for the basic sensors and neural networks necessary for autonomous navigation. The chapter culminated in an overview of the ME4320 course, Advanced Engineering Design. Here, the fundamental project goals for improving the course were reiterated. The next chapter discusses the general methodology of the project. Specifically, the following chapter illustrates the iterative methodological process used to satisfy all project goals.

### 3.0 General Methodology

While subsequent chapters detail the specific mechanical, electrical, and neural network design, this chapter first provides an overview of the general methodology. Acknowledging all of the work conducted, the general nature of the project can be divided into four stages: (i) research, (ii) proof of concept (POC), (iii) car development, and (iv) neural network development.

During the research phase, the team sought to answer three fundamental questions. First, what additional design requirements could be added to the ME4320 car project? Second, is it possible to make the ME4320 cars autonomous? Third, what kind of real-time data could be captured in order to monitor and aid the operation of the cars? While each question undergoes more elaborate analysis in the following chapters, preliminary research led to the following conclusions. First, the implementation of a front suspension, rear suspension, and rear differential could be a possible addition to the ME4320 car design requirements. Second, a neural network leveraging input from a camera could feasibly make the ME4320 cars autonomous. Third, the car orientation, temperature, and motor RPM could be important real-time data relevant to the operation of the car.

Having conducted preliminary research, the next step was the creation of a POC. Among many questions, the primary uncertainties regarding the capabilities of the car's control system and AI included: what hardware is necessary for controls and autonomy, if camera and receiver training data could be collected, if a neural network could be created and trained, if a neural network could control a test car, and if a neural network could make real-time decisions using images from an input camera. While explicitly detailed in Chapter 4, each of the aforementioned questions was successfully validated using a test car. Consequently, successful fulfillment of the project requirements became more plausible.

While developing the POC, the initial car development stage simultaneously began. Taking into account the new design additions (i.e. front suspension, rear suspension, rear differential), an initial car was designed and manufactured. Given the importance of AI development and testing, this initial car was designed and manufactured as quickly as possible. While the original car was quickly manufactured, however, continued car modifications and iterations were made throughout the remainder of the project.

Immediately following the initial car design and assembly, the neural network development stage began. For the remainder of the project, both the car and neural network development stages ran in parallel. Ultimately, the goal of the neural network development stage was to create a neural network that could drive around a track. This involved collecting data, generating neural network models, testing the models, altering test parameters, and then refining the process.

Ultimately, as the project was highly interdisciplinary in nature, specific design choices regularly impacted the design and performance of seemingly unrelated car features. Consequently, the methodology of creating a scaled autonomous car was highly iterative. This was most significant between the car design stage and the neural network development stage.

Despite the volatile nature of the design process, however, the flow chart shown in Figure 3-1 depicts the general methodology described above.

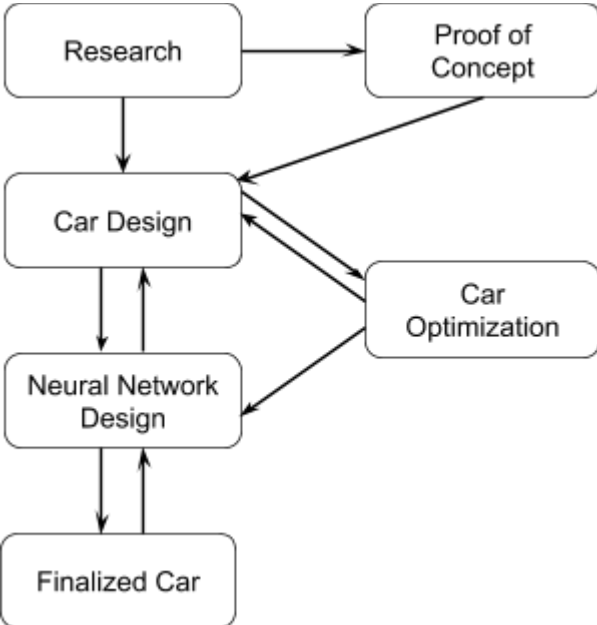


Figure 3-1: General Methodology Flow Chart

Having established the general project procedure, the next chapter discusses the preliminary work performed on the test car given to the team. Specifically, the next chapter discusses the proof of concept nature of the test vehicle.

## 4.0 The Test Car

This chapter discusses the test car used to develop a proof of concept (POC) for the artificial neural network (ANN) with a camera. As the principal use of the test car was the validation of the POC, little was done to modify the underlying car design. Instead, the majority of the car modification was done in the effort to make it autonomous. This chapter is divided into seven sections. Section 4.1 discusses the test car that was provided to the team. Section 4.2 discusses the hardware that was selected for POC testing. Section 4.3 discusses the mounts that were made to add the new hardware components to the test car. Section 4.4 discusses the procedure to control the test car using the POC hardware. Section 4.5 discusses the powering of the new hardware. Section 4.6 discusses the problems the team encountered while implementing the AI to the test car. Finally, Section 4.7 illustrates the results of the POC following the test car experimentation. For reiteration, the fundamental POC questions from Chapter 3 can be seen in Table 4-1 below.

Table 4-1: Proof of Concept Questions for Test Car

1. What hardware is necessary for the control and autonomy systems of the RC car?
2. Can camera and receiver training data be collected?
3. Can a neural network be created and trained?
4. Can a neural network control the test car?
5. Can a neural network make real-time decisions using images from an input camera?

### 4.1 Donation of a Test Car

For initial research and autonomous testing, a car previously designed from a Worcester Polytechnic Institute (WPI) class was used. Courtesy of previous students from the Advanced Engineering Design class, the vehicle was a 1:10 scale car with remote control capabilities. Seen in Figure 4-1, the car modeled the specifications for the Advanced Engineering Design course (ME4320). Consequently, the car was rear-wheel drive and had no front or rear suspension. Despite the minimalist design, however, the car could drive and was capable of providing preliminary testing data.

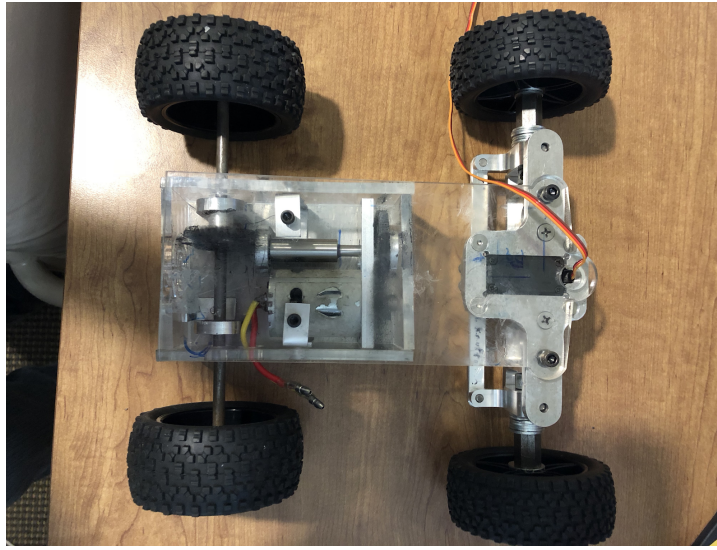


Figure 4-1: Test RC Scale Car. created by Alan Hunt, Colin Maschack, Aimilios, Tachiaos, Jake Halverson, Jesse Kaufman, Marissa Ford, Kyle Whittaker, Howard Vance, Frank Bucknor, and Edward Crofts.

## 4.2 Preliminary Hardware

In order to ultimately perform autonomous navigation, the test car also needed to have the appropriate hardware. Specifically, the test car required hardware for sensory data collection, car control (i.e. servo, ESC), and AI computation. In addition to the required tasks, the selected hardware had to satisfy general constraints. Among the many constraints, the hardware had to interface correctly, have reasonable power requirements, be reasonably scaled to fit a 1:10 scale car, and be reasonably cheap.

### 4.2.1 Sensor

In order to collect the car's sensory data, many different sensors were considered. The considered sensors included: camera, ultrasonic sensor, LiDAR, RADAR, and WiFi localization. As a fundamental goal of the project was to produce an autonomous system that did not rely on mapping or localization, the use of WiFi beacons was immediately rejected. Furthermore, while ultrasonic, RADAR, and LiDAR are commonly used on full-sized vehicles, these sensors fit more appropriately with Simultaneous Location and Mapping (SLAM) or sensor fusion systems. As these systems rely on high volume point clouds to reproduce the surrounding contours, the systems rely on computationally expensive mapping for accurate driving.

Ultimately, the one sensor which best accommodated the constraints and design goals of the project was the camera. Cheap, easy to integrate, and capable of non-SLAM applications, the camera theoretically provided all data necessary for simple autonomy. Furthermore, as the camera is the sole sensory device used by Donkey Car, it was theoretically sufficient for the fundamental project goals. The Amadget Wide Angle Fish-eye Camera Module, discussed in

the next chapter, was used as the camera for the test car. Specifications of the camera are in said chapter.

#### **4.2.2 Car Control**

The next piece of hardware selection was the microcontroller used to control the servo and ESC. Using the Donkey Car as a reference, the Arduino was selected. Capable of interfacing with most microcomputers and fitted with sufficient digital IO (Input/Output) and pulse modulation pins, the Arduino was a cheap and effective means of controlling the car performance. While there are many different Arduino boards, the Arduino Mega was selected. Equipped with 54 digital IO pins and 4 UART (Universal Asynchronous Receiver and Transmitter) serial ports, this model was well equipped to adapt to the various needs of the project [63].

#### **4.2.3 AI Computation**

The last hardware requirement was the selection of a microcomputer capable of operating neural network models. Among the many options, the considered microcomputers included: the Raspberry Pi 3b+, Odroid-XU4, Nvidia Jetson Nano, and the Intel Compute Stick STK1. While each system was capable of running a neural network model, there were several design variations differentiating the compatibility of each system. The Intel Compute Stick, while powerful, lacked the interfaces necessary for autonomous navigation. Containing only two USB ports and no other pins, the Intel Compute Stick lacked adaptability [64]. The Jetson Nano, although equipped with the necessary interfaces and a GPU, was a more expensive solution [65]. Furthermore, much of the additional computational power provided by the Jetson would be unnecessary for the applications. The Odroid-XU4 was a relatively cheap and powerful system also considered [66]. Unlike the Jetson Nano and Intel Compute Stick, the Odroid did not have any fundamental flaws. Despite the strengths of the Odroid-XU4, however, the Raspberry Pi 3b+ was used [2]. The cheapest option with sufficient processing and interfacing capabilities, the Raspberry Pi could handle all necessary tasks. Furthermore, the strong Raspberry Pi community and its use in the Donkey Car were fundamental reasons for the selection of the Raspberry Pi.

Ultimately, while the selected components were diligently selected, their proof of success was still unknown. Consequently, the experimentation discussed in the following sections played a pivotal role in determining whether or not the selected hardware would be adequate for the control and AI system of the real car.

### **4.3 Mounts**

To accommodate the camera, Arduino, and Raspberry Pi, the test car had to be appropriately modified. First, in order to implement the camera, one major design consideration was the inclusion of variable mounting locations in order to adequately test the impact of the camera angle on neural network performance. In order to produce an interface capable of

housing the variable camera placement, the original battery, electronic speed controller (ESC), and receiver mounts were replaced with a pegboard-style cover (Figure 4-3). The camera mount, attaching the camera to the pegboard, was then designed as an assembly with three sections (Figure 4-3a). The first section, as seen in Figure 4-3c, attached the assembly to the pegboard. Once secured, pole segments could be added or removed to the base to change the camera height. Finally, the third piece held the camera at a specified angle (Figure 4-3b). Ultimately very easy to manufacture and modify, the development of this pegboard camera mount system was highly effective and versatile.

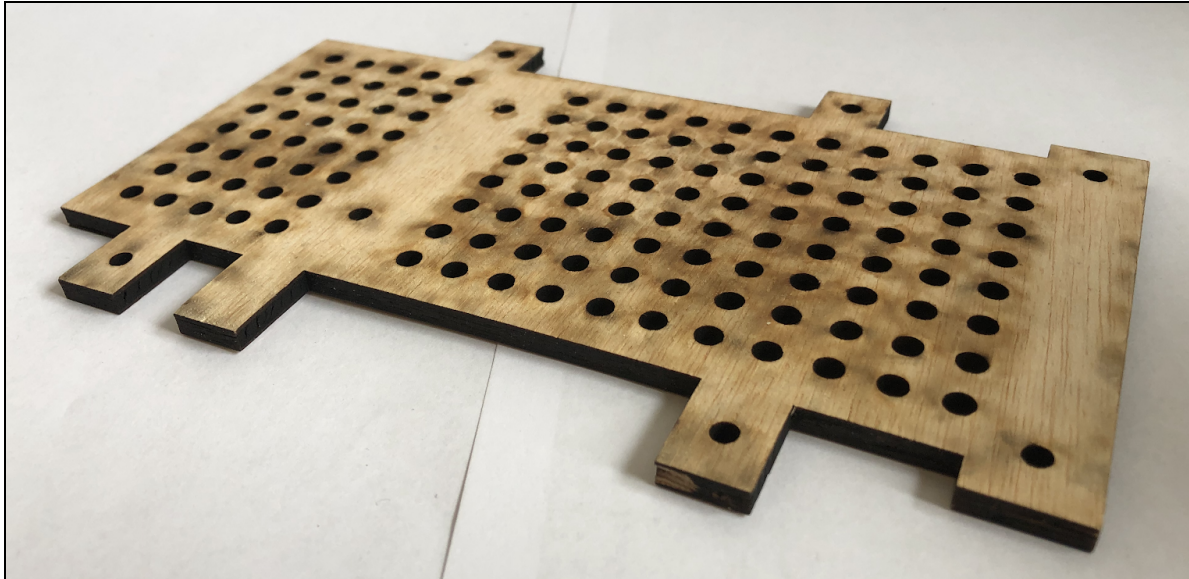


Figure 4-2: Pegboard-style cover so the camera can be mounted at several locations.

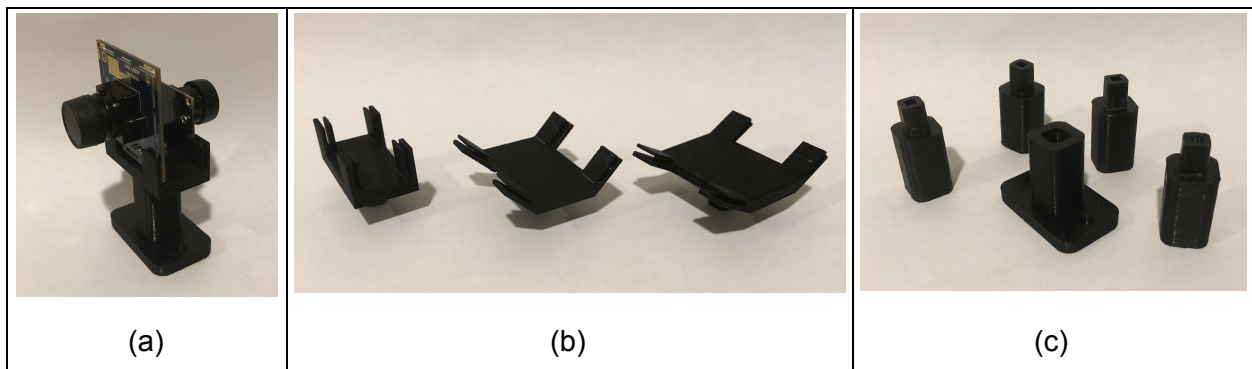


Figure 4-3: Camera mount assembly (a) with interchangeable camera to pole mounts (b), a base to pegboard mount (c), and pole segments.

With the camera appropriately mounted, the next step was the implementation of the Arduino and Raspberry Pi. Using the same pegboard illustrated in Figure 4-2, the Arduino and Raspberry Pi were properly fitted as shown in Figure 4-4. Unfortunately, space limitations on the pegboard left little room for the battery, receiver, and ESC on the test car. Consequently, all three pieces were hung from the side of the car using the mounts shown in Figure 4-5. The test

car, with all the components on the car, is shown in Figure 4-6. While this design modification ultimately reduced the turning radius by approximately 15 degrees, the design additions integrated all of the hardware necessary to test the POC. As such, the decrease in mechanical performance was tolerable considering the test car was only meant to provide a POC regarding the functionality, design and performance of the AI.

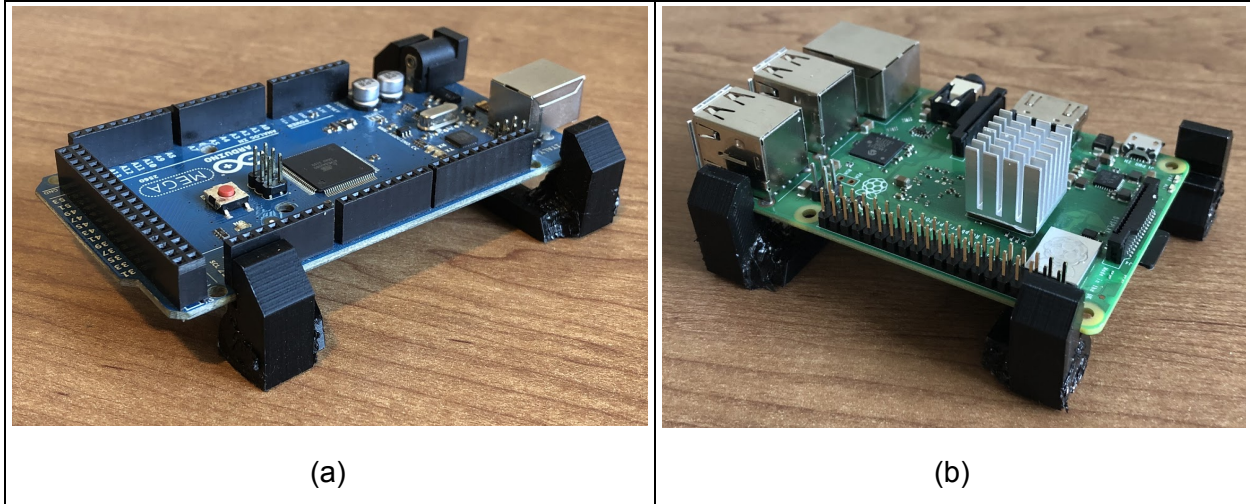


Figure 4-4: Arduino (a) and Raspberry Pi (b) pegboard mounts

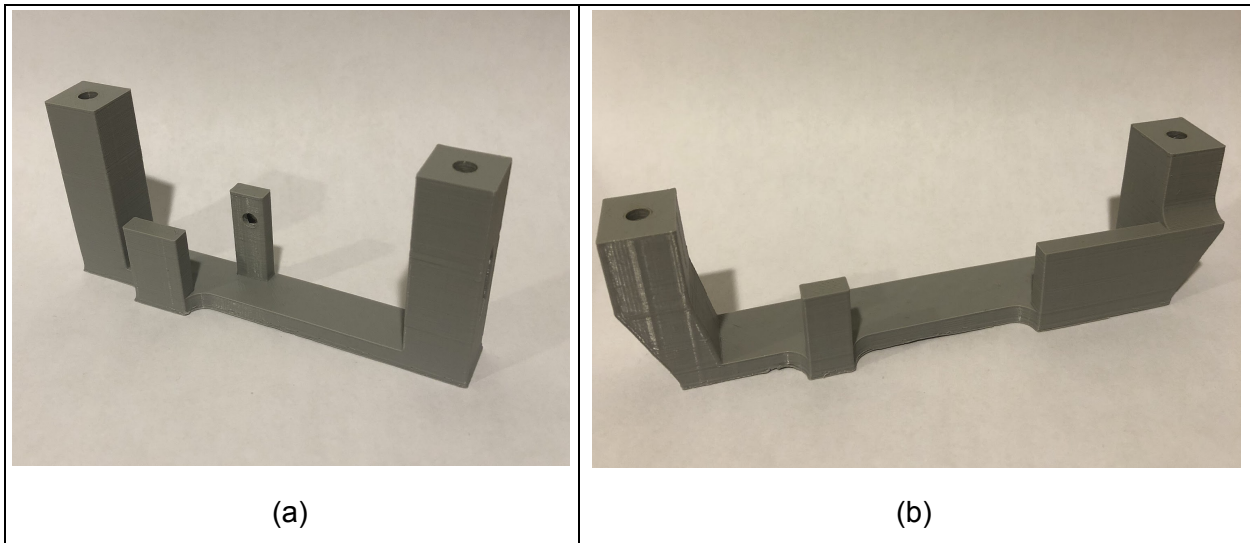


Figure 4-5: ESC and Receiver mount (a), and Battery mount (b)



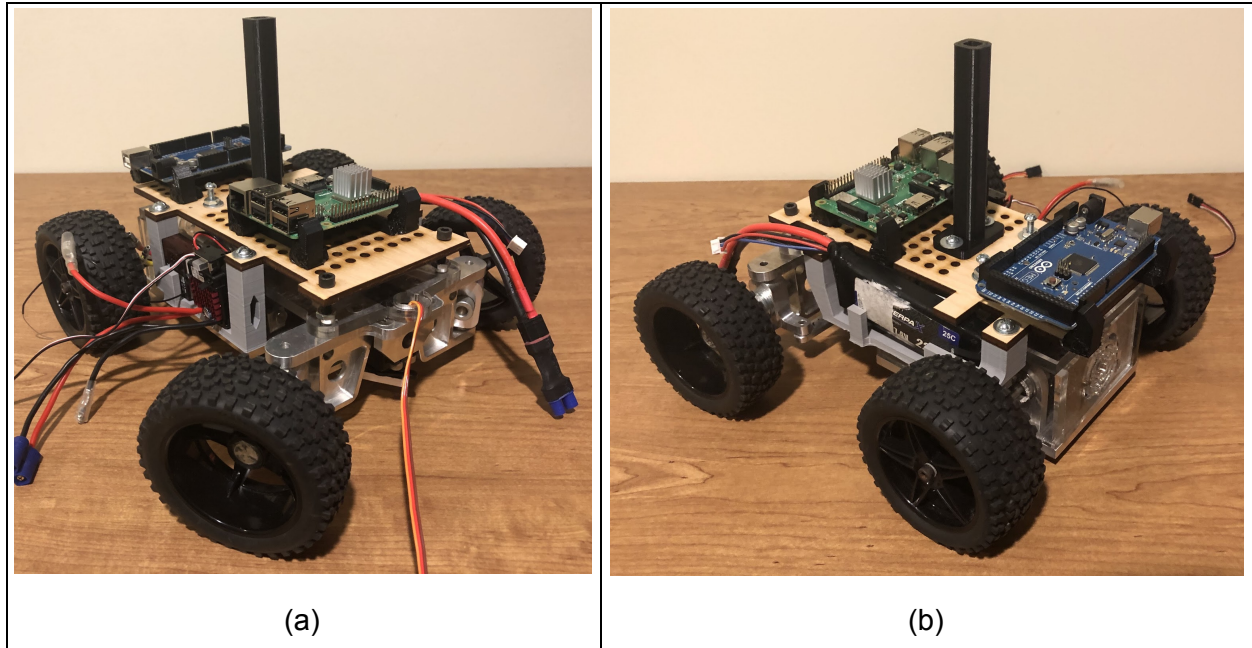


Figure 4-6: Test car modifications, Front right (a) and back left (b) views of test car with mounts and components.

#### 4.4 Communication

For the neural network to make correct decisions, it needed to be trained from the data of realistic case scenarios. Naturally, every situation can be decomposed into a decision matrix containing the necessary information for the neural network to act appropriately. During training, the neural network was then provided the correct response known as the output matrix. This meant that training the car's neural network required input from the camera (the decision matrix) and the corresponding output to the RC car (the output matrix). Both of these matrices could be created by driving the RC car, recording the input from the camera, and recording the output of the receiver. For this reason, one of the goals of the POC was to validate that the team could successfully collect the output from the receiver.

The output from receivers is a pulse modulated wave (PMW) with a duty cycle ranging from 5% to 10% (Figure 4-7) and a period of approximately 20ms. In order to determine the output from the receiver, the duty cycle needed to be measured. As discussed in Section 4.2.2, the team chose an Arduino to read the output from the receiver and be a bridge between the neural network and RC car. To read the signal from the receiver, the team connected pins 22 and 23 of the Arduino to channels two and three on the receiver (Figure 4-8). The following, two methods were developed to measure the PWM from the two channels.

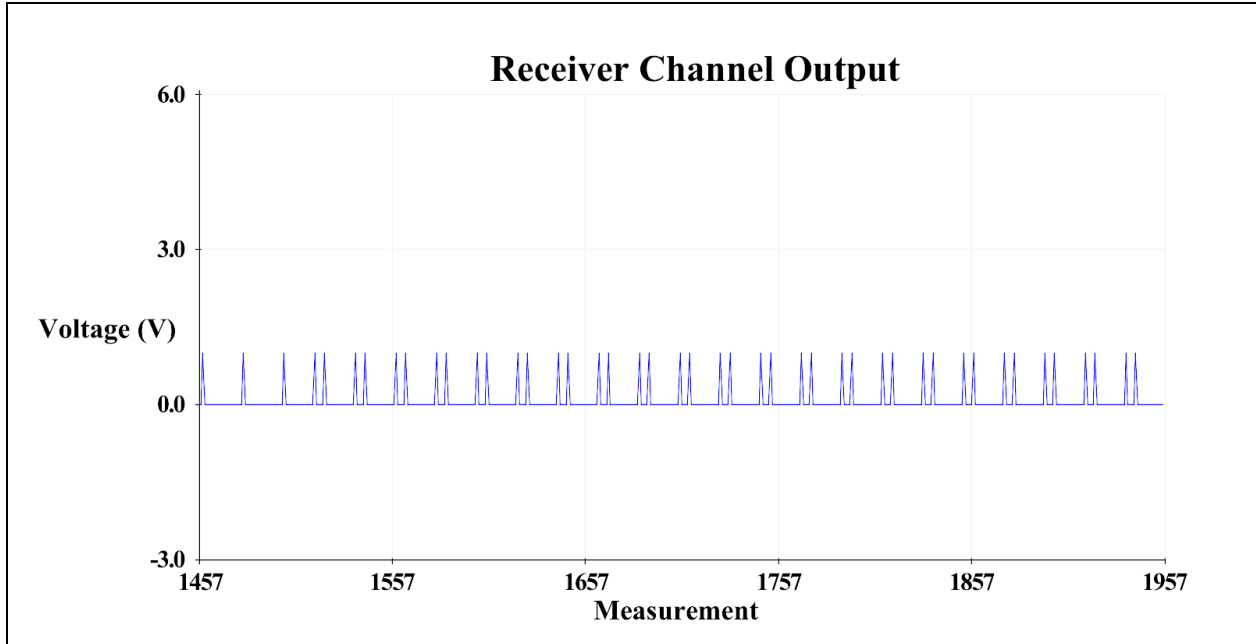


Figure 4-7: Voltage of receiver output signal (measured with Arduino)

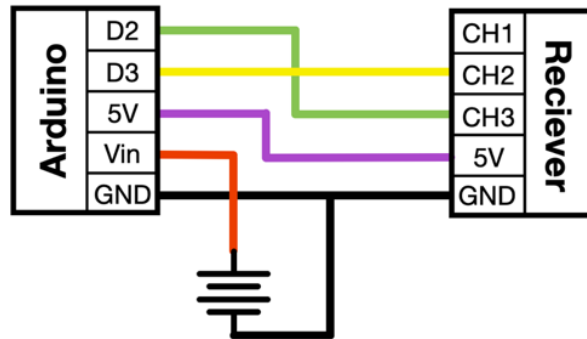


Figure 4-8: Arduino and Receiver Schematic. Note the colors within the figure are to better illustrate the individual wires.

The first method looked for state changes at each input pin. If there was no change, the program would begin to execute its other functions. If there was a change, the program would update the duty cycle. This method was successful, as shown in Figure 4-9, which shows the duty cycle changing per every change in state of the input pin. When a second duty cycle pin was added, however, the amount of noise created by time missed between state measurements became significant. As illustrated in Figure 4-10, the resulting noise made this method not viable.

The second method measured the pulse of the duty cycle. Instead of continuing onto other computations between measurements, the program stopped to measure the exact rise and fall times of the next pulse before moving on. This method was far more accurate than the original with little to no noise (Figure 4-11). While the noise was minimized, the time it took to

measure the duty cycle increased significantly. Figures 4-12 and 4-13 show the time it took to measure a duty cycle with method one, approximately 16ms (less than one duty cycle), and the time it took to measure a duty cycle with the second method, approximately 40ms (two duty cycles). This was important because there were two pins that needed to be measured, which caused each loop in the Arduino to take approximately 80ms, reducing the responsiveness of the program. Ultimately though, the improved accuracy was determined to be worth the extra time requirements because the program maintained a relatively fast response time.

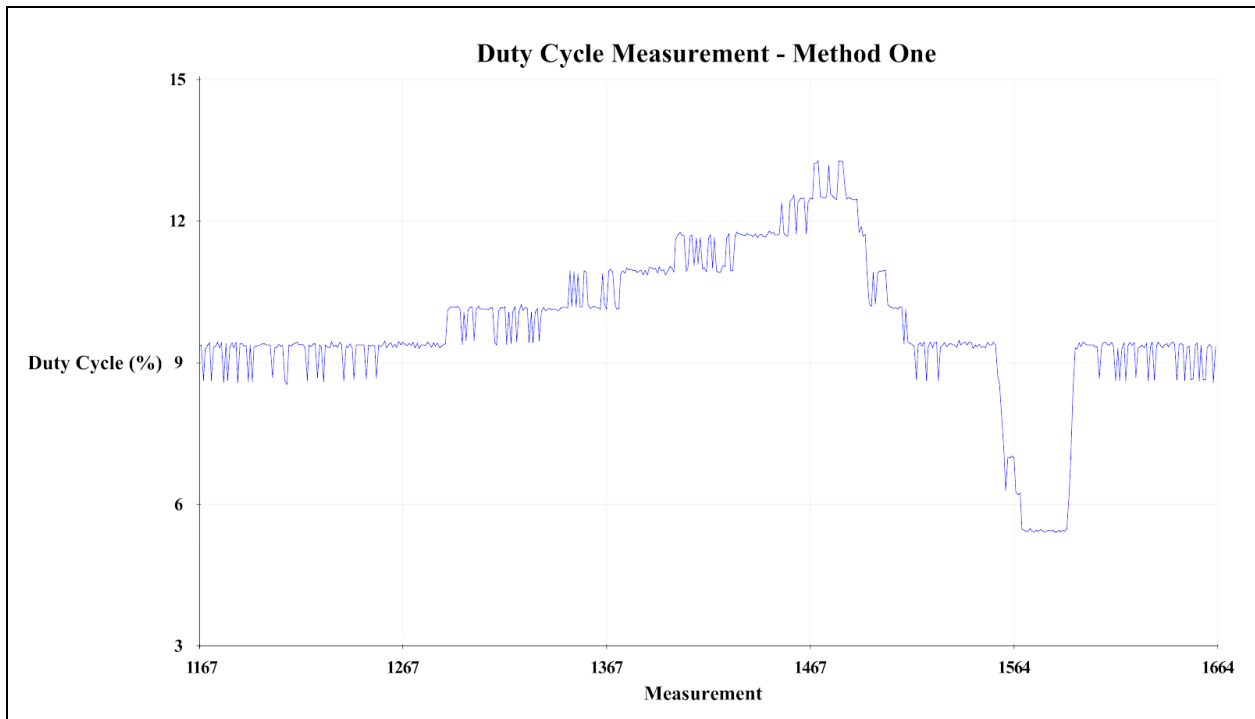


Figure 4-9: Method one duty cycle measurement with one pin measurement

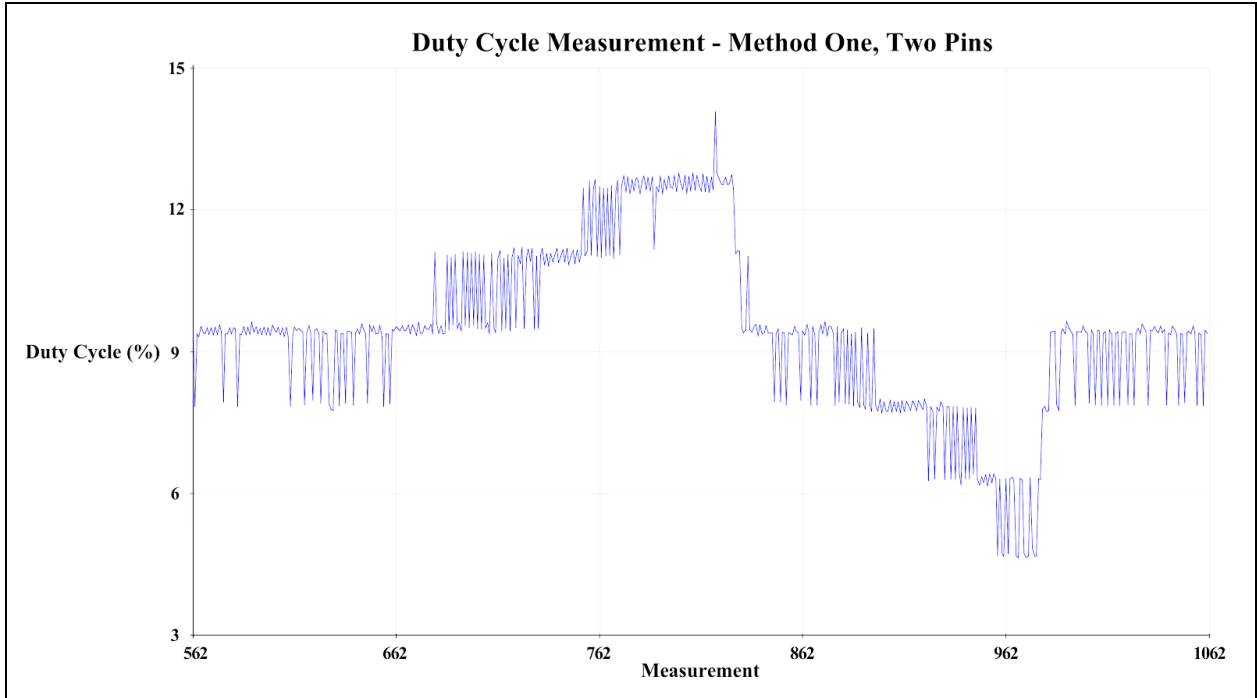


Figure 4-10: Method one duty cycle measurement with two pins

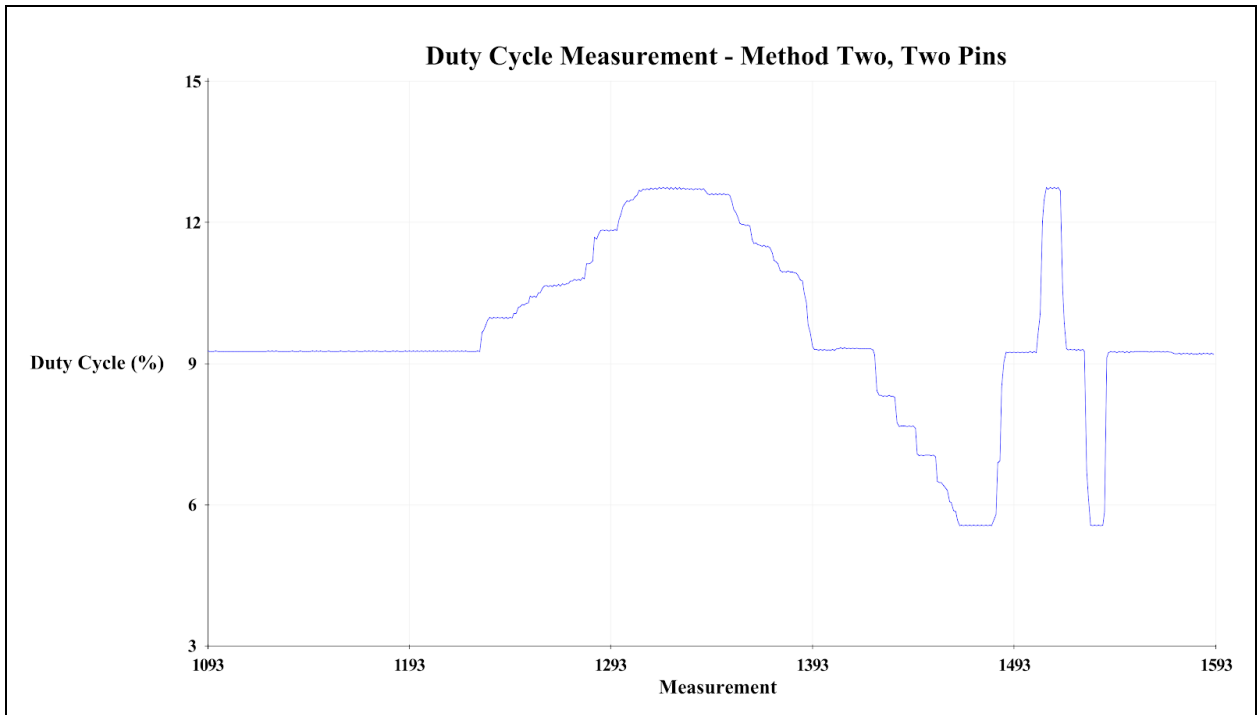


Figure 4-11: Method two duty cycle measurement with two pins

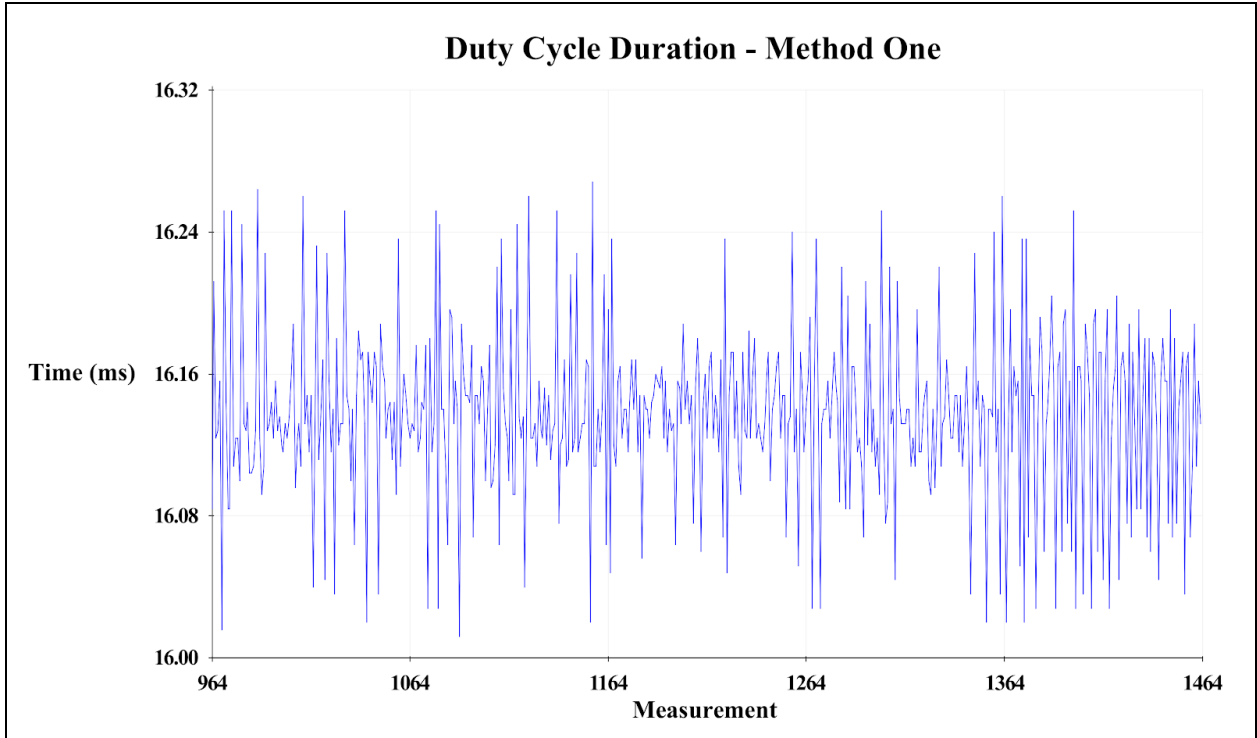


Figure 4-12: Duty cycle measurement time of one pin using method one

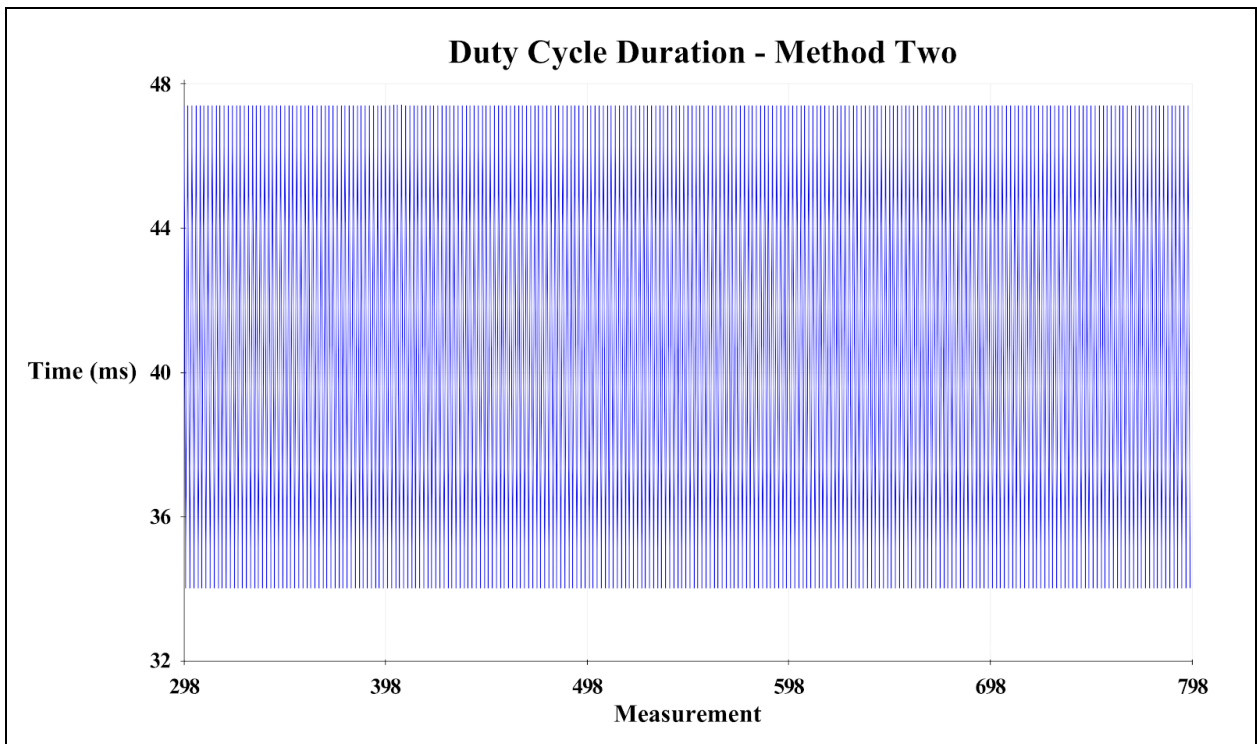


Figure 4-13: Duty cycle measurement time of one pin using method two

Once an accurate method of measuring the duty cycle from the receiver had been completed, the team needed to show that the Arduino could successfully control the RC car. This was done by creating a program titled “Conductor”. The program began by measuring the duty cycle from the receiver (Figure 4-11). Then the duty cycle was converted into degrees, where 0 degrees is a 5% duty cycle and 180 degrees is a 10% duty cycle [48]. Lastly, the Servo library, a standard Arduino library, was used to output a PWM signal corresponding to the computed degree. To send the signal to the ESC and servo, the team connected the ESC and servo signal pins to pins 4 and 5 on the Arduino, respectively (Figure 4-16). Both of these pins could produce a PWM signal. Figure 4-14 shows the duty cycle percentage that is read from the receiver and Figure 4-15 shows the output PWM to the servo from the Arduino.

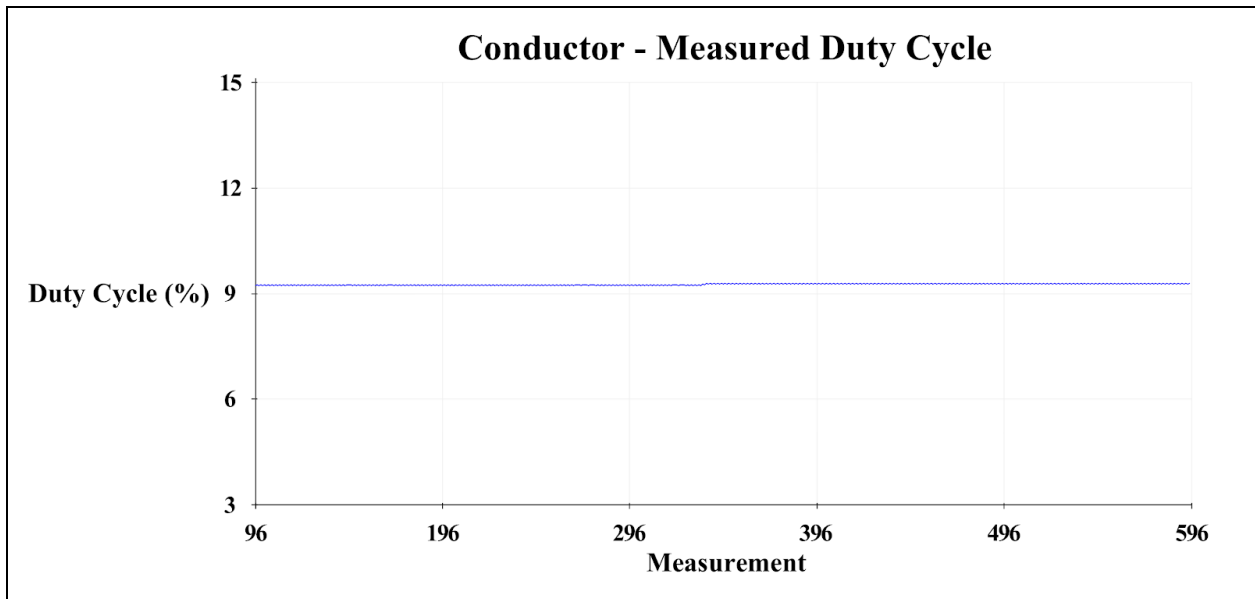


Figure 4-14: Conductor, measured duty cycle of the input signal

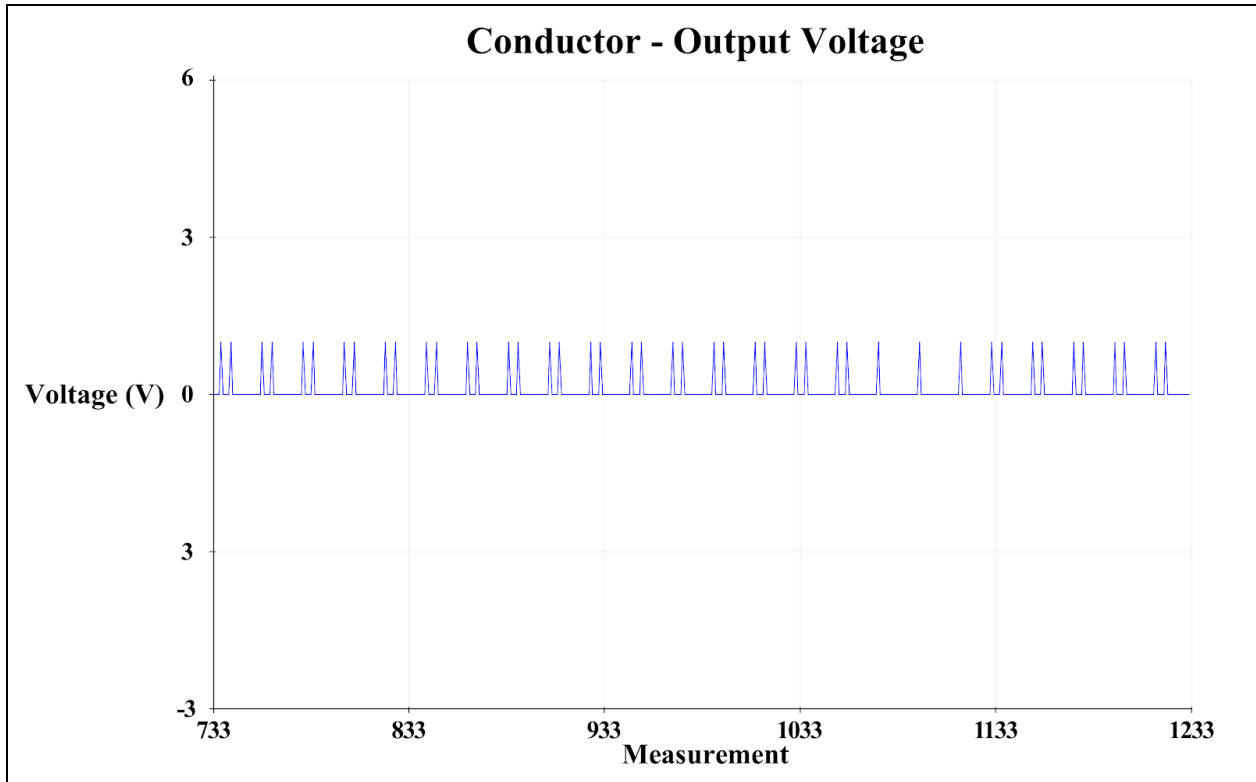


Figure 4-15: Conductor, the output signal from the Arduino

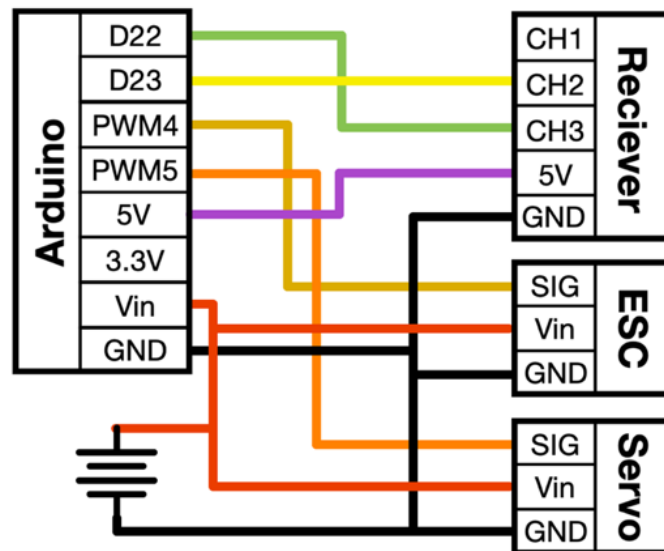


Figure 4-16: Conductor circuit schematic. Note the colors within the figure are to better illustrate the individual wires.

After the Arduino could successfully measure the input from the receiver and control the test car, the team created a program that would allow the neural network to control the Arduino through a serial connection. This was done by replacing the receiver input with commands read from the serial connection with the Raspberry Pi. As expected, the Arduino capably read the instructions sent by the Raspberry Pi.

## 4.5 Power

Once fully modified, the test car contained two Raspberry Pis, two Arduino Megas, an ESC, a servo and a receiver. The first Raspberry Pi and Arduino pair was intended to monitor the car's sensory systems (Chapter 7), the second Raspberry Pi and Arduino pair was intended to be used by the ANN (Chapter 8), and the remaining components were needed for the test car to have the required functionality of an RC car. In order to power these components, a 7.4V, 2.2Ah, 25C battery that came with the test car was used. While all other components were compatible, the Raspberry Pis were unable to connect directly to the 7.4V LiPo battery [68]. Instead, the Raspberry Pis utilized a buck converter which stepped the 7.4V down to 5V. In order to obtain a functional power regulator, the team initially began designing a buck converter that could power the Raspberry Pis and Arduinos. Ultimately, however, the team prioritized the development of the neural network and purchased two voltage regulators to power the Raspberry Pis. The remaining components were powered directly by the battery.

## 4.6 Implementation Problems

With the test car appropriately modified, system control and neural network testing began. While Chapter 8 provides a more detailed explanation of the training and testing process, the general neural network procedure was as follows. First, the test car was driven using the remote control, on a given track, for about 10 minutes. Although this quantity of training data was insufficient for producing effective models that could drive around the track the car was trained on, the minimal training data satisfied the POC. With data from the camera and the controller input, the data was next synchronized and used for the creation of a neural network model. Having produced the model, the model was then loaded onto the Raspberry Pi and the car electronics were reconfigured. Specifically, the operation of the receiver was replaced by the Arduino. Ultimately, the neural network was then tested.

Unfortunately, shortly after preliminary testing began, problems with the test car began to develop. First, the pre-installed servo motor in control of the steering assembly failed. Once disassembled, it was determined that the gears in the servo had been stripped, as shown in Figure 4-17a. The team assumed that the gears stripped during a crash. Once the broken servo motor was replaced, preliminary testing resumed. Once again, however, the new servo motor failed. To prevent the breaking of future servos, one of the steering assembly links was modified. While the original link was made of aluminum, the new part was 3D printed with a thin neck, as shown in Figure 4-17b. Designed to have minimal bending during normal use and break in the event of a crash, the new steering component acted as a failsafe to protect the servo motor. Having fixed the immediate design issue with the car, the car became mechanically capable to fully resume POC testing.





Figure 4-17: Steering assembly failures of test car, servo with stripped gears (a) and replacement steering link (b).

Although the car seemed mechanically capable, the team encountered a second problem. While performing training and testing, the car would stop after approximately 30 minutes of operation. After investigating the problem, the team found that the battery could not retain a full charge. When depleted, the battery was unable to supply the current draw required by the ESC. As a result, the battery's output voltage would drop, the Raspberry Pi would power off, and the ESC would change to safety mode. As the preliminary neural network testing was only done as a POC, the team did not attempt to fix or replace the battery. This limitation, however, was both an inconvenience and an obstacle for preliminary neural network design.

#### 4.7 Proof of Concept Results

The fundamental purpose of the test car was to establish a Proof of Concept. Specifically referenced in Table 4-1, the primary POC questions sought to establish the feasibility of the design considerations chosen for the future vehicle. With respect to question 1, the hardware selection, preliminary results established the efficacy of using a camera, Arduino, and Raspberry Pi. All systems operated and, as seen in Figure 4-7, the Arduino proved its capability of interacting with the receiver. The successful selection of hardware ensuingly solved the uncertainties affiliated with question 2. Used as training data, both the camera frames and controller inputs were captured from the selected hardware. Consequently, this data solved question 3 and allowed for the creation of preliminary neural network models.

Unfortunately, questions 4 and 5 of the POC were unanswered from the experimentation on the test car. While neural networks were capable of being loaded onto the Raspberry Pi, the Arduino was unable to read the Raspberry Pi's instructions. Consequently, the neural network exhibited no control of the vehicle. Although the interface of Raspberry Pi and Arduino was not perfected with the test car, however, it was quickly fixed once initial testing was conducted on the real car.

Ultimately, while the test car was not optimal for autonomous driving, it played a pivotal role in the development of the project. Providing a foundation on which preliminary work

regarding controls and neural networks could be tested, the test car served as a proof of concept for future work. With the proof of concept, the next chapter highlights the process of creating a new custom car.

## 5.0 Custom Car

Although there are plenty of RC cars available on the market that could have been purchased and customized for the project, it was most appropriate to design and manufacture a custom car. Not only was the design and manufacturing an important academic exercise, the design helped determine the feasibility of creating a scale car with features not currently required in the ME4320 course. Furthermore, the custom design helped create a car that was specifically designed for autonomous testing.

This chapter discusses the custom car built by the team to create and test an autonomous car and modular sensor package. Due to the interdisciplinary nature of the project, poor selection of initial components, and manufacturing difficulties, the design of the custom car was very iterative. As a result, this chapter was divided into two sections. The first section discusses the early stages of the custom car including the initial components, system designs, and electronics. Additionally, the section discusses the design problems of the initial car. The second section discusses changes that were made throughout the project to improve the car and the analysis done on the final design of the car.

### 5.1 Early Stages

While the car underwent consistent redesign, modification, and maintenance, Figure 5-1 illustrates one of the initial car design iterations. Equipped with all necessary hardware, as well as, front and rear suspension systems, the car contained all necessary equipment for autonomous testing. This section discusses the systems and components in the initial design.

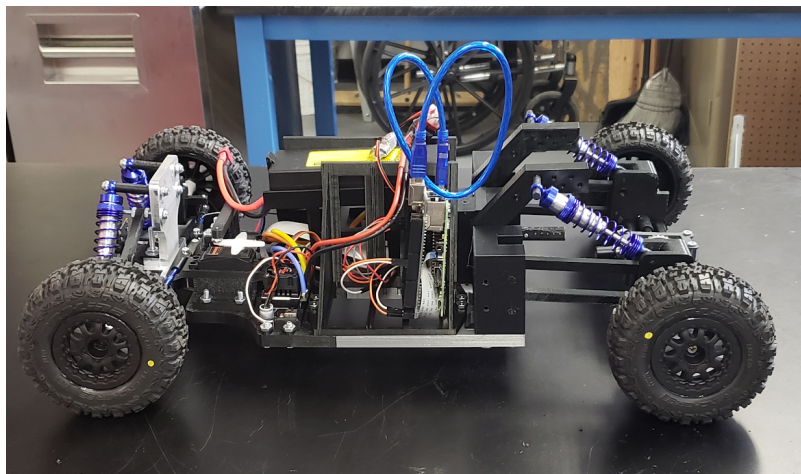


Figure 5-1: Initial Car Design

One of the difficulties of designing the custom car was fitting all the mechanical systems and electrical components into a compact design. The initial layout of the car will be described with the aid of Figure 5-2 below. The front and rear suspensions were limited to the front and back ends of the chassis, labeled as A and H respectively. The steering assembly was placed directly behind the front suspension, labeled as B. The electrical components are all placed in the middle of the chassis. The receiver and ESC are behind the steering assembly, labeled E.

The Raspberry Pi, Arduino, and circuit boards are on the sides of the car encompassing the battery and gearbox. The circuit boxes holding the Raspberry Pis, Arduinos, and circuit boards are labelled C and F. The battery and gearbox are in box D, with the battery above the gearbox. Lastly, the rear differential and driveshaft are in the center rear of the vehicle, located in boxes G and I, respectively.

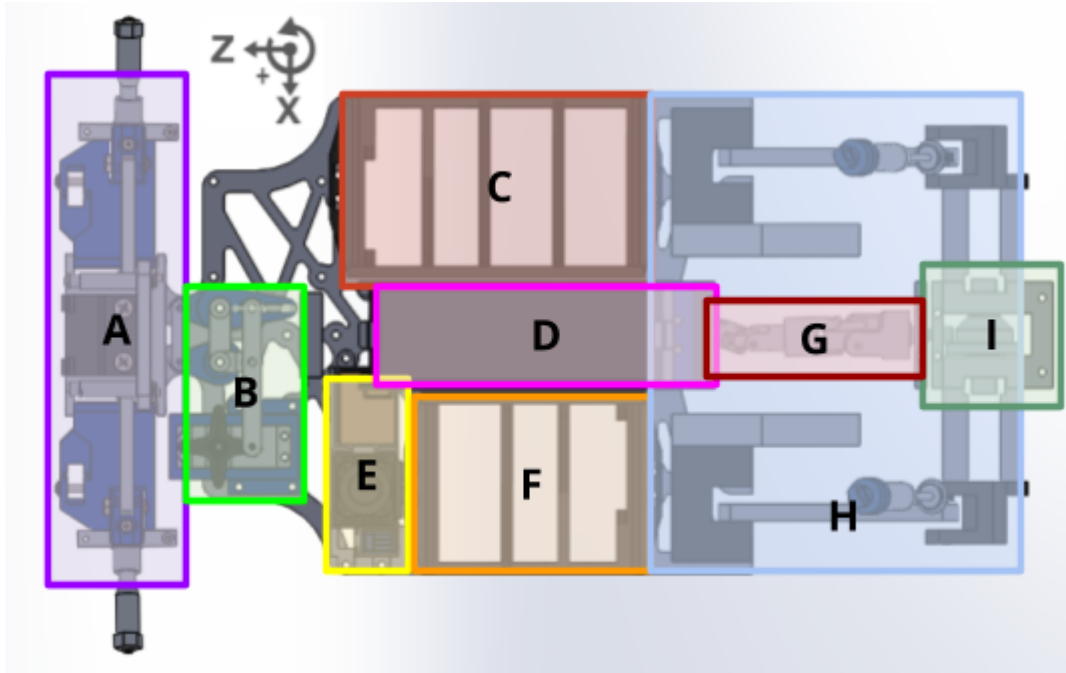


Figure 5-2: Initial layout of custom car

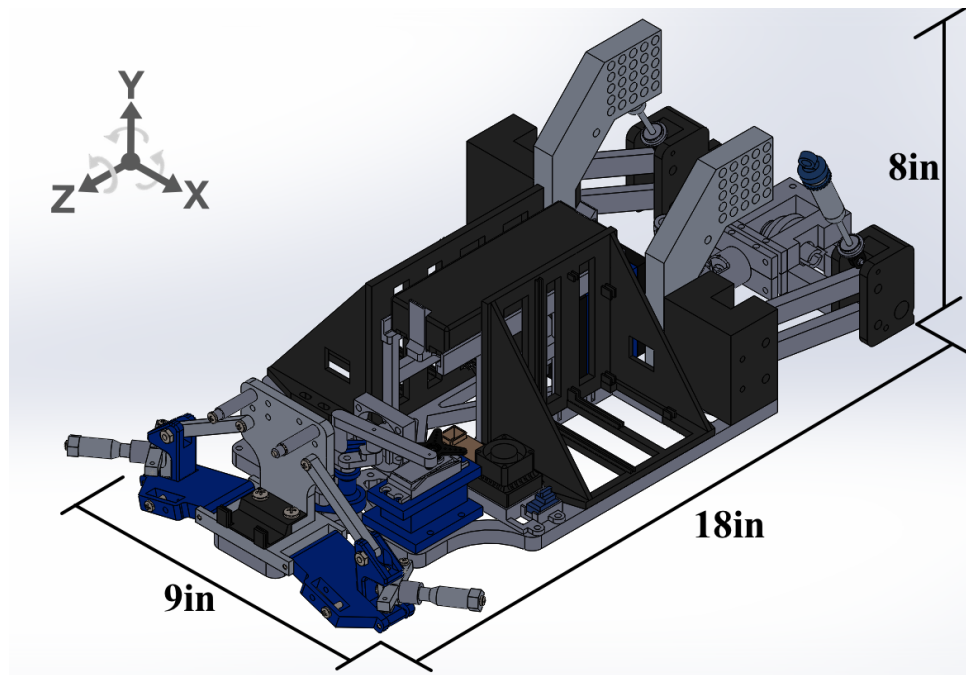


Figure 5-3: Initial custom car dimensions

The initial car design was approximately 9 inches wide and 18 inches long. Furthermore, the car weighed approximately 6.3 pounds and was approximately 8 inches tall without the camera tower (Figure 5-3). As can be seen in Figure 5-1, the majority of the car was 3D printed with a few selected parts machined. While certain components such as the shock absorbers, wheels, and camber links were purchased, the majority of the car was custom designed. For a complete Bill of Materials (BoM), refer to Appendix F.

Proceeding sections discuss the specific systems in the custom car. Figure 5-4 should be referred to while reading these sections to help understand how the various subsystems are connected.

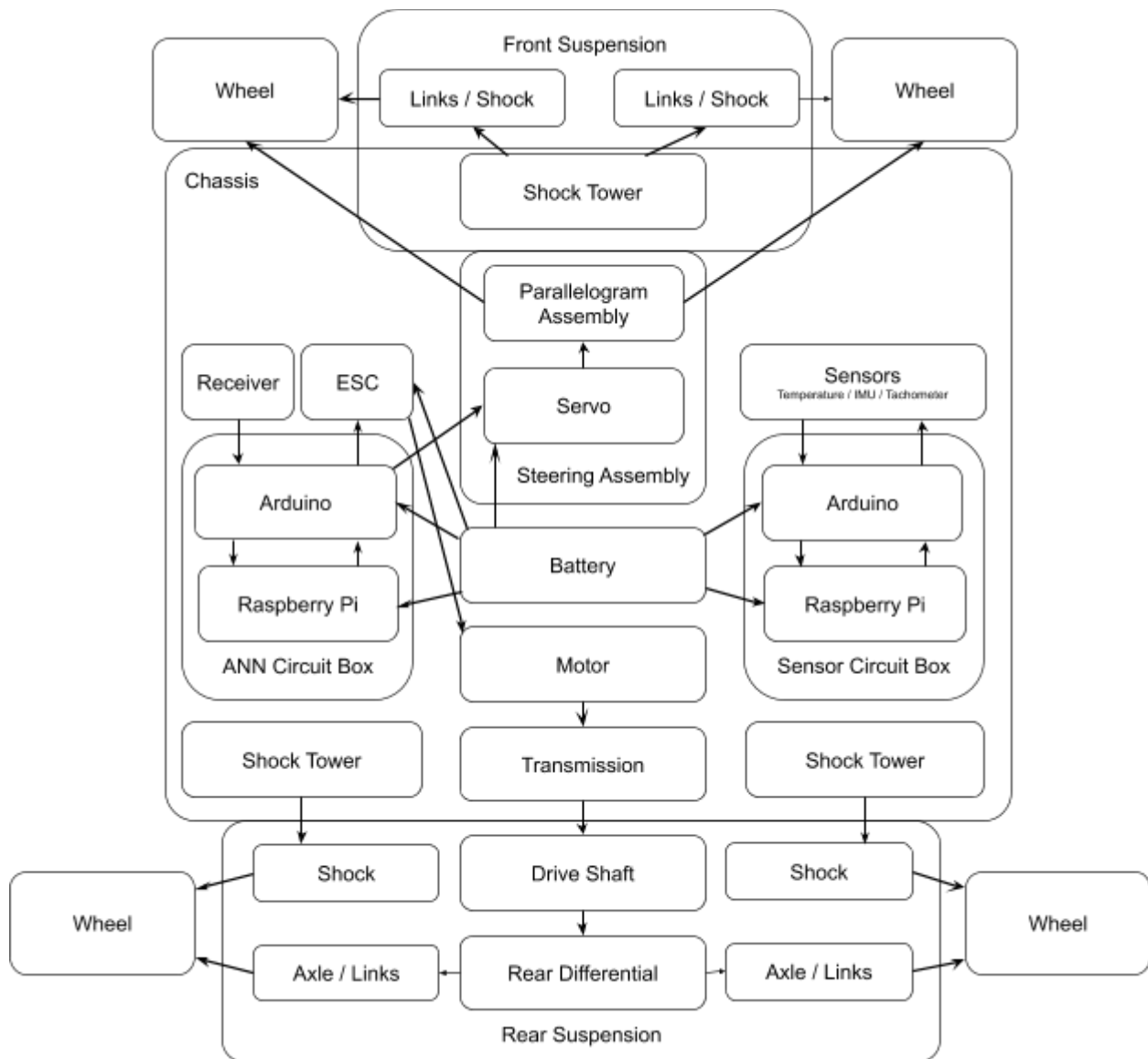


Figure 5-4: Vehicle Subsystem connections

### 5.1.1 Steering Assembly

The steering assembly designed by the team and located in box B of Figure 5-2 was based on a parallelogram steering system. The steering system connected the servo to the left of the parallelogram via a horizontal link. When the servo turned, it subsequently pulled or pushed the bottom link of the parallelogram. This, in turn, changed the angle of the left and right links in the parallelogram by pivoting it around an anchoring point on the chassis. When the right link angle changed, it caused the top link to push or pull the links connecting the parallelogram to the wheel mounts as represented by links G and I. Ultimately, this produced a steering system with a turning radius of approximately 2 feet and a maximum turn angle of approximately 45 degrees. For further illustration, a top view of the steering assembly can be seen below in Figures 5-5 and 5-6.

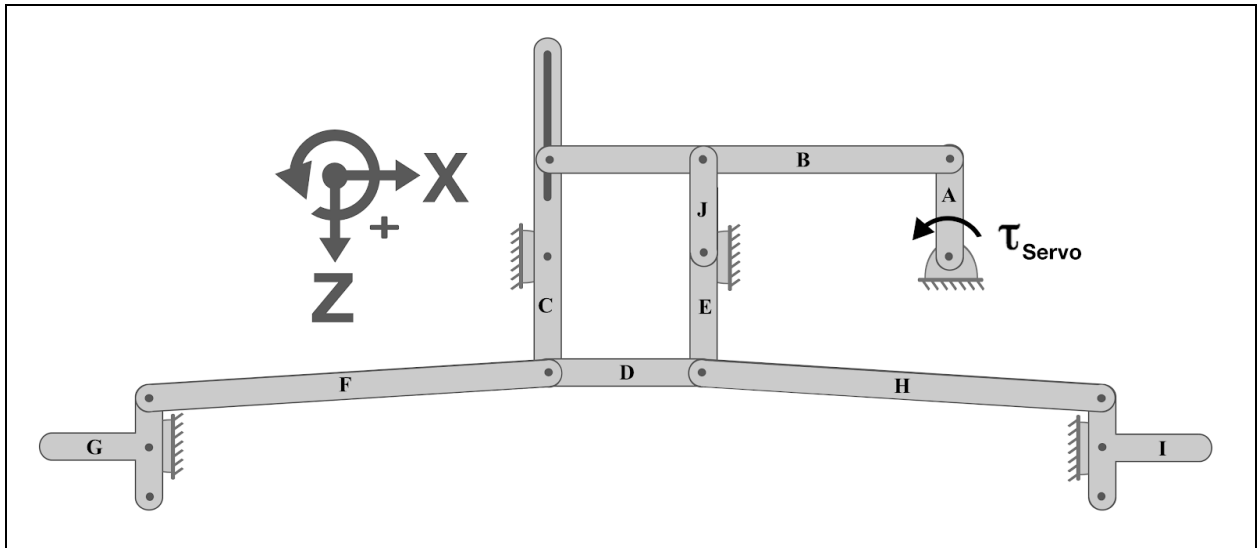


Figure 5-5: Kinematic diagram of steering assembly

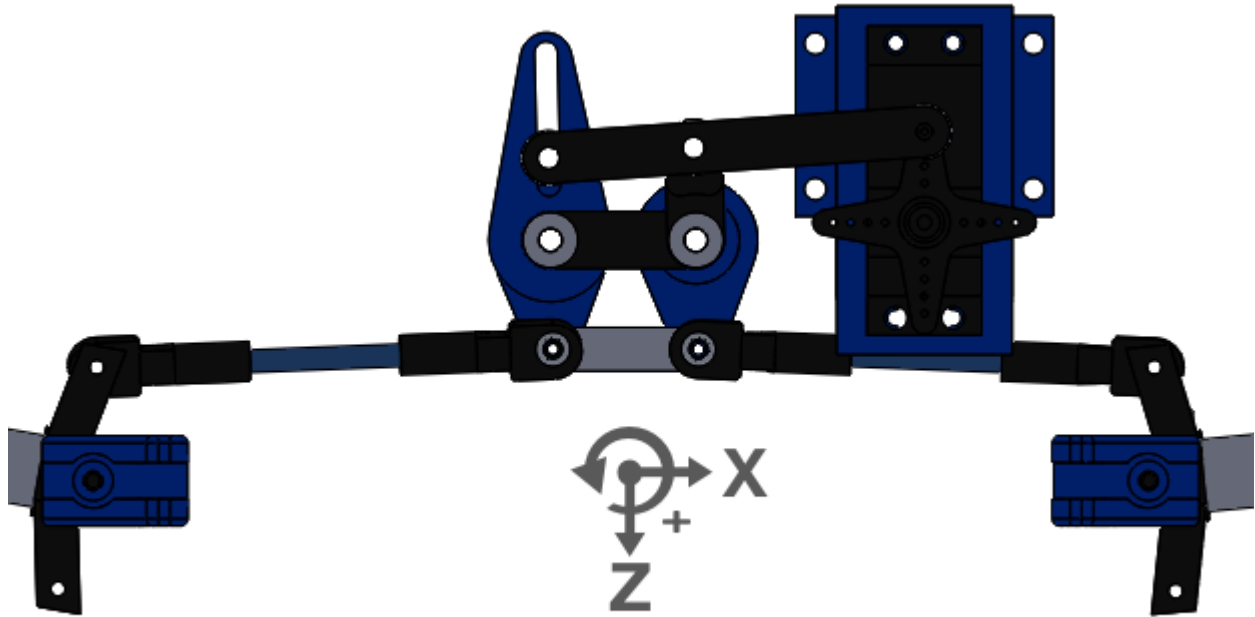


Figure 5-6: Steering System



Figure 5-7: Selected Servo

To select the initial servo, the team estimated that the maximum force necessary to turn the wheels would be about 5kg-cm. This was estimated by using Figure 5-5 to calculate the torque at the servo caused by a force acting at the wheel. The force at the wheel was assumed to be the full weight of the car in order to add a considerable safety factor for unconsidered forces (i.e. friction). Due to this calculation, the S6020 High Torque Mid Speed Digital Plastic Servo, seen in Figure 5-7, was chosen. This plastic servo has a torque of between 7.2-10.5 kg-cm. The servo also has quick response speed between 0.19-0.23 seconds per 60 degrees of rotation. Other servo specifications are listed in Table 5-1. In addition to exceeding the

estimating torque requirements, the servo was recommended by the owner of the Turn4Hobbytown where other components were purchased. A basic model of the servo was made in SolidWorks and can be seen in Figure 5-8.

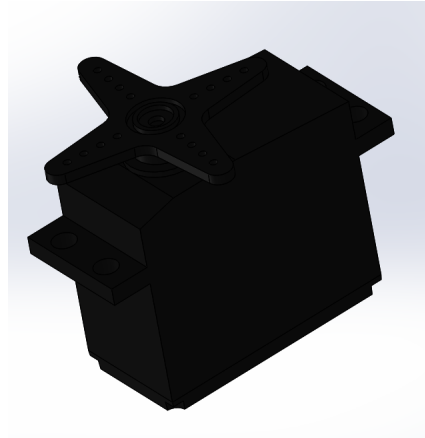


Figure 5-8: SolidWorks Drawing of Servo

Table 5-1: Servo Motor Specifications

Torque	7.2kg-cm@4.8V, 10.5kg-cm@6V
Speed	0.23sec/60deg@4.8V, 0.19sec/60deg@6V
Dimensions	1.6"(L)x0.8"(W)x1.5"(H)
Weight	1.7oz
Gear Material	Metal with 1 Plastic gear

### 5.1.2 Front Suspension

Designed around the existing steering assembly, the front suspension of the car was designed from a modified version of a double wishbone suspension. A common design in commercially available RC cars, the selected wishbone suspension is a modified four bar linkage.

The front suspension, as illustrated in Figure 5-9, was comprised of several components including the top and bottom links, shock tower, shocks, steering knuckle, and lower plate. Acting as the mechanical foundation, the shock tower was rigidly attached to the base of the car through the lower plate. The shocks then attached from the shock tower to the lower links. One side of the lower link was attached to the lower plate, which held the pin on which the lower links pivoted, while the other side was attached to the steering knuckle. The middle of the steering knuckle supported the stub axle that connected to the wheel, while the top half was connected to the top link. The top link was then connected back to the shock tower.



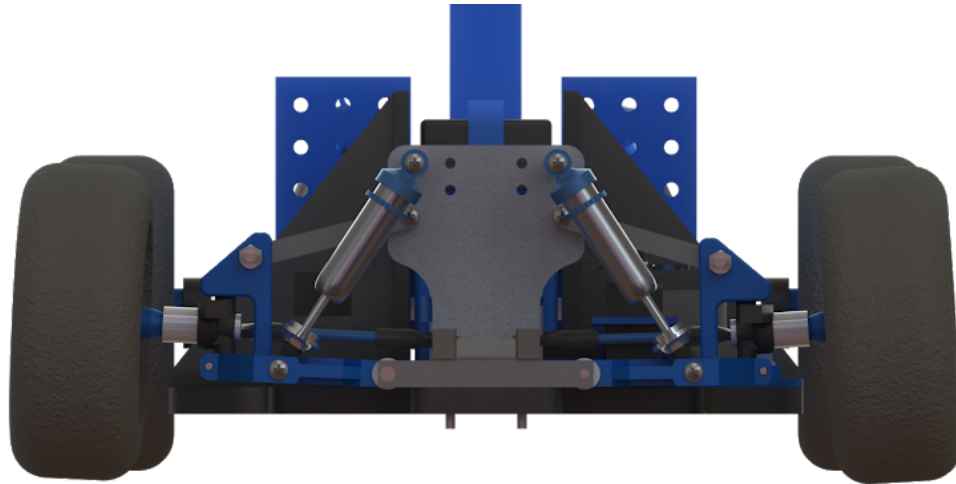


Figure 5-9: Front Suspension

The selected shocks had an overall extended length of 110mm and a compressed length of 70mm. This meant that the shocks could experience a maximum undulation inducing 40mm of length change. There were several problems with the front suspension, that prevented the suspensions from articulating fully. First, the shocks were mounted at an angle so the actual maximum lateral translation was closer to 35mm instead of the maximum 40mm. Second, the shock absorbers used passive, air-controlled dampeners. Consequently, the dampeners commonly had air pockets which affected the responsivity and travel of the entire shock absorber. Ultimately, while not tested due to time constraints, the onboard IMU could have been used to approximate the actual undulations experienced by the front suspension.

### 5.1.3 Drivetrain

#### Motor

With the front suspension and steering assemblies created, the next subassembly of the vehicle was the drivetrain. Consisting of a DC brushless electric motor, electronic speed controller, gearbox, driveshaft, and differential, the drivetrain produced and translated power to the wheels. As the vehicle was rear-wheel drive, the design of the drivetrain was also integrated into the design of the rear suspension.

During the motor selection, the team first determined what the required input torque from the motor would be. This value was estimated by determining the stall torque required to lift 75 percent of the car's expected weight from one of the rear wheels. The estimated weight of the car was 3kg and the radius of the wheels was 6.35cm. Based on these values, the team determined that the required torque of the motor at the rear axle would need to be 1.4Nm. Due to this design constraint, the team selected the Justock-3650 (Figure 5-10). The Justock-3650 is a 1800KV DC brushless electric motor that produces 110W of power. The motor has a stall current of 103.5A and a torque constant of  $5.56e-4$ , resulting in a stall torque of 0.0575Nm. After being attached to the drivetrain with a 1:27 step down, the output torque at the wheels was 1.55Nm. Additional specifications about the motor can be found below in Table 5-2.



Figure 5-10: Selected Motor

Table 5-2: Motor Specifications

Model	Justock-3650-G2-21.5T
KV (No-load)	1800KV
Resistance	0.0715Ω
Current (No-load)	1.3A
Current at M.O.P	33A
Max. Output Power	110W
Dimensions	Diameter=3.17mm Length=52.5mm
Weight	182g
Input Voltage	7.4V

Having selected the Justock-3650, it was next modeled in SolidWorks in order to assist the design of the gearbox. Specifically, the dimensions of the motor were essential in designing adequate clearance and alignment. The SolidWorks model of the DC brushless electric motor can be seen in Figure 5-11.

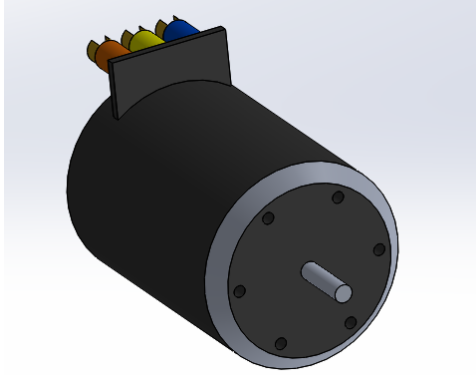


Figure 5-11: SolidWorks Drawing of Motor

### **Electronic Speed Controller**

To complement the selected DC motor, the next step was the selection of an electronic speed controller (ESC). As seen in Figure 5-12, the XR10 Justock ESC was selected. Compatible with the chosen motor, the ESC satisfied the one fundamental requirement for its operation. As illustrated in Table 5-3, the XR10 also had many other specifications. Similar to the motor, the ESC was modeled in SolidWorks as seen in Figure 5-13.

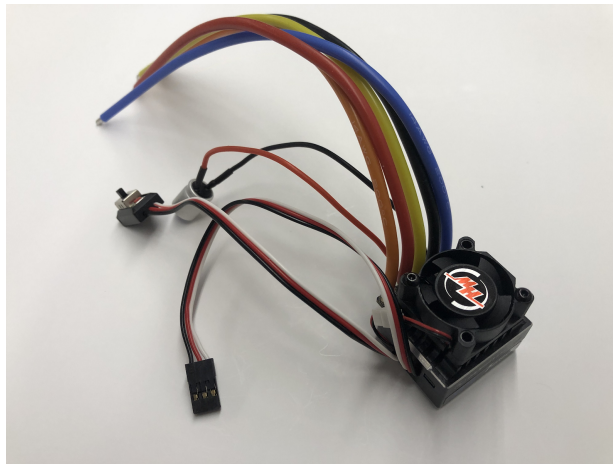


Figure 5-12: Selected ESC

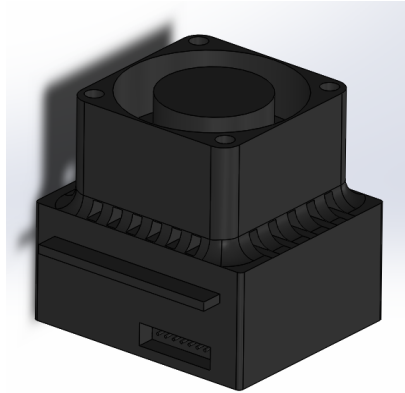


Figure 5-13: SolidWorks Drawing of ESC

Table 5-3: ESC Specifications

Model	XERUN XR10 Justock
Output (BEC)	6V, 2A
Continuous Current	60A
Peak Current	380A
Dimensions	33.5mm(L) x 28.5mm(W) x 30.5mm(H)
Weight	59.8g

### Gearbox

With the necessary electronics to produce torque, the next step in the drivetrain was the gearbox. A means of reducing the high RPM of the electric motor to generate greater torque, the gearbox design can be seen in Figures 5-14 and 5-15.

In accordance with the requirements for ME4320, a principle design goal for the car was to have an overall speed ratio of 1:27 from the motor to the wheels. The gearbox had a step down of 1:9, and the differential that was selected had an additional 1:3 reduction. Together, the entire gearing system produced create an overall 1:27 reduction. Within the gearbox, the 1:9 reduction was created using a 12 tooth, 48 pitch pinion gear, and a 36 tooth, 48 pitch spur gear. To ensure their compatibility with the selected motor, the smaller pinion gears were purchased from a local hobby store, Turn4Hobbytown. Meanwhile, the larger spur gears were purchased on McMaster Carr. Table 5-4 gives more detail about the components of the gearbox.

Table 5-4: Gear Specifications

<b>Stage One Gears</b>	<b># of Teeth</b>	<b>Pitch (teeth/in)</b>	<b>Pitch Diameter (in)</b>	<b>Shaft Diameter (in)</b>	<b>Mounted on</b>	<b>Material</b>
Input Pinion Gear	12	48	0.25	1/8	Motor Shaft	Steel
Output Spur Gear	36	48	0.75	1/4	Compound Axle	Steel
<b>Stage Two Gears</b>	<b># of Teeth</b>	<b>Pitch</b>	<b>Pitch Diameter</b>	<b>Shaft Size</b>	<b>Mounted On</b>	<b>Material</b>
Input Pinion Gear	13	48	0.25	1/8	Compound Axle	Steel
Output Spur Gear	36	48	0.75	1/4	Output Shaft	Steel

By stacking the two stages, the input and output shafts of the gearbox could be positioned so that they were in line with each other. This was done in order to reduce the overall size of the gearbox, thus making it possible to mount the gearbox at the center of the car. The overall size of the gearbox is 2.375" x 2" x 2.5".

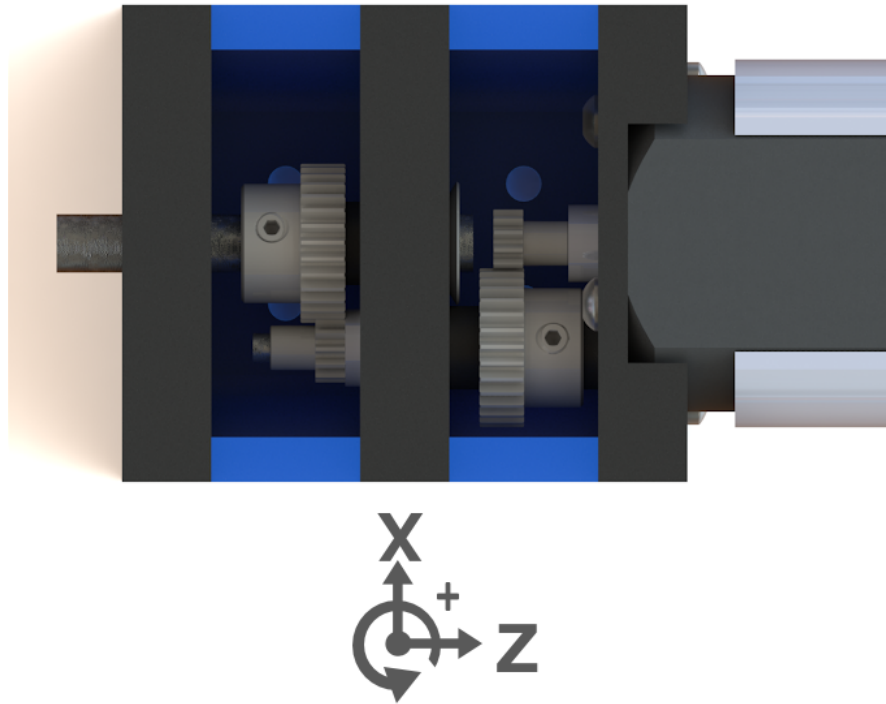


Figure 5-14: Transmission Design

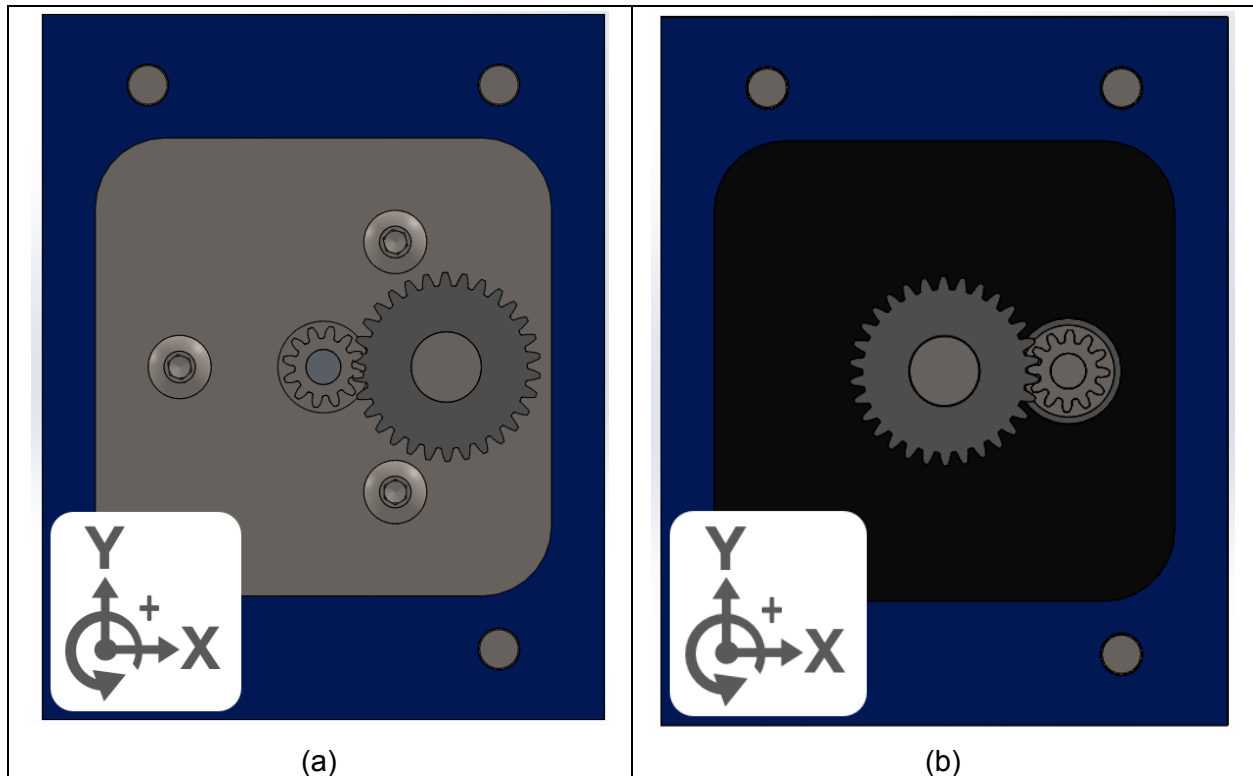


Figure 5-15: Gear Mesh Within Gearbox. (a) Stage 1, (b) Stage 2

In order to assemble the gearbox, the first step was to attach the middle spacers shown in blue in Figure 5-14 to the base plate of the car. Next, the motor mounting plate was screwed to the motor, and the pinion gear was attached to the motor shaft. All gears are attached to their respective shafts via set screw and any shafts that were machined had a facet to screw on. Following, the first stage spur gear was mounted on the compound axle and the shaft was slid into the middle mounting plate. Once the middle mounting plate was in place, the second stage pinion gear was attached. The second stage spur gear was then mounted to the output shaft and the output shaft was fed through the outer plate. Screws were then threaded through all of the plates and the gearbox was closed.

During gearbox assembly, several inspections were performed including: testing the fit of all shafts to ensure that they spun freely, ensuring that the gears were aligned and lubricated, ensuring that the shafts were at the right height, ensuring that the gears were tightly mounted on the shafts, and that the output shaft and motor shaft have enough clearance. The distance between the output shaft and motor input shaft was 0.085". Each of these inspections was critical to the consistent operation of our gearbox.

### Driveshaft

From the gearbox, the next step in the drivetrain was the transfer of rotational power from the output shaft to the rear differential. For a car with either an independent rear

suspension or no rear suspension, the rear differential could have been connected directly to the output of the gearbox using a rigid cylindrical rod. As this car iteration utilized a dependent rear suspension, however, the design of the driveshaft was more complicated.

As the rear suspension allowed the entirety of the rear axle to articulate, the output shaft had to be able to transfer power to the rear differential at variable angles. Given this design constraint, the output shaft utilized a universal joint purchased from Turn4Hobbytown. This piece was then given appropriate adaptors and was modified to 4.5 inches to fit the dimensions of the car. An illustration of the customized output shaft can be seen in Figure 5-16.

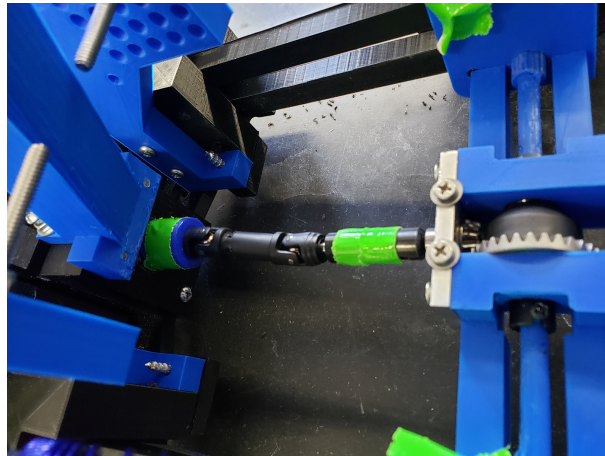


Figure 5-16: Driveshaft Design

## Differential

Connected to the universal joint driveshaft, the final step of the drivetrain was the differential and rear axle. As illustrated in Figures 5-17 and 5-18, the car was equipped with an open differential with a 1:3 step down ratio. Furthermore, the ring gear was 2 inches long with While full-size vehicles normally use limited slip differentials or other more complicated mechanisms, the car did not require nor need a sophisticated differential. Instead, the car needed a simple open differential which could rotate the torque of the output shaft to the rear axle while simultaneously allowing each wheel to spin independently.

In order to support the two gears of the differential, a housing was also constructed. Using the low-friction nature of PLA as bearings, the housing consisted of a top and bottom piece which encompassed the pinion and ring gears of the differential. Ultimately, this housing helped ensure the stability of the differential gears while simultaneously articulating with the rear axles and wheels. Furthermore, as the differential housing was 5.45 inches long, its legs helped contact the axle mounts on the rear axle. This allowed the differential housing to also act as a Panhard rod, reducing the lateral strain experienced at the rear.

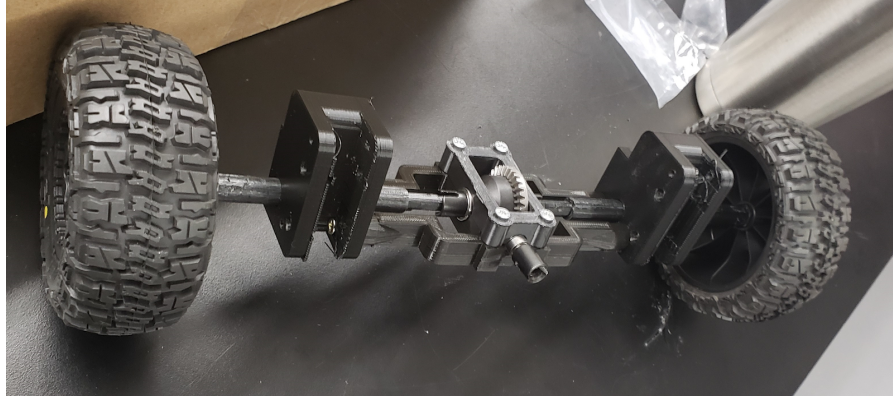


Figure 5-17: Differential and Differential Housing



Figure 5-18: Selected Differential. Reproduced from [67].

Table 5-5: Differential Specifications

Model	JLB Racing EA1057
Step Down Ratio	1:3
Material	Iron
Dimensions	43mm(L) x 50mm(W) x 26mm(H)

With the differential integrated, the drivetrain was fully assembled. As seen in Figure 5-19, the drivetrain looked as follows. Particularly of note was the length of the driveshaft which covered the gap between the gearbox and the input pinion to the rear differential. As the gap had to accommodate the free rotation of the driveshaft and universal joint, the distance was 4.05 inches.



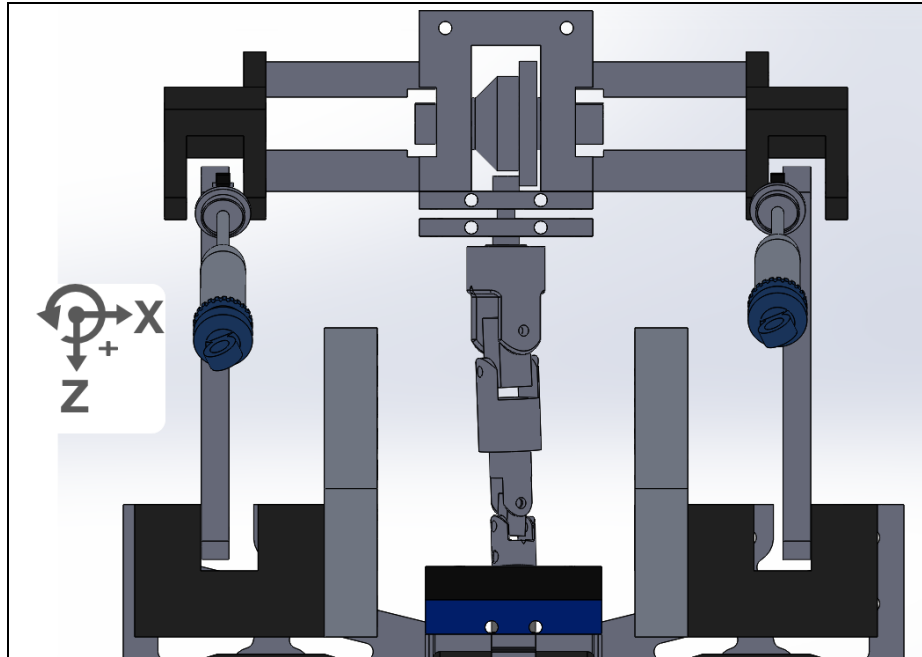


Figure 5-19: SolidWorks Assembly of Powertrain

#### 5.1.4 Rear Suspension

The final design step was the implementation of the shock absorbers and linkages to create a rear suspension. In order to connect the rear axle to the base plate of the car, a four-bar linkage was used. Comprised of two vertically stacked rods on each side of the differential, these rods physically connected the rear axle to the car chassis while still allowing vertical motion. Specifically, the design of the four-bar mechanism can be seen in Figure 5-21. As can be seen in the illustration, the rear axle mounts had bearings to ensure that the rear axle could still rotate while supporting the weight of the linkages.

Although the linkages ensured that the rear axle remained attached to the chassis, the linkages were unable to keep the rear of the chassis elevated. Consequently, in order to control the vertical elevation and displacement of the chassis, two shock absorbers were implemented. Using a shock tower fastened to the base plate as the foundation, the other end of the shock absorbers was attached to the rear axle mounts used for the linkages. As for the four-bar linkage, the implementation of the shock absorbers can be seen in Figure 5-21.

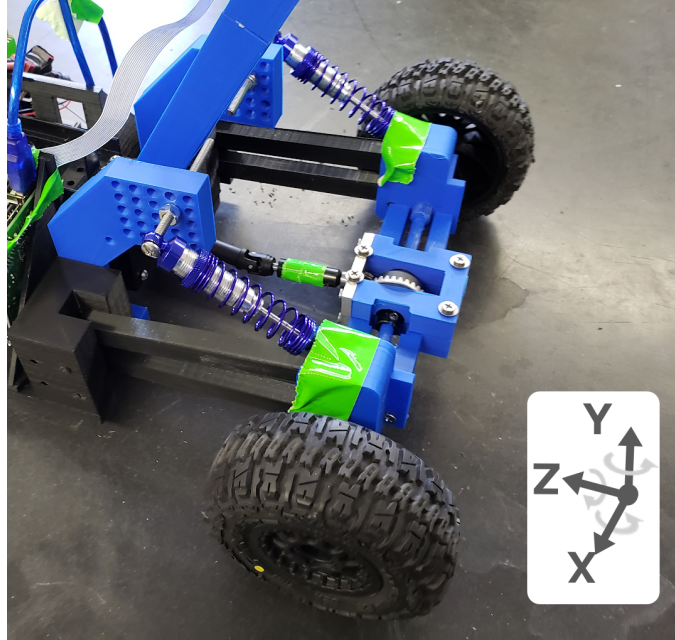


Figure 5-20: Dependent Rear Suspension

Using the same shock absorbers as the front suspension, the maximum theoretical travel of the rear suspension was 40mm. Due to the extreme shock absorber angle (45 degrees) and the presence of air gaps within the dampeners, however, the rear suspension was discovered to achieve a maximum displacement of 20mm when externally forced.

### 5.1.5 Other Components

#### Battery



Figure 5-21: Selected Battery

Seen in Figure 5-21 is the battery selected for the RC car. This battery is a 2-cell lithium polymer battery in a hard case. The battery capacity of 5200mAh was chosen specifically so that the team could run the car for long periods of time without needing to recharge or replace the battery. Furthermore, the 50C discharge rate allowed the group to power all of the onboard

electronics as well as the servo and motor. Using an estimated power usage of 1.7W, 1.0W, 20W, and 2.3W for the Raspberry Pi, Arduino, motor, and servo, the estimated power draw of the components was 27W. This resulted in an estimated battery life of about 1.4hr. The battery was then modeled into SolidWorks as seen in Figure 5-22. Additionally, Table 5-6 illustrates the battery specifications.

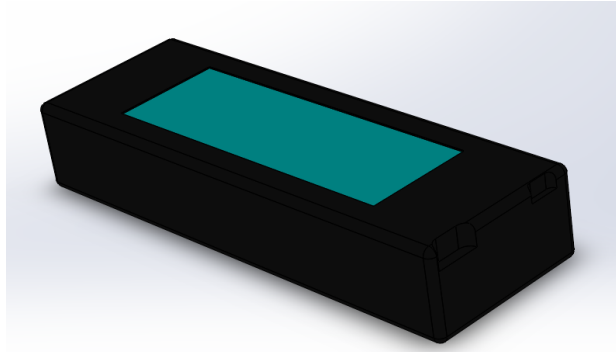


Figure 5-22: SolidWorks Drawing of Battery

Table 5-6: Battery Specifications

Voltage	7.4V
Energy Storage Capacity	5200mAh
Discharge Rate	50C
Charge Rate	2C
Dimensions	138mm(L) x 47mm(W) x 25mm(H)
Weight	266g

### Camera

The specifications required by the team were that it needed to have a wide field of view and be compatible with the Raspberry Pi 3 B+. The camera needed to have a wide angle view to capture the amount of data necessary to run the ANN. The camera had to also be compatible with the Raspberry Pi 3 B+ because it needed to be able to interface with the hardware on the custom car. To meet these specifications, the team chose the Amadget Wide Angle Fish-eye Camera Module, model OV564, shown in Figure 5-XX. This camera has a 175 degree field of view and is compatible with the Raspberry Pi 3 B+, making it perfectly suited the team's needs. Additional Amadget Wide Angle Fish-eye Camera Module, model OV564, specifications are tabulated in Table 5-7.

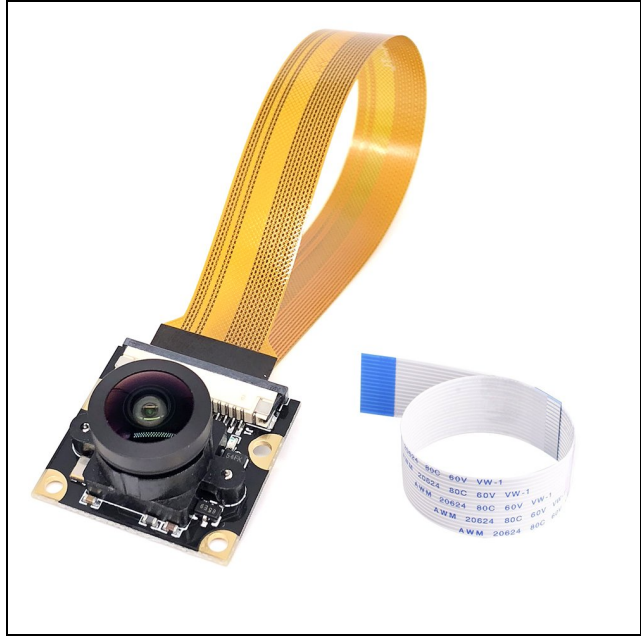


Figure 5-23: Selected camera [67]

Table 5-7: Camera Specifications

Model	OV5647
Compatibility	Raspberry Pi Models A/B/B+, Pi2 and Pi3
Lens Type	Fish-eye
Resolution	2952 x 1944
Viewing Angle	175°
Focal Length	Adjustable
Dimensions	25mm(L) x 24mm(W) x 9mm(H)
Weight	32g

## Chassis

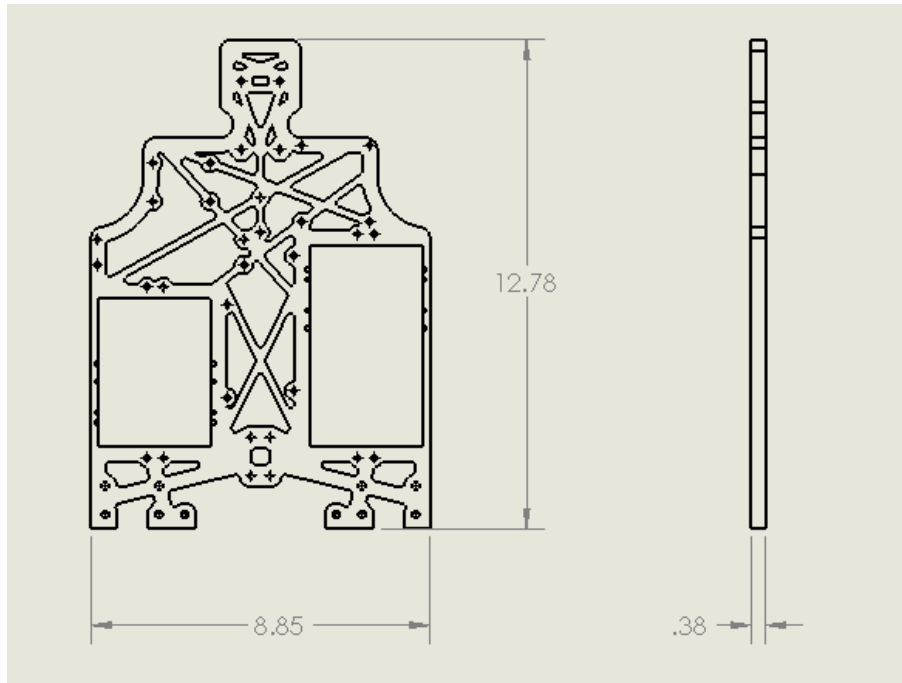


Figure 5-24: Chassis

The chassis shown in Figure 5-24 served two purposes. First, it connected the front and rear of the vehicle using a rigid frame. Second, it held all of the components needed to operate the car, including the servo, motor, Arduino, Raspberry Pi, and all remaining electronics. During its design, the mounting locations of each of the components and the turning radius of the wheels were considered. Holes for screws were designed for each of the components and component mounts, and an appropriate amount of space was given for the front wheels to turn. Ultimately, the base was 12.78inches-by-8.85inches-by-0.38inches and 3D printed.

## Mounts

The parts that needed to be mounted on the custom car included two Arduinos, two Raspberry Pis, three PCBs, three temperature sensors, an IMU, a battery, an ESC, and a receiver. In order to make the PCBs, Arduinos and Raspberry Pis readily accessible, separate boxes were placed on opposite sides of the car. Shown in Figure 5-25, the boxes allowed for the vertical placement of all PCBs, Arduinos, and Raspberry Pis. As seen in Figure 5-26, slots behind the two Raspberry Pi CPUs were made for two of the temperature sensors. Furthermore, the clip shown in Figure 5-27 went around the motor to hold a third temperature sensor. The battery was placed in the center of the car, above the motor to make the battery easily accessible. The mount, Figure 5-28, was specifically designed to raise the battery above the gearbox while also containing a slot for the IMU. Lastly, the receiver and the ESC, were mounted in front of the left circuit box. Seen in Figure 5-29, the mount was designed so the components would remain motionless during car operation while remaining easily accessible for modification.

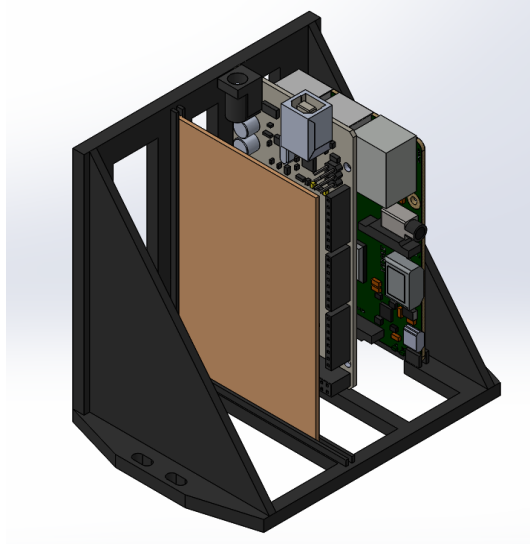


Figure 5-25: Raspberry Pi, Arduino, and PCB Mount

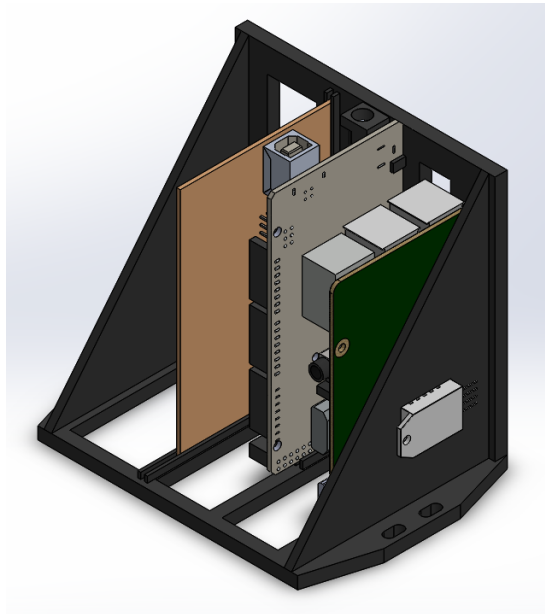


Figure 5-26: Temperature Sensor Mounts for Raspberry Pi

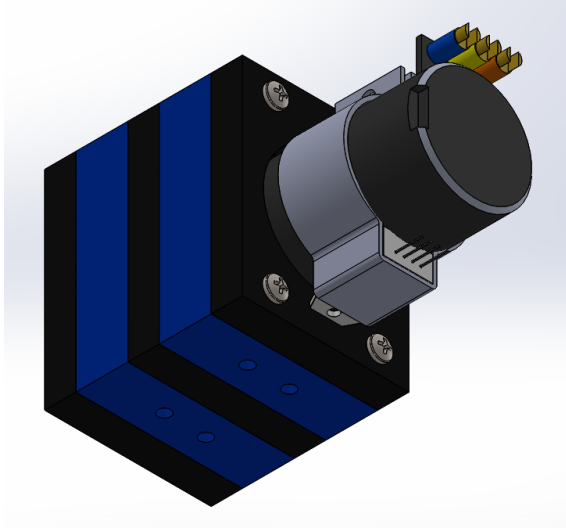


Figure 5-27: Temperature Sensor Mount for Motor

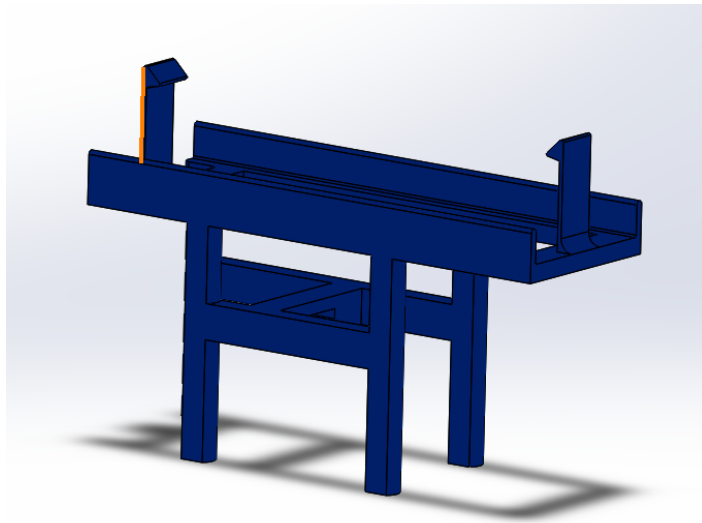


Figure 5-28: Battery Mount

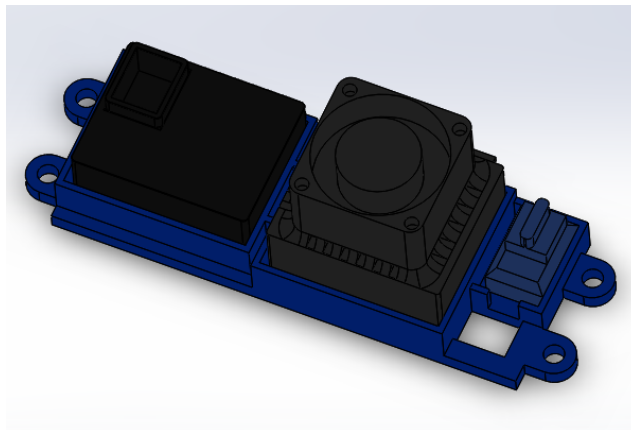


Figure 5-29: ESC and Receiver Mount

## 5.1.6 Electronics

### Power

The electrical components on the initial custom car were powered by the same methods used to power the components on the test car. The two Raspberry Pi CPUs on the car were powered by two voltage regulators that stepped the voltage down from 7.4V to 5.0V. The two Arduinos, the ESC, and the servo were directly powered by the battery. The other components on the car, for example, the receiver and camera, were powered from the Arduino or Raspberry Pi.

### Control Systems

The system controls for the servo and ESC were the same as the ones used in the test car. They were directly controlled by an Arduino, which received instructions from a receiver during training and a Raspberry Pi during neural network testing. When the Arduino was controlled by the receiver, the servo and ESC commands were measured as two PMWs. Subsequently, the commands were converted into degrees and sent to the servo and ESC using the standard servo library as seen in Figure 5-30. When the Arduino was controlled by the ANN, the commands were read, in degrees, from the Arduino's serial connection with the Raspberry Pi. The commands were then sent to the servo and ESC using the standard servo library as in Figure 5-31.

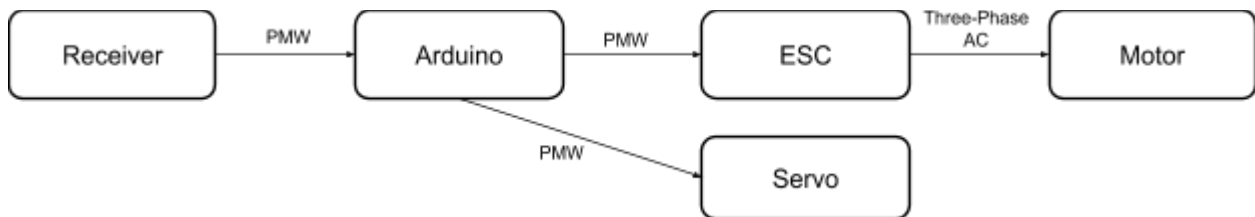


Figure 5-30: Control of Servo and Motor During ANN Training

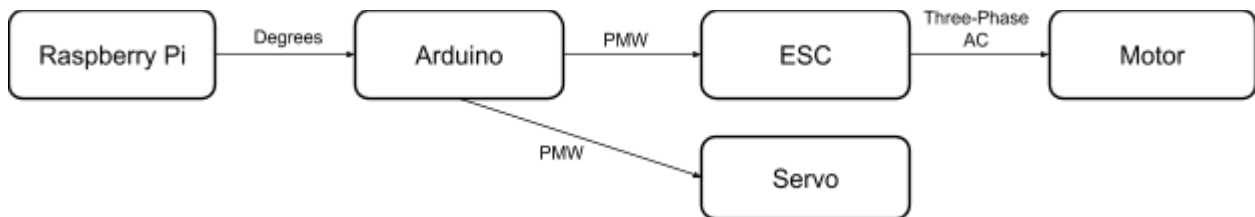


Figure 5-31: Control of Servo and Motor During ANN Testing

## 5.2 Design Issues

After developing the first stage of the car, the neural network training and testing began. Unfortunately, the frequent use of the vehicle revealed several flaws in the car's design. The systems with the most significant problems were the front suspension, the steering assembly, and the gearbox positioning.



### **5.2.1 Front Suspension**

As the front suspension moved vertically, the wheels toed either inward or outward. This phenomenon was a result of the connection between the tie rod and the wheel mount not moving synchronously with the wheel mount pivot axis as the front suspension moved vertically. This was a consequence of the suspension links not experiencing the same horizontal travel as the connection between the tie rod and the wheel mount. This inconsistency had three causes. The first inconsistency occurred because the tie rod in the steering linkage had a different length than the top and bottom links of the suspension (77mm, 58mm, and 60mm respectively). The different lengths resulted in different rotations and therefore different horizontal travels. The second inconsistency was caused by the suspension links not being the same length (58mm and 60mm). These different lengths also resulted in different rotations and therefore different horizontal travel of the links. The last reason for the inconsistency was that the tie rod and suspension links had misaligned mounting locations and were not parallel. Again, this resulted in different rotations and therefore different horizontal travel of the links.

The toeing, while a problem on its own, produced an additional problem. Due to the way the steering linkage interacted with the suspension, the compression of the shocks induced changes in both the angle of toe and the contact angle between the tires and the ground. This meant that the actual observable undulation that the front suspension could undergo was reduced from 35mm to about 15mm. This was not due to binding in the shocks, but was a constraint to ensure that the position of the tire was relatively consistent throughout the travel of the suspension system.

### **5.2.2 Steering Assembly**

The steering assembly had two problems. The first problem involved play in the steering of about a quarter of an inch. This was a significant issue because the neural network expected a deterministic environment. This meant that when a command was given, the output was exactly as commanded with no variability. When this did not happen, the ANN was put into situations that it was not trained to handle, causing the car to crash. The play resulted in a nondeterministic world because it would cause drift, i.e. a command from the ANN of straight did not result in the car going perfectly straight. The play in the steering assembly was caused by the imprecision of the 3D printed parts used in the steering system. As prints came off of the printer, they would often be either too big or too small at locations such as holes. In order to account for this, holes were undersized by approximately 2mm and then drilled out. This method resulted in holes that were more consistent and more precise than holes that were printed to size. The second problem with the steering assembly involved the servo. When the car would crash, the servo would short circuit and fail. This happened because as the wheels hit a wall, the servo would continue to send power even though the position of the servo was not changing. Eventually, this maximum current draw would cause the servo to burn out. This resulted in the team needing to replace the servo twice.

### **5.2.3 Gearbox**

The last significant design flaw was that the gearbox was mounting too far back on the chassis. With the gearbox located where it was, the team had trouble fitting a driveshaft, which required two universal joints and a dog bone, between the transmission and the rear differential. Although the team was able to work around this by moving back the rear suspension, it lengthened the car by approximately 4in, making it more difficult to drive on the small tracks for which the car was designed. Had the gearbox been mounted closer to the front of the car, there would have been more space for the drive shaft connecting the transmission to the rear differential.

### **5.2.4 Additional Issues**

Another design flaw, although not significant to a specific section, was the melting of 3D printed parts due to excessive rubbing. Most notably, this caused problems in the steering assembly and the rear differential. In the steering assembly, excessive rubbing of the rotating wheel on the wheel mount created significant play. In the rear differential, the excessive rubbing caused both the housing and plastic axles to deform, causing the gears to slip and the car to have areas of high friction.

Several of the problems listed above were related to either poor design or a poor understanding of the way components were going to be manufactured. For example, the original housing for the rear differential was made out of 3D printed PLA. As the axle spun, it rubbed on the housing creating friction and heat. As the part heated up, the plastic melted and the housing was no longer able to function properly. Had the component been designed better from the beginning so that there was less friction, or had the component been made out of a more wear and temperature resistant material, this problem could have been avoided.

Nearly all of the components on the car were manufactured on campus through a variety of methods, including CNC Mills, CNC Lathes, 3D printers, and hand tools. Each of these methods had limitations that should have been considered earlier in the design process. For example, the rear axles in the car were very long rods with stepped diameters. The parts were to be machined out of steel which would require the use of a lathe, but due to the length of the axles and the limitations of turning operations, the parts had to be redesigned. The lathe can only have so much stock material protruding from the chuck or the workpiece will experience significant amounts of deflection. To combat this deflection, the axles were redesigned so that less of the overall workpiece needed to be turned down. If we had considered the limitations of the lathe when designing this component, we could have saved time and materials.

In order to prevent issues such as these for future students who may work on this Major Qualifying Project, or students in ME 4320, we created a Design for Manufacturability Guide that can be used to create more manufacturable parts. This guide outlines the limitations of each manufacturing method as well as some general design guidelines. This ensures that parts will be manufacturable and function as intended.

### 5.3 Last Stage

After becoming more aware of the design flaws with the initial custom car, the team created several iterations of the car systems to fix or improve the problems. In addition, other features were added to increase the car's usability. This section discusses the result from the iterations including the modifications that were made and the final car design.

#### 5.3.1 Component Changes

To prevent shorting or overheating, the original servo was replaced with an all metal servo that had a higher torque specification. The specifications of the servo motor can be seen in Table 5-8 while Figure 5-32 shows an image of the new servo.

Table 5-8: New Servo Motor Specifications

Model	SC-1268SG
Input Voltage	7.4V
Speed	0.11 sec/60°
Torque	25.0 kg-cm
Dimensions	54mm(L) x 20.2mm(W) x 37.2mm(H)
Weight	62.0g



Figure 5-32: New Servo Motor

### 5.3.2 Design Alterations

To improve the play in the steering, the team made modifications to both the steering assembly, and front suspension. Furthermore, the car underwent a significant redesign of the rear suspension. To reduce the imprecision of the 3D printed parts that contributed to the play, each part was altered so the pin holes were initially printed too small (approximately 2mm each). After being printed, each hole was then stripped with the screw or pin that would hold it in place so the parts would have a tight fit. The wheel mounts in the steering assembly were also adjusted so the wheels tilted towards the car, making it so the car had a natural resting location. These modifications reduced play within the car's steering by about 95 percent. The second problem with the steering assembly, the wheels toeing when the front suspension moves vertically, was not fixed because any design alterations that would have fixed it would have required a full redesign for the front of the car. However, the team did develop a solution where the steering assembly turned the wheels while keeping the tie rods parallel with the suspension links. This was done by ensuring that the tie rod always pivots about the same point. In the solution, when the servo was turned, it extended or contracted rods with a notch on which the tie rods pivot. The extending or contracting of the rods pushed or pulled a slanted slider, consequently causing the tie rods to extend or contract. The tie rods remain parallel with the suspension links because they always rotate about the rods connected to the servo. The model of the solution that the team developed is shown in Figures 5-33 and 5-34.

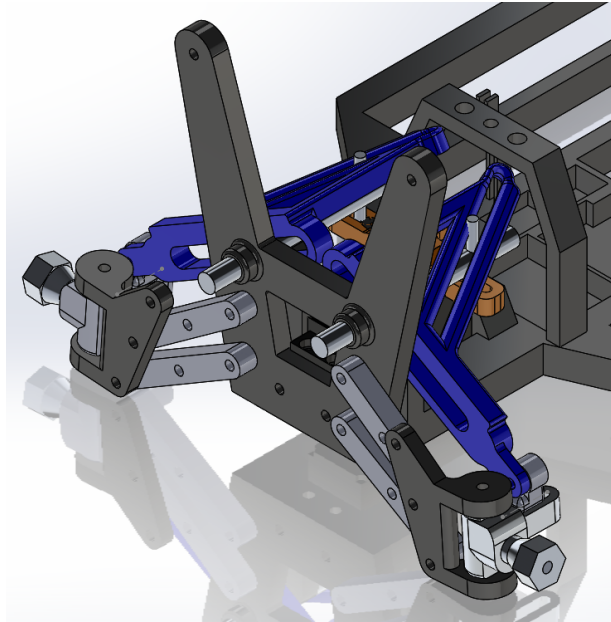


Table 5-33: Isometric view of steering assembly solution

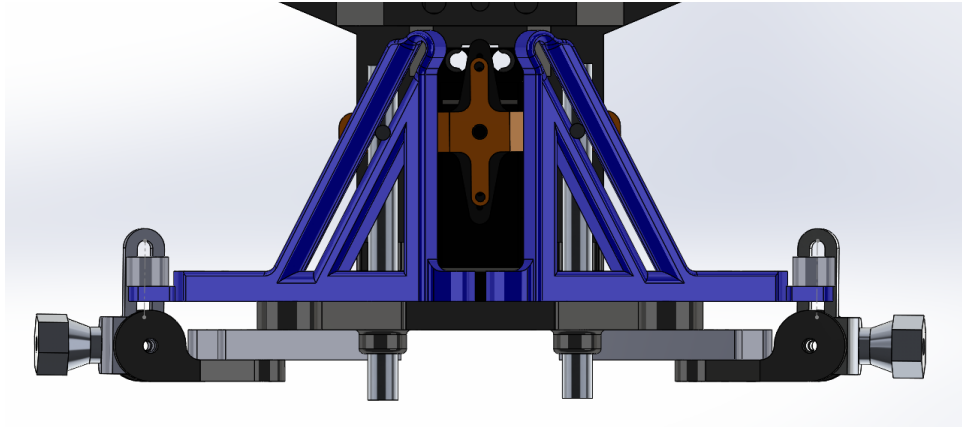


Table 5-34: Top view of steering assembly solution

In addition to replacing the servo to prevent shorting or overheating from a crash, the team also designed a bumper. The bumper can be seen in Figure 5-35 below. The bumper was attached to the front of the chassis and encompassed the majority of the front wheels. This modification prevented the wheels and the servo from jamming during a crash. See Figure 5-35 for a model of the front of the car with the bumper.

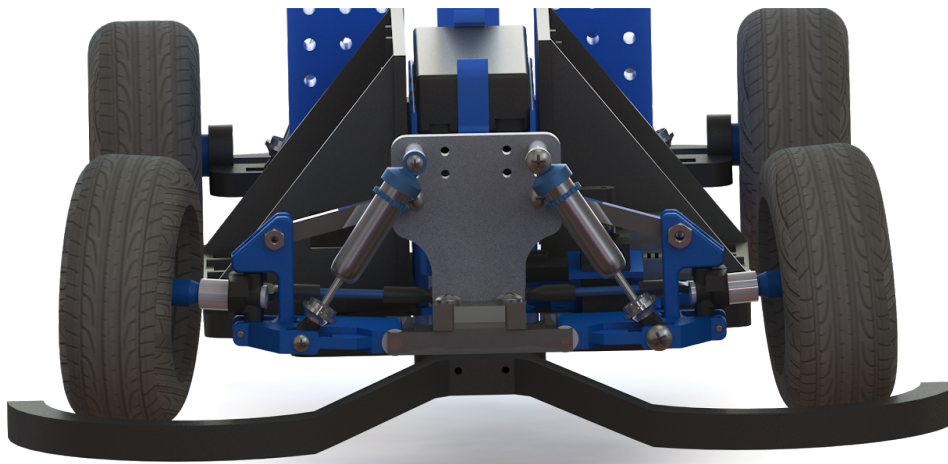


Figure 5-35: Model of the custom car with the bumper

In order to reduce the overall length of the vehicle, the rear suspension was also redesigned. Changed from dependent to independent, the new rear suspension utilized a modified wishbone structure similar to that of the front suspension. While not shown in Figure 5-36, the new rear suspension used two modified universal joints as rear axles. By making the rear suspension independent, this consequently eliminated the need for a long drive shaft. As a result, the overall length of the car reduced by approximately 3 inches.

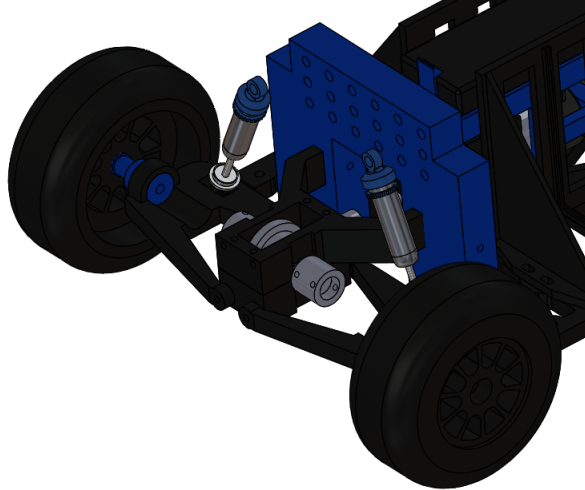


Figure 5-36: Independent Rear Suspension

Taking into account the design and component changes, the finalized car was produced as seen in Figure 5-37.

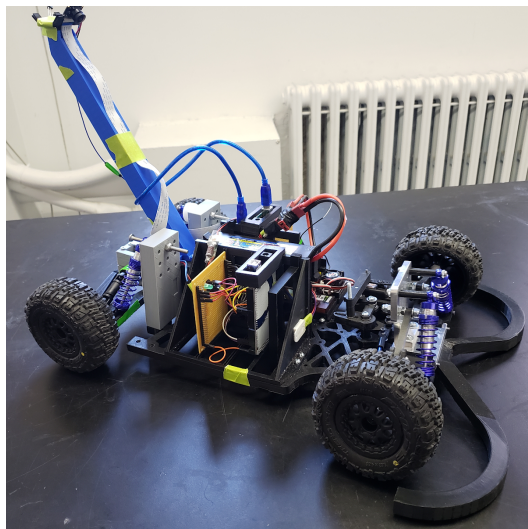


Figure 5-37: Finalized Car

### 5.3.3 Controls

Although the same method for controlling the car was used in all the iterations of custom car, several features were added to improve the ease of training and testing. The alterations included combining the programs that control the Arduino via the receiver and the ANN, adding the ability to pause and play the ANN testing, and adding the ability to collect data with steps different amounts of turning options. The team combined the programs so switching from training and ANN testing could be done immediately. The pause and play feature for the ANN testing was added so when the car started driving could be controlled more easily. The ability to collect data with different amounts of turning steps was added so that the data collected would be the same as the data outputted by the ANN, which produces an output with a predetermined

amount of turning steps. Each of these features was controlled with a remote controller with the feature that is being used being indicated by three LEDs on the car.

## **5.4 Conclusion**

In conclusion, the team successfully created a custom 1:10 scale car. Included on the car was a custom independent front suspension, independent rear suspension, rear differential, transmission, and steering assembly. Although the car had several design flaws, they were not significant, allowing the project to continue forward.

After the design of the car had been completed, the team analyzed the custom components and systems on the car. The types of analyses done included static analysis, bond graph analysis, finite element analysis, and dynamic analysis. The next chapter discusses each of these analyses in detail.

## 6.0 Mechanical Analysis

In order to monitor the loads experienced by the car, many of the car's assemblies underwent analysis. Consisting of static, dynamic, finite element, and bond graph studies, the majority of the vehicle was analyzed.

### 6.1 Static Analysis - Front Suspension

Seen in Figure 6-1 is a SolidWorks model of half of the front suspension. Furthermore, Figure 6-2 illustrates the front suspension in a kinematic, linkage outline. Ultimately, the primary interest with the static analysis was to determine the internal forces endured by the front suspension components, as well as, the system's theoretical displacement.

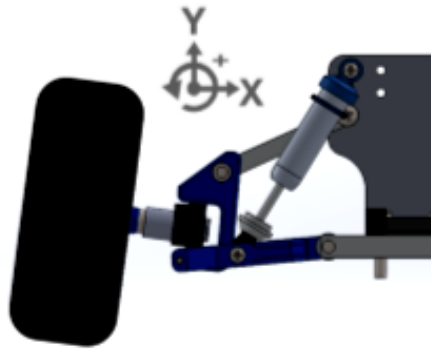


Figure 6-1: SolidWorks Model of Half of Front Suspension

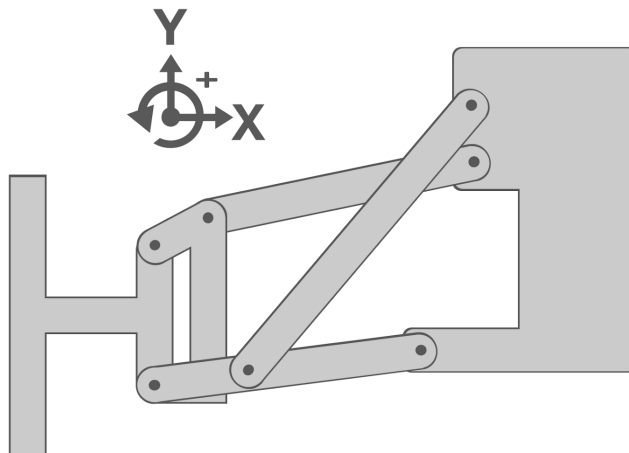


Figure 6-2: Kinematic Outline of Front Suspension

Using Figure 6-1 as a reference, the front suspension was first broken into its fundamental components. Following, free body diagrams of the forces undergone by each



component were produced. Finally, using these free body diagrams, the sum of the forces in the x and y direction were set to 0 and the sum of the moments in the z direction was set to 0.

Seen in Figure 6-3 is the first component analyzed. Consisting of the wheel, stub axle, and wheel mount, these pieces were simplified into one piece. Taking into account the 5 degree positive camber, the free body diagram of the simplified component can be seen in Figure 6-4. Set as the ground, (0,0), force A represented the normal force exerted by the surface on the wheel. While the normal force only consisted of a y-component for a static case, the x-component was included for clarity. At their corresponding locations, forces B and C represented the interaction at the interface between the wheel assembly and 2.

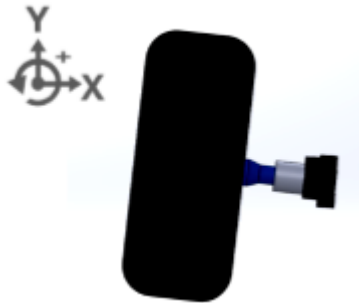


Figure 6-3: Wheel Assembly

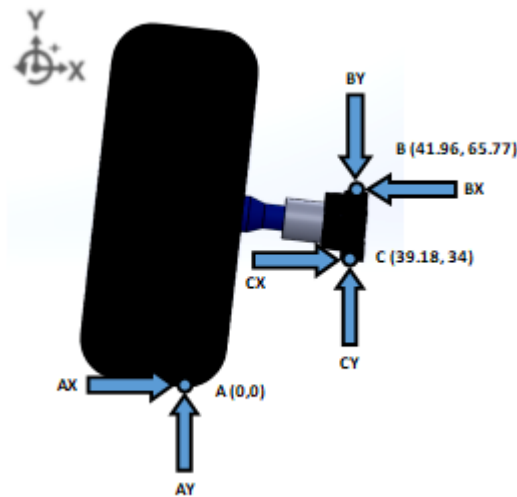


Figure 6-4: Free Body Diagram of Wheel Assembly

Due to static theory, the summation of the forces within the x and y direction had to be equal to 0. Additionally, the moment about an arbitrary force location had to be equal to 0. Given these fundamental laws of statics, the following equations were created.

$$\sum F_x = 0 = AX - BX + CX$$

$$\sum F_y = 0 = AY - BY + CY$$

$$\sum M_A = 0 = 65.77BX - 41.96BY - 34CX + 39.18CY$$

Seen in Figure 6-5 is the mechanism which allowed for articulation of the front suspension while also allowing for rotation and camber adjustment of the wheels. Taking into account the forces and locations of said forces, the free body diagram seen in Figure 6-6 illustrates the loads experienced by this piece. Particular interest should be devoted to the units of the locations of the forces within the free body diagrams. All force locations were specifically written in units of millimeters.



Figure 6-5: Wheel Mount Holder

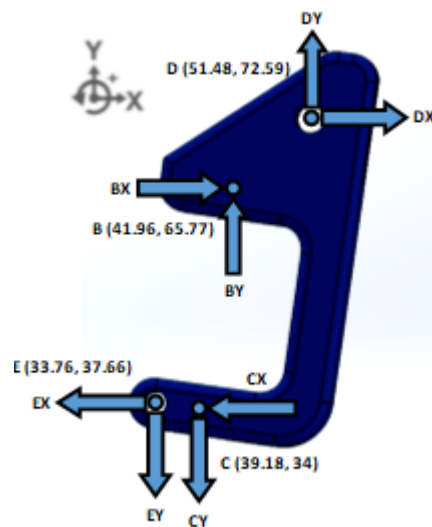


Figure 6-6: Free Body Diagram of Wheel Mount Holder

$$\sum F_x = 0 = BX - CX + DX - EX$$

$$\sum F_y = 0 = BY - CY + DY - EY$$

$$\sum M_B = 0 = -31.77CX + 2.78CY - 6.82DX + 9.52DY - 28.11EX + 8.2EY$$

Seen in Figure 6-7 is the camber link which controlled the camber of the front wheels. Its free body diagram in Figure 6-8 depicts illustrates the loads experienced by this component.



Figure 6-7: Camber Link

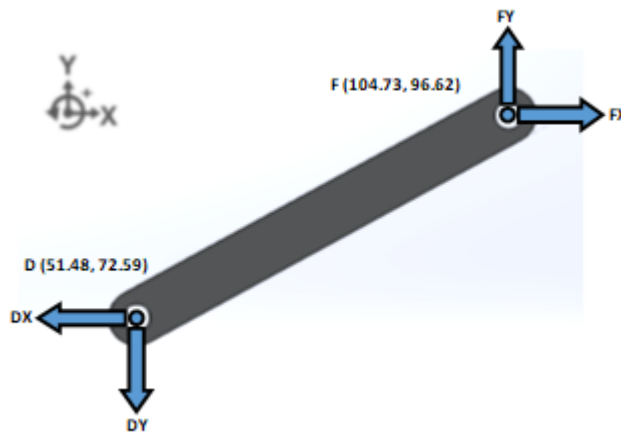


Figure 6-8: Free Body Diagram of Camber Link

$$\sum F_x = 0 = -DX + FX$$

$$\sum F_y = 0 = -DY + FY$$

$$\sum M_D = 0 = -24.03FX + 53.24FY$$

Seen in Figure 6-9 is the modified wishbone used to create an independent front suspension. As illustrated within the free body diagram of Figure 6-10, this piece experienced forces from its connection with the main frame and the wheels. Additionally, however, the wishbone had to account for the forces exerted by the shock absorbers.



Figure 6-9: Wishbone

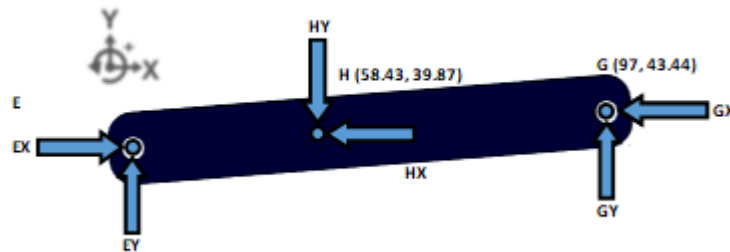


Figure 6-10: Free Body Diagram of Wishbone

$$\sum F_x = 0 = EX - GX - HX$$

$$\sum F_y = 0 = EY + GY - HY$$

$$\sum M_E = 0 = 5.78GX + 63.24GY + 2.21HX - 24.67HY$$

Seen in Figure 6-11 is the shock absorber. A very important piece for the front suspension, the shock absorber helps support the weight of the vehicle, while also allowing for articulation of the suspension. The forces undergone by the shock absorber can be seen in the free body diagram in Figure 6-12.



Figure 6-11: Shock Absorber

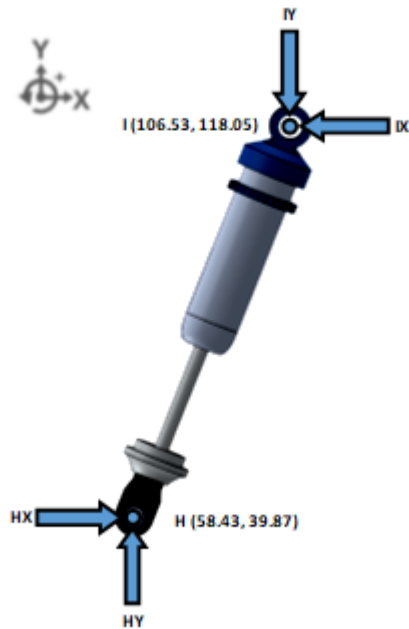


Figure 6-12: Free Body Diagram of Shock Absorber

$$\sum F_x = 0 = HX - IX$$

$$\sum F_y = 0 = HY - IY$$

$$\sum M_H = 0 = 78.18IX - 48.1IY$$

The final component of the front suspension is illustrated in Figure 6-13. The mounting mechanism, this component connected the wishbone to the chassis of the vehicle while also serving as a shock tower for the two front shock absorbers. The force body diagram on half of the mounting mechanism can be seen in Figure 6-14. Within the figure, the term 'Force' refers to one quarter of the car's total weight. This was based on the assumption that the weight of the car could be evenly divided into four concentrated forces at each wheel.



Figure 6-13: Shock Tower

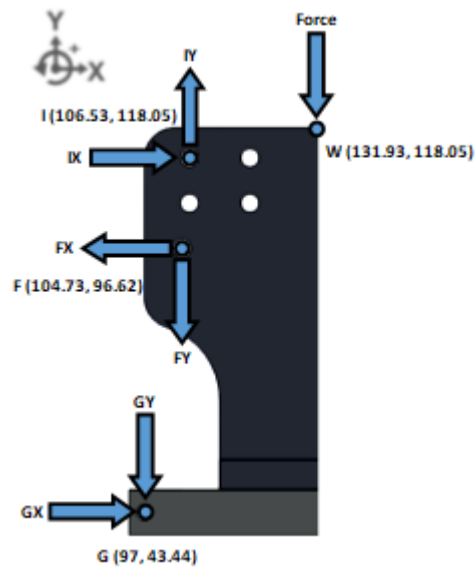


Figure 6-14: Free Body Diagram of Shock Tower

$$\sum F_x = 0 = -FX + GX + IX$$

$$\sum F_y = 7.01N = -FY - GY + IY - Force$$

$$\sum M_G = 0 = 53.18FX - 7.73FY - 74.61IX + 9.53IY = 244.86N - mm$$

Ultimately, while the weight of the car could have been represented as an external force at Point A, this analysis analyzed the forces at Point A as purely reactionary. Furthermore, while friction, suspension undulation, and centripetal force could have been integrated, the analysis represented the resting forces present within the front suspension with a static car.

Having obtained 18 equations representing the static conditions of the front suspension, a system of equations was created to determine the 18 unknowns. The determined forces at each component, in units of Newtons, can be seen in Table 6-1 below.

Table 6-1: Static Forces of Front Suspension

AX = 0N	DX = 25.5565N	GX = 8.2178N
AY = 7N	DY = 11.5350N	GY = 9.6367N
BX = 9.2874N	EX = 25.5565N	HX = 17.3387N
BY = 7.3409N	EY = 18.5450N	HY = 28.1817N
CX = 9.2874N	FX = 25.5565N	IX = 17.3387N
CY = 0.3309N	FY = 11.5350N	IY = 28.1817N

Modeling the entire front suspension system as a linkage, the displacements experienced by each component can be determined using dynamics equations. Using the forces endured by the shock absorber, the travel of the front suspension can be estimated. Using Hooke's Law,  $F = kx$ , the internal forces experienced by the shock absorber would produce a displacement indicating the car's equilibrium ride height. As time was limited, the experimental values for the spring and damping constants of the shock absorber were never determined. Consequently, the equilibrium compression of the front shock absorbers cannot be currently calculated.

## 6.2 Bond Graph Analysis

Bond graph analysis was performed separately on the gearbox and suspension system. The analysis was done not to create templates that describe the systems. These template will be experimentally tested and improved by future teams to develop models that adequately describe the actual systems. Included in each system template is a system diagram, a bond graph, a list of the assumptions present in the bond graph, and a differential matrix of the system's state equations. There were several nomenclature formats used during the bond graph analysis of the gearbox. Refer to the list of nomenclature for nomenclature format clarification.

### Gearbox

Due to the complexity of real systems, the team simplified the bond graph by making several assumptions about the gearbox. The first assumption the team made was that the gears in the gearbox were perfectly machined and mesh perfectly. This means that damping and spring effects between gears did not need to be considered in the model. The team also assumed that all rotations within the gearbox are small, allowing the team to use the equation  $v = \omega r$  to relate velocity to angular velocity. Additionally, angular moments  $J_1$ ,  $J_4$ , and  $J_7$  were assumed to occur at gear A,  $D_2$ , and  $D_7$  respectively. This meant that these inertias act at the

corresponding locations and can therefore be placed in those locations in the bond graph. Lastly, the components connecting to the output shaft of the gearbox were assumed to have a reactionary torque and were therefore modeled as a source of effort. Figure 6-15 contains a diagram labeling the components, dampers, inertial masses, and springs in the gearbox and the bond graph of the gearbox can be seen below in Figure 6-16.

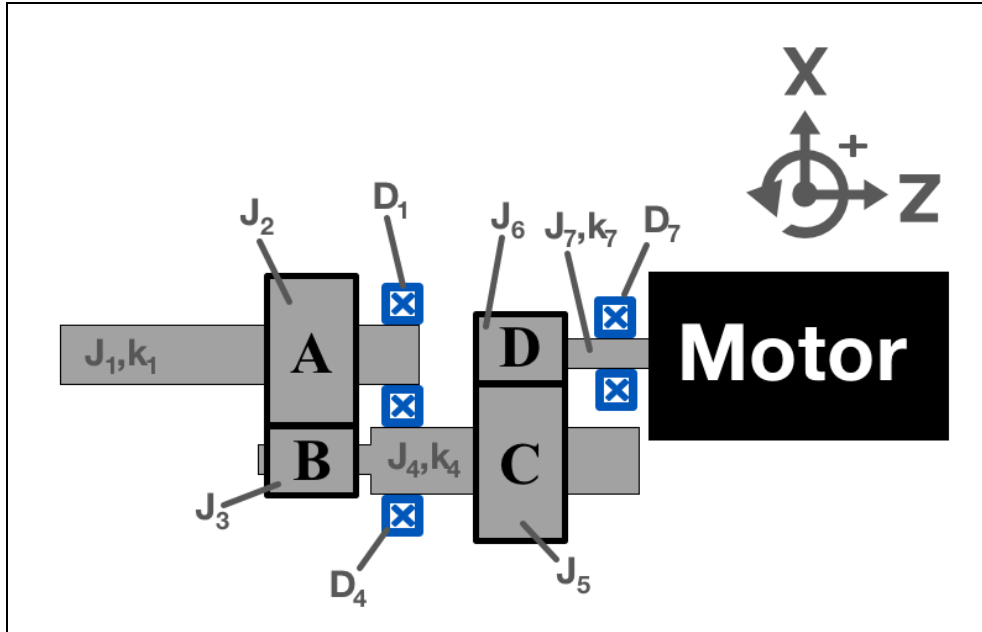


Figure 6-15: Labeled Gearbox Diagram

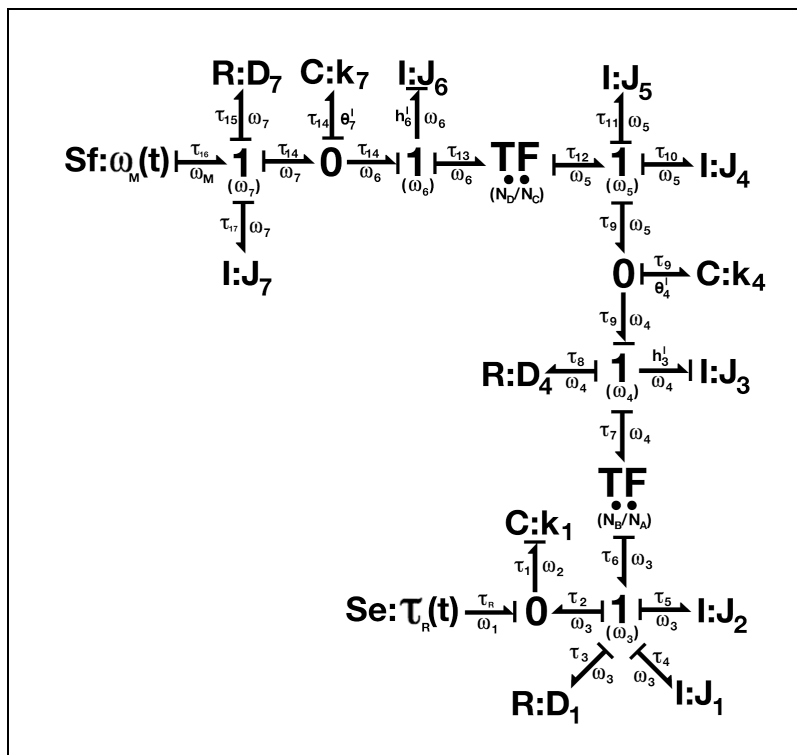


Figure 6-16: Gearbox Bond Graph



From the bond graph, the team found four state equations. Then, using the flows and efforts outlined in the bond graph, the team developed a differential matrix to describe the state equations. The matrix and other related equations are shown in Figure 6-17. The accompanying equations and their derivations used to develop the matrix are shown in appendix C.

$$n = 4: \theta_7', h_6', \theta_4', h_3' = F(\theta_7, h_6, \theta_4, h_3, \omega_M, \tau_R)$$

$$\begin{bmatrix} \theta_7' \\ h_6' \\ \theta_4' \\ h_3' \end{bmatrix} = M \begin{bmatrix} \theta_7 \\ h_6 \\ \theta_4 \\ h_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \omega_M + \begin{bmatrix} 0 \\ 0 \\ 0 \\ c \end{bmatrix} \tau_R$$

$$M = \begin{bmatrix} 0 & -J_6^{-1} & 0 & 0 \\ \frac{J_6 k_7}{J_5 + J_4 + J_6} & 0 & \frac{-N_C J_6 k_4}{N_D (J_5 + J_4 + J_6)} & 0 \\ 0 & \left( \frac{N_C}{N_D J_6} \right) & 0 & -J_3^{-1} \\ 0 & 0 & \frac{J_3 k_4 \theta_4}{J_1 + J_2 + J_3} & \frac{-D_4 - D_1}{J_1 + J_2 + J_3} \end{bmatrix}$$

$$c = \frac{-J_3 N_A}{N_B (J_1 + J_2 + J_3)}$$

Figure 6-17: Gearbox matrix equations

## Suspension

Similar to actual gearboxes, actual suspension systems are very complex. To simplify this suspension system, the team made several assumptions. The first assumption the team made was that the front and rear suspension on the right side of the car was identical to the front and rear suspension on the left side. This allowed the team to use the half car model to analyze the suspension. The team also assumed that links connecting the wheels to the body car and the rubber on the wheels can be modeled as a spring and damper in parallel. The third assumption the team made was that the chassis of the car is rigid and therefore does not bend. Lastly, the team assumed that forces in the Z direction and masses one and two are negligible. As a result all of these components were neglected. Figure 6-18 contains a diagram of the half car model, damping constants, inertials, and spring constants used for the analysis. Additionally, the bond graph of the suspension system can be seen in Figure 6-19. [69]

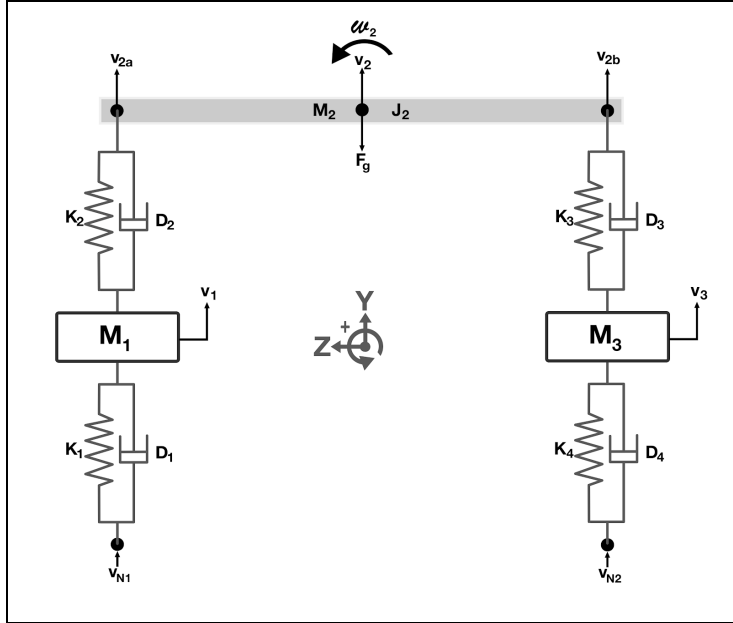


Figure 6-18: Half car model for suspension

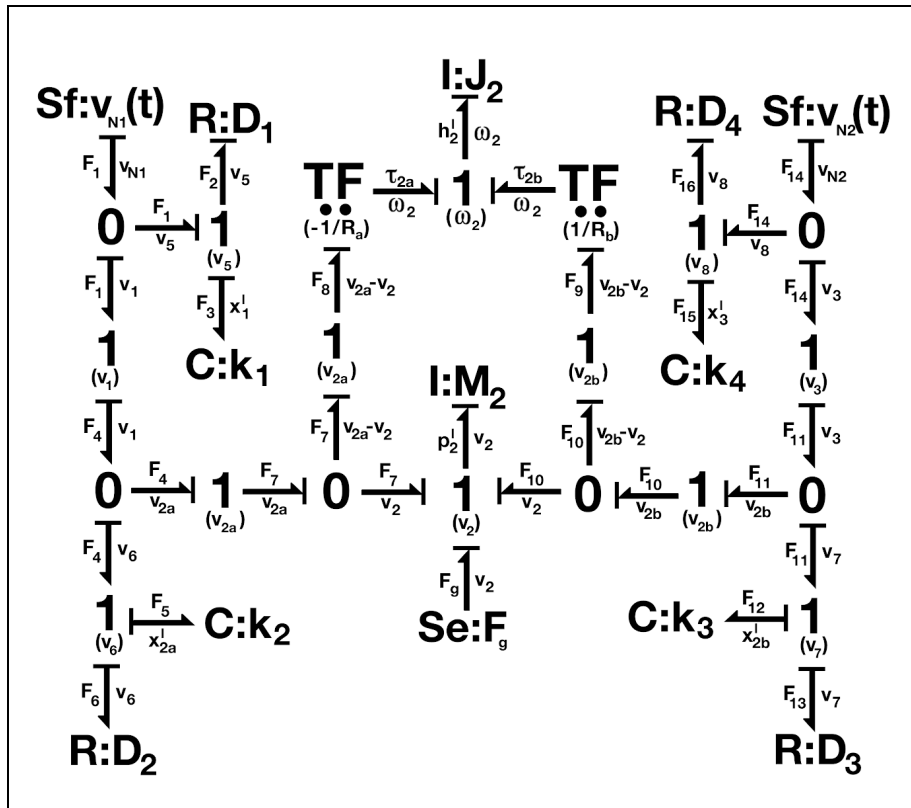


Figure 6-19: Suspension Bond Graph

From the bond graph, the team found six state equations. Each state equation was then solved for using the flows and efforts in the bond graph. Lastly, using these equations a differential matrix was created that describes the state equations. The state equations and differential matrix can be seen below in Figure 6-20. By obtaining D through experimentation, the state equations can be solved. For additional information, both the accompanying equations and their derivations used to develop the matrix are shown in appendix C.

$$n = 6: x'_1, x'_{2a}, p'_2, h'_2, x'_{2b}, x'_3 = F(x_1, x_{2a}, p_2, h_2, x_{2b}, x_3, v_{N1}, v_{N2}, F_g)$$

$$\begin{bmatrix} x'_1 \\ x'_{2a} \\ x'_3 \\ x'_{2b} \\ p'_2 \\ h'_2 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x_1 \\ x_{2a} \\ x_3 \\ x_{2b} \\ p_2 \\ h_2 \end{bmatrix} + \begin{bmatrix} a \\ b \\ 0 \\ 0 \\ e \\ g \end{bmatrix} \mathbf{v}_{N1} + \begin{bmatrix} 0 \\ 0 \\ c \\ d \\ f \\ h \end{bmatrix} \mathbf{v}_{N2} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \mathbf{F}_g$$

$$\mathbf{M} = \begin{pmatrix} \frac{-k_1}{D_1 + D_2} & \frac{k_2}{D_1 + D_2} & 0 & 0 & \frac{-D_2}{m_2 \left(1 + \frac{D_2}{D_1}\right)} & \frac{D_2 R_a}{J_2 (D_1 + D_2)} \\ \frac{k_1}{D_2 - D_1} & \frac{k_2}{D_1 - D_2} & 0 & 0 & \frac{D_1}{m_2 (D_1 - D_2)} & \frac{D_1 R_a}{J_2 (D_2 - D_1)} \\ 0 & 0 & \frac{-k_3}{D_4 + D_3} & \frac{k_3}{D_4 + D_3} & \frac{-D_3}{m_2 \left(1 + \frac{D_3}{D_4}\right)} & \frac{-D_3 R_b}{J_2 (D_4 + D_3)} \\ 0 & 0 & \frac{k_4}{D_3 - D_4} & \frac{k_3}{D_4 - D_3} & \frac{-D_4}{m_2 (D_3 - D_4)} & \frac{D_4 R_b}{J_2 (D_4 - D_3)} \\ \frac{k_1}{1 - \frac{D_1}{D_2}} & k_2 - \frac{k_2}{\left(1 - \frac{D_1}{D_2}\right)} & \frac{k_4}{1 - \frac{D_4}{D_3}} & k_2 - \frac{k_3}{\left(1 - \frac{D_4}{D_3}\right)} & \frac{-1}{m_2} \left( \frac{D_1}{\left(1 - \frac{D_1}{D_2}\right)} + \frac{D_4}{\left(1 - \frac{D_4}{D_3}\right)} \right) & \frac{1}{J_2} \left( \frac{D_1 R_a}{\left(1 - \frac{D_1}{D_2}\right)} - \frac{D_4 R_b}{\left(1 - \frac{D_4}{D_3}\right)} \right) \\ \frac{k_1 x_1}{R_a \left(1 - \frac{D_1}{D_2}\right)} & \frac{-k_2}{R_a} \left(1 + \frac{1}{\left(1 - \frac{D_1}{D_2}\right)}\right) & \frac{k_4 x_3}{R_b \left(1 - \frac{D_4}{D_3}\right)} & \frac{k_2}{R_b} \left(1 - \frac{1}{\left(1 - \frac{D_4}{D_3}\right)}\right) & \frac{1}{m_2} \left( \frac{D_1}{R_a \left(1 - \frac{D_1}{D_2}\right)} - \frac{D_4}{R_b \left(1 - \frac{D_4}{D_3}\right)} \right) & \frac{-1}{J_2} \left( \frac{D_1 R_a}{R_a \left(1 - \frac{D_1}{D_2}\right)} + \frac{D_4 R_b}{R_b \left(1 - \frac{D_4}{D_3}\right)} \right) \end{pmatrix}$$

$$a = \frac{D_2}{D_1 + D_2} \quad b = \frac{D_1}{D_2 - D_1} \quad c = \frac{D_3}{D_4 + D_3} \quad d = \frac{D_4}{D_3 - D_4}$$

$$e = \frac{D_1}{1 - \frac{D_1}{D_2}} \quad f = \frac{D_4}{1 - \frac{D_4}{D_3}} \quad g = \frac{D_3}{R_a \left(1 - \frac{D_1}{D_2}\right)} \quad h = \frac{D_4}{R_b \left(1 - \frac{D_4}{D_3}\right)}$$

Figure 6-20. Suspension Bond Graph Matrix

### 6.3 Finite Element Analysis for Von Mises Stress

In order to determine the critical points on several of the more complex components, we used Finite Element Analysis (FEA). FEA works by taking complex parts and breaking them into smaller shapes that can be analyzed more easily. Each finite element is then analyzed and then the elements are combined back into the complex shape, in order to analyze the entire model. FEA can be used to do several types of analysis, such as static, fatigue, buckling, and thermal analysis. In our project, we used FEA to locate the critical regions, where there was the most stress on the part, as well as the location of the maximum deflection. Below, you can find the process for setting up and running a static analysis using solidworks simulation.

The first step in static FEA analysis is to assign a material to the part. Many of the calculations performed by the software are dependant on the material properties of the part. This can be done by selecting a material from the materials database, or by adding a custom material.

With a material selected, the next step was to define the fixturing of the part. Fixtures tell the FEA software how the part is held in place. There are several types of fixtures, and you should select the type that matches how the part is fixtured. The most common fixture types are fixed, fixed hinge, roller slider, and elastic support.

After the fixturing has been defined, the next step is to define the loads on the part. There are several types of loads that can be applied such as forces, torques, pressures, and gravity. In order to apply these loads, the program asks you for the magnitude of the load and the location. Once you have defined the material, fixturing, and loading, you are ready to run a basic study. For the servo horn in our steering assembly, the component can be seen below.

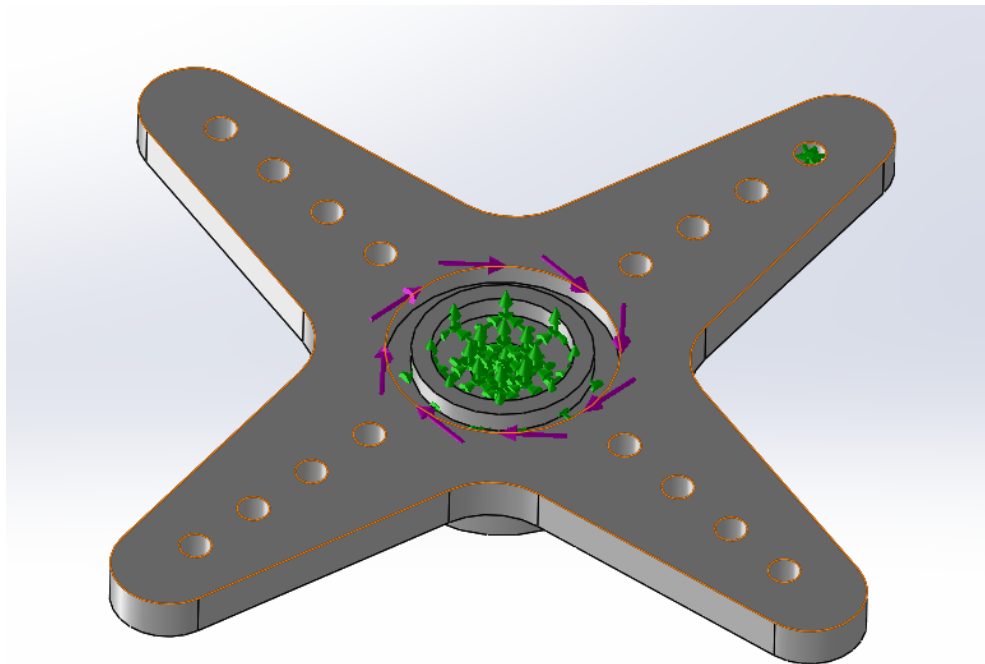


Figure 6-21: SolidWorks FEA of Servo Horn

As you can see in the Figure 6-21, the green arrows represent fixturing locations and the purple arrows represent the loads. In the part above, the part is rigidly fixed about the central axis. On this servo horn, the part is fixtured through the central hole where the screw is used to keep it rigidly attached to the servo. The servo provides a torque on the part through the central axis. The magnitude of the torque was found from the servo specifications provided by the servo manufacturer. There is a pin through the outermost hole on the appendage that then connects to the next link in the steering system, which acts as a rigid fixture in this simulation.

Once we have defined the fixtures and loads on the servo horn, we run the study to calculate the critical points. This creates a mesh which breaks your component into finite elements and analyzes each element individually. One thing to be aware of, is that the program

show a deformed result that is exaggerated when compared to the actual deformation seen in the part. The Von Mises Stress results can be seen in Figure 6-22.

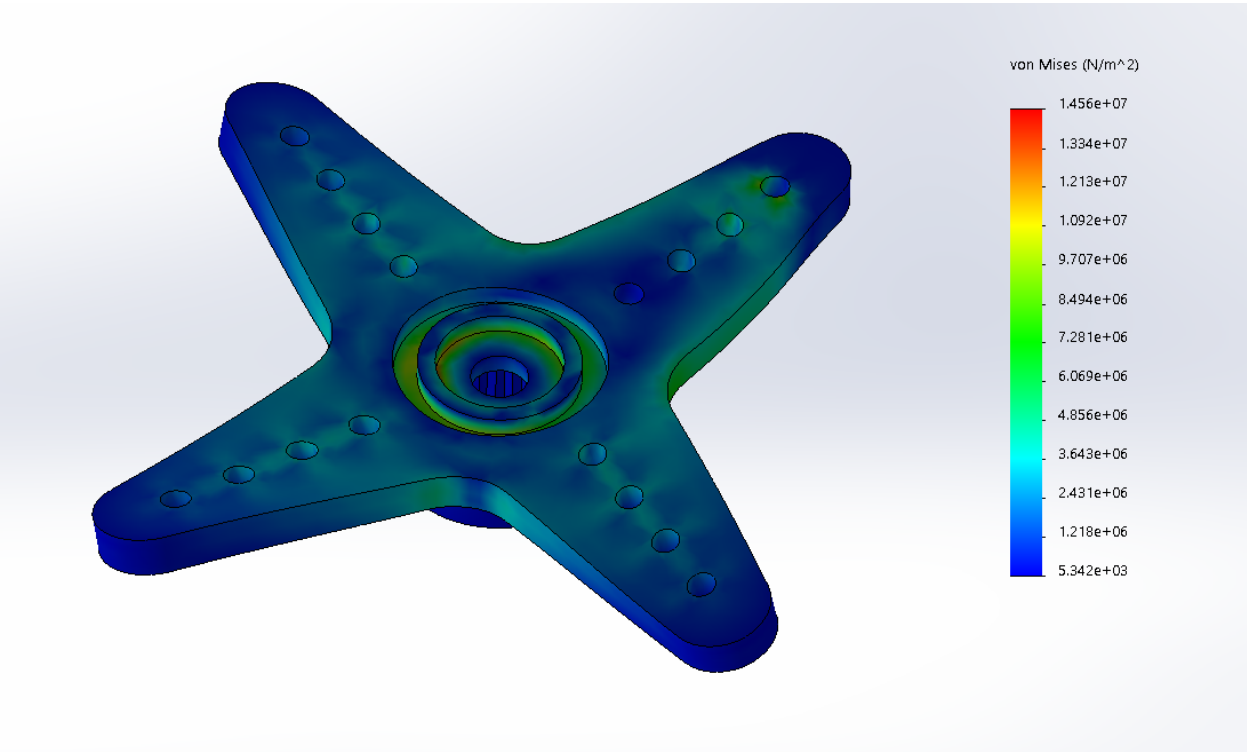


Figure 6-22: Von Mises Stress of Servo Horn

In Figure 6-22, the areas of higher stress are represented by warmer colors, which helps visually identify the locations of maximum stress. The values used to set up this analysis can be seen Table 6-2 below.

Table 6-2: Simulation Parameters for Von Mises Stress

<b>Material Used</b>	<b>ABS</b>	
Elastic Modulus	2000000000	Pa
Poisson's Ratio	0.394	
Mass density	1020	kg/m <sup>3</sup>
Tensile Strength	30000000	Pa
Shear Modulus	318900000	Pa
<b>Fixture Type</b>	Location	
Fixed Geometry	Countersunk Screw Hole	
Fixed Geometry	Pin Hole for Steering Arm Connection	
<b>Applied Load</b>		
Torque	146	oz-in
<b>Results</b>		
Max Von Mises Stress	24700000	Pa
Maximum Displacement	0.2317	mm
Maximum Strain	0.0147	

These locations where the stress is maximum are the locations where the part is most likely to break. You can see another example of this in Figure 6-23.

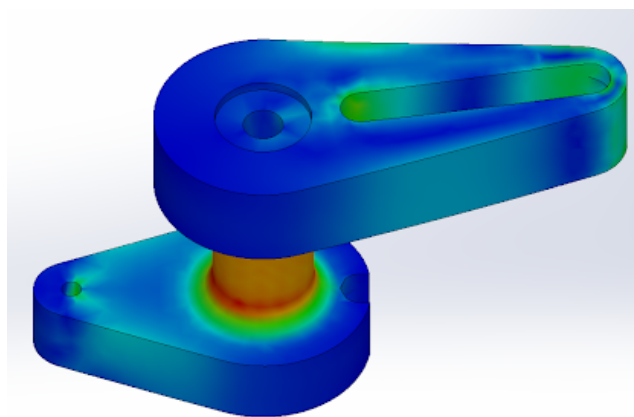


Figure 6-23: Von Mises Stress of Steering Pivot

In the part above from the steering assembly, the FEA analysis shows us that the part is most likely to break from torsional stress around the center of the column. When analyzing this

component, the results of the static analysis for the steering system was used in order to determine what the resulting forces were on the component and where they were located.

In the steering are pictured in Figure 6-24 below, we can see that the maximum shear stress is actual located at the holes where the pins go that connect arm to the steering posts.

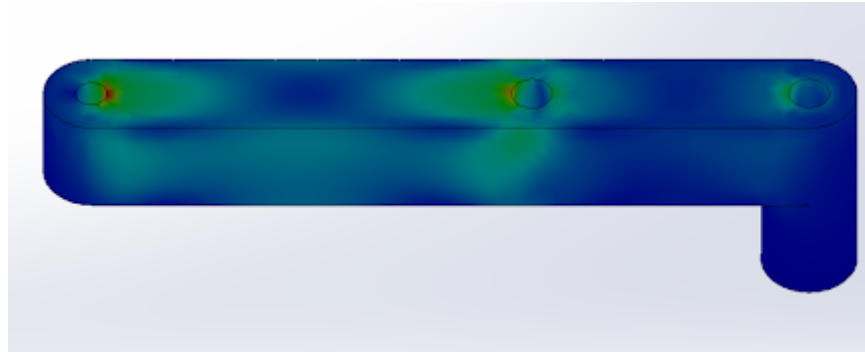


Figure 6-24: Von Mises Stress of Steering Arm

Results of static Finite Element Analysis of stress and displacement of several components are shown in Appendix A.

#### 6.4 Static and Dynamic Analysis for Steering System

Static and dynamic analysis was done on a simplified two dimensional model of the car. This model assumes that the links are rigid, that all links are made of Polylactic Acid by 3D printing, the forces in the third dimension are negligible, the joints connecting links are frictionless and massless, the links and masses are small enough to be considered point masses, links GI and HK are assumed to be only in the Y direction, and the center of mass of each link is considered to be the center of the link. The diagram of the segments is shown in Figure 6-25 and the methods and solutions are in Appendix H. Using the input torque of the servo motor, which was 19.04 lb-in, the torque at the wheel's point of rotation, points J and L when solved statically was 11.7 lb-in and was 9.8454 lb-in when solved dynamically. Both of these values allow the steering system to overcome the friction of the large wheels on the ground to make sure that the car handles in an expected way and can always turn the wheels.

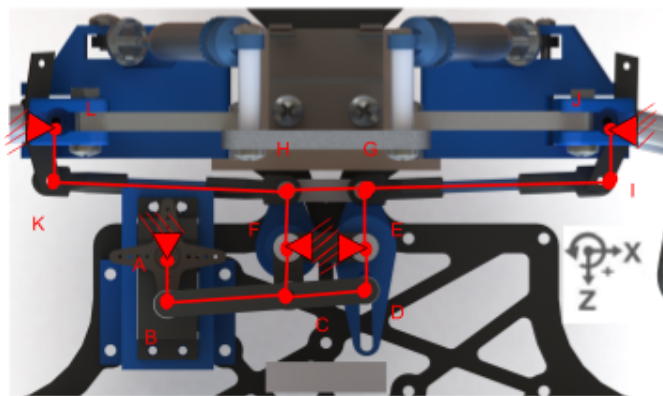


Figure 6-25: Steering Linkage System

## **7.0 Sensor Dashboard**

The fundamental goal of the sensor dashboard was to create a sensor system that could monitor a scale car's performance. This system is intended to be integrated into ME4320 course. Within the course, students will be tasked with implementing the sensor system into their manufactured cars. The collected data is intended to be used as a validation for the theoretical analysis performed in the course. In order to make the sensor system compatible with the various car designs produced in the course, the system was designed to be both modular and customizable.

This chapter is divided into three sections. The first section discusses the sensors used in the dashboard. The second section discusses the implementation of the sensors onto the existing car hardware. The third section discusses the modularity of the dashboard. Finally, the last section will discuss the real-time capabilities of the sensor dashboard.

### **7.1 Sensors**

There are numerous measurements that can be used to verify the theoretical calculations done on the ME4320 cars. These measurements include component travel, component stress and strain, shaft rotation, heat generation, spring forces, and damping forces. The measurements selected for the initial design of the sensor dashboard were heat generation, component travel, shaft rotation. These measurements were included because the equipment required to take these measurements, an IMU, a tachometer, and temperature sensors, are inexpensive, not excessively complex, and easily scalable.

#### **7.1.1 Thermal System**

As the scale car's control system consists of many electrical and mechanical components that are sensitive to heat, it was important to monitor the heat generated in the car and if needed regulate the heat flow within the vehicle. In order to do this, temperature sensors and a cooling fan were both added to the sensor dashboard. The temperature was intended to be monitored in real-time and when the system was too hot, the cooling fan would turn on.

#### **Temperature Sensor**

Of the numerous temperature sensors available, the team narrowed the selection to the DHT11 and the DHT22. This was done due to the available and inexpensiveness of the temperature sensors. After further investigation, it was determined that the DHT-22 was better suited for the team's needs than the DHT11. This was due to the larger temperature range that the sensor could measure with a minimal cost increase. An image for the DHT22 temperature sensor and more information on both of the sensors can be seen in Figure 7-1 (reproduced as is from [71]) and Table 7-1, respectively.



Table 7-1: DHT22 vs. DHT11 Temperature Sensors [71][72]

Temperature Sensor	DHT 22	DHT 11
Power Supply	3.3-6V DC	3-5.5V DC
Temperature Range	-40 to 80 Celcius	0-50 Celsius
Measures Humidity	0 to 100% with 2-5% accuracy	humidity 20-90% with 4-8% accuracy
Average Sensing Period	2 s	2 s
Dimensions	14*18*5.5mm	12*15.5*5.5mm
Weight	0.32 ounces	0.1 ounces
Cost	\$ 10	\$ 8

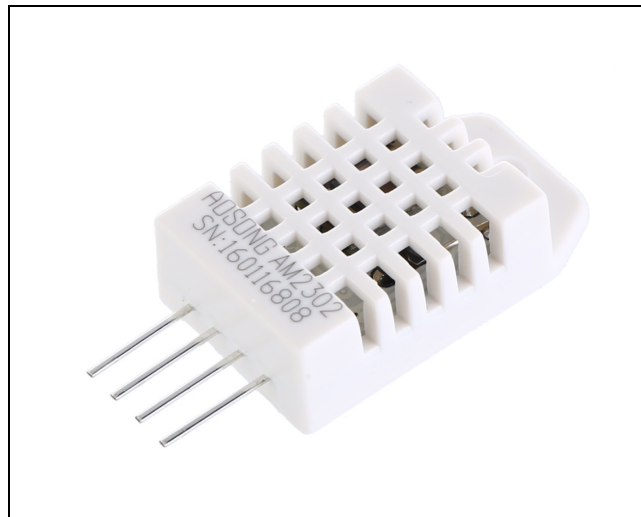


Figure 7-1: DHT22 Temperature Sensor. Reproduced as is from [71].

### Cooling Fan

For the cooling fan, a 4-pin DC cooling fan was chosen over a regular 3-pin DC cooling fan. A 4-pin fan was chosen over a 3-pin fan because 3-pin fans require additional external circuit to control the fan's speed control whereas, no such circuitry would be necessary while using a 4-pin fan. The additional circuitry was a deterrent because it increased both the complexity and cost, consequently reducing its scalability. The 4-pin fan that was chosen was the Noctua NF-A4x10 due to its low power and space requirements. The fan has a maximum input power of 0.35W and was only 4cm tall, 4cm wide, and 1cm in depth. An image for the Noctua NF-A4x10 and more details about the fan can be seen in Figure 7-2 (reproduced as is from [72]) and Table 7-2, respectively.



Figure 7-2: Noctua NF-A4x10 cooling fan. Reproduced as is from [73].

For the 4-pin fan, Noctua NF-A4x10 5V PWM cooling fan was selected and integrated into the thermal system (Figure 7-2 reproduced as is from [73]). The fan specifications are listed in Table 7-2. [73]

Table 7-2: Cooling Fan Specifications [73]

Cooling Fan	NF-A4x10 5V PWM
Size	40 x 40 x 10 mm
Max. rotational speed (+/-10%)	5000 RPM
Max. airflow	8.9 m <sup>3</sup> /h
Max. acoustical noise	19.6 dB(A)
Max. static pressure	1.95 mm H <sub>2</sub> O
Max. input power / operating voltage	0.35W / 5V
Connector	4 - Pin

### 7.1.2 Inertial Measurement Unit

There are many IMUs that are commercially available. Pololu MinIMU-9 was selected for due to the extensive documentation provided by the manufacturer. Additionally, it was selected because is small, can be easily used with an Arduino and was readily accessible to the team. Figure 7-3 illustrates the Pololu MinIMU-9's size, axis orientation and pins. Specifications of the Pololu MinIMU-9 are tabulated in Table 7-3

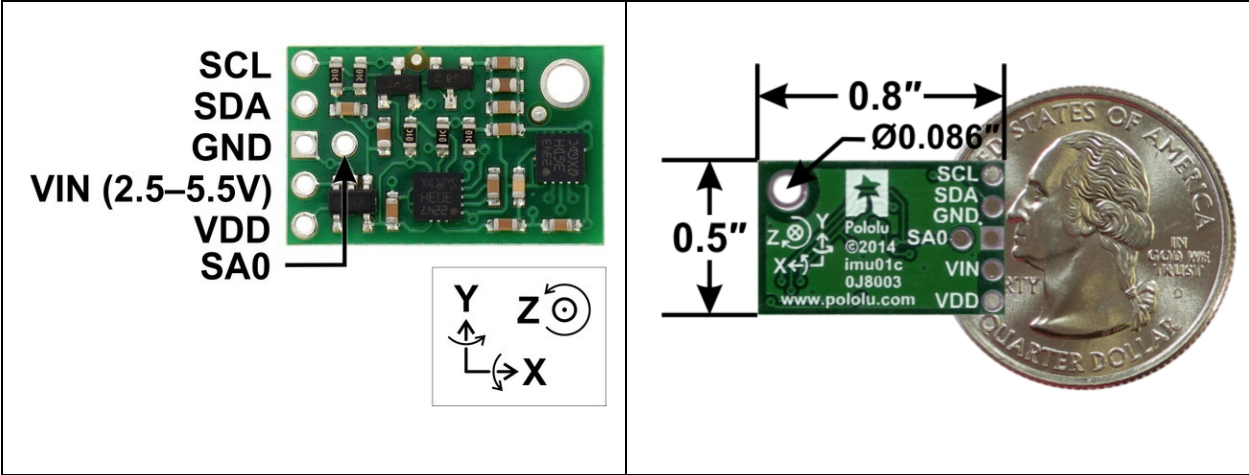


Figure 7-3: Pololu MinIMU-9 v3. Reproduced as is from [74].

Table 7-3: Pololu MinIMU-9 v3 specifications [75]

Dimensions	0.8" x 0.5" x 0.1"
Weight without header pins	0.7 g (0.02 oz)
Operating voltage	2.5 to 5.5 V
Supply current	10 mA
Output format (I <sup>2</sup> C)	Gyro: one 16-bit reading per axis
	Accelerometer: one 12-bit reading (left-justified) per axis
	Magnetometer: one 12-bit reading (right-justified) per axis
Sensitivity range (configurable)	Gyro (°/s): ±250, ±500, or ±2000
	Accelerometer(g): ±2, ±4, ±8, or ±16
	Magnetometer (gauss): ±1.3, ±1.9, ±2.5, ±4.0, ±4.7, ±5.6, or ±8.1

**7.1.3 Tachometer**

For simplicity, the tachometer used for the sensor module was an electromagnetic, non-contact tachometer using a hall-effect sensor and a magnet. The selected hall-effect sensor was the KY-003 (Figure 7-4, reproduced as is from [76]) which was accompanied with the use of neodymium magnets (Figure 7-5, reproduced as is from [77]).

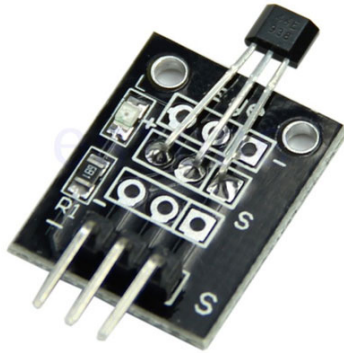


Figure 7-4: KY-003 Hall-Effect Sensor. Reproduced as is from [76].



Figure 7-5: Neodymium Magnets. Reproduced as is from [77].

## 7.2 Implementation

In order to make the sensor dashboard fully functional, each sensor needed to be calibrated and tested before being integrated into one functional sensor dashboard. This section is a detailed methodology of the design process that was carried out to implement the thermal system, IMU and tachometer. These components are controlled and monitored by an Arduino, as shown below in Figure 7-6.

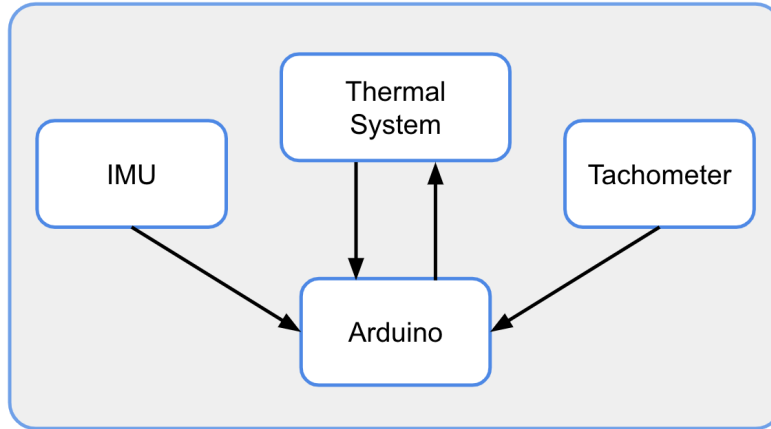


Figure 7-6: Arudino and sensor communication

### 7.2.1 Thermal System

As discussed previously, the thermal system is comprised of a cooling fan (Noctua NF-A4x10) and temperature sensors (DHT22s). The temperature sensors are intended to monitor the heat generated by mechanical and electrical systems. When a threshold temperature has been reached the cooling fan is turned on to regulate the heat flow. The layout of the thermal system is shown below in Figure 7-7.

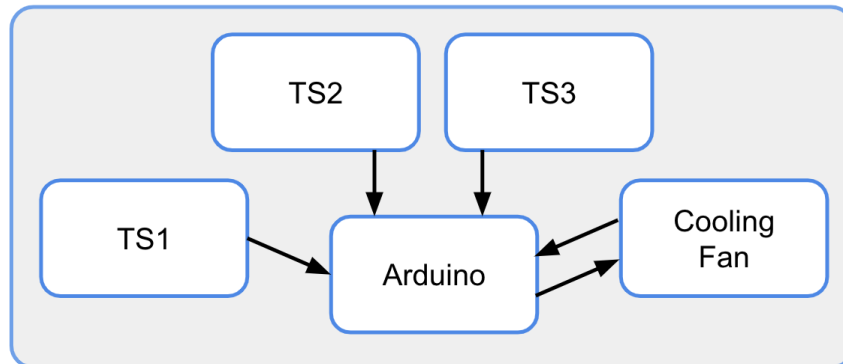


Figure 7-7: Arudino communication with the thermal system

### Temperature Sensors

To implement the DHT22 temperature, pins 1, 2, and 4 of the of DHT22 were connected to the, 5V, PWM 2, and ground pins of an Arduino mega, respectively. Pin 3 of the DHT22 is not used. Additionally, a 10K $\Omega$  pull up resistor was used at pin 2 of the DHT22. A diagram of the circuit can be found below in Figure 7-8.

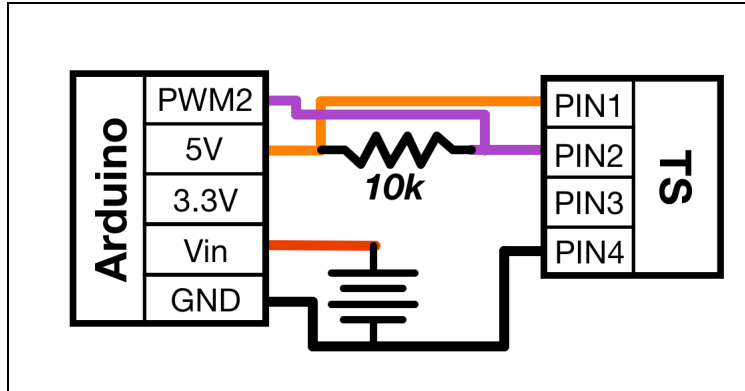


Figure 7-8: DHT22 Temperature sensor circuit

Using the template described above, more DHT22 temperature sensors could be added to the Arduino, allowing the temperature to be monitor at multiple locations. The team decided to use three temperature sensors. The number of sensors can be increased or decreased however to fit a ME4320 team's individual needs. A test circuit that used the three temperature sensors is shown in Figure 7-9.

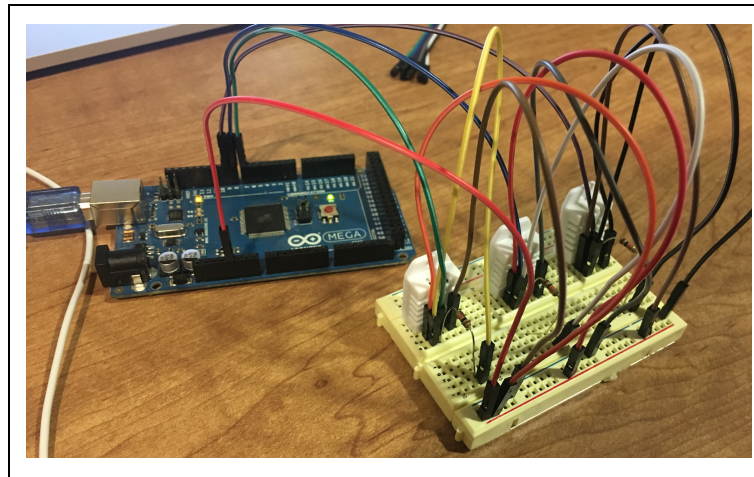


Figure 7-9: Multiple DHT22 temperature sensors circuit

The Arduino program that used the temperature sensors, read multiple temperature data points from each of the three temperature sensors and then outputted an average of the temperatures measured at each sensor, as seen in Figure 7-10. The Arduino code for installing these temperature sensors and obtaining an average temperature is shown in Appendix D. The average temperatures were then used to control the cooling fan. The logic that governed the fan actuation is explained in the next section.

```
Humidity: 30.60 %, Temp: 23.10 Celsius
Humidity: 31.10 %, Temp: 24.50 Celsius
Humidity: 29.80 %, Temp: 23.90 Celsius
Average Temp: 23.83 Celsius
Humidity: 38.70 %, Temp: 23.20 Celsius
Humidity: 38.70 %, Temp: 25.00 Celsius
Humidity: 34.20 %, Temp: 24.30 Celsius
Average Temp: 24.17 Celsius
Humidity: 69.40 %, Temp: 23.40 Celsius
Humidity: 58.20 %, Temp: 25.70 Celsius
Humidity: 54.60 %, Temp: 25.00 Celsius
Average Temp: 24.70 Celsius
```

Figure 7-10: Average temperature reading output to the Arduino's serial monitor

Once the temperature sensor circuit template had been made and Arduino code that could read the average temperature sensors were complete, three strategic temperature sensors were chosen. The locations were chosen based on which parts of the custom car were most prone to heat. These locations were beside the two Raspberry Pis and under the motor, each of which is a major heat producing unit on the custom car. Figure 7-11 shows the locations of the temperature sensors on the custom car.

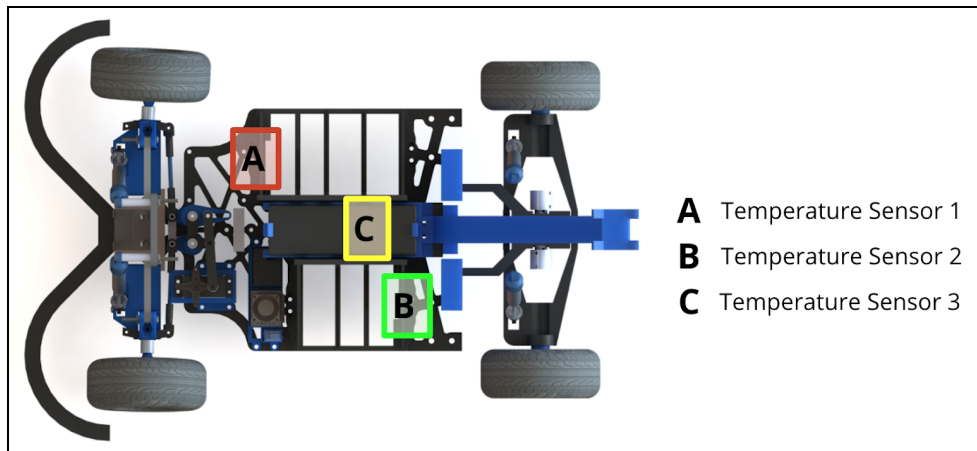


Figure 7-11: Temperature sensor locations

### Cooling Fan

To implement the selected cooling fan, the VCC, Ground, and PWM pins of the of Noctua NF-A4x10 were connected to the, 5V, ground, and PWM 9 pins of an arduino mega, respectively. The tach pin on the NF-A4x10 is not used. A diagram of the circuit can be found below in Figure 7-12.

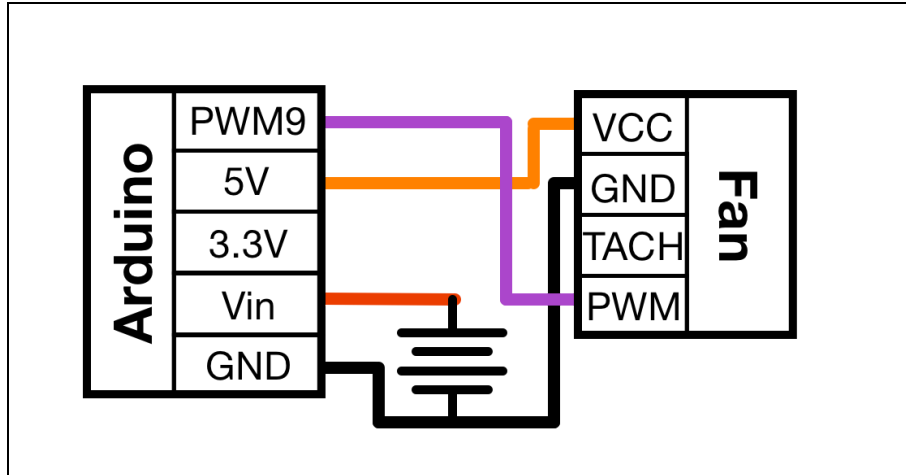


Figure 7-12: NF-A4x10 circuit

The fan speed was controlled by a PWM signal sent from the Arduino. The signal sent from the Arduino is set by using the analog write command with an input parameter between 0 - 255. Setting the fan speed to 255 results in PWM signal with a 100% duty cycle i.e. full voltage of 5V to achieve maximum rotations per minute. Other such values than can be used to control the duty cycle, which in turn controls the fan speed, are tabulated in the Table 7-4 below.

Table 7-4: PWM pin values and corresponding duty cycles

<i>PWM pin value</i>	<i>Duty cycle</i>	<i>Rpm (fan speed)</i>
0-1	0%	OFF
85	25%	One fourth speed
127	50%	Half the maximum speed
170	75%	75 % of the maximum speed
225	100%	Maximum speed

For the fan placement, the team determined that the center of the car would be the best place for the fan. The team chose this location because it had the most restricted air flow and had the highest concentration of heat generation, which came from the motor and Raspberry Pis. The combination of these two facts made the center of the car requiring the most heat regulation. Figure 7-13 depicts the location on the custom car where the fan was placed.



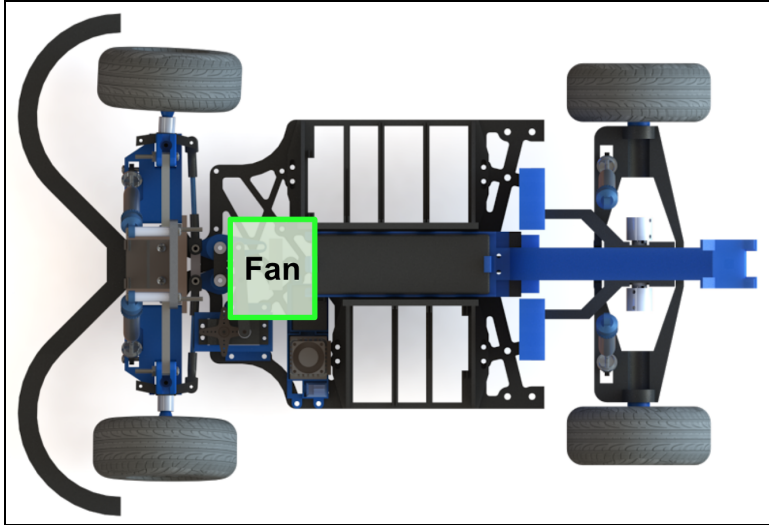


Figure 7-13: Cooling Fan Placement

### Thermal Control

After setting up the three temperature sensors and a cooling fan individually. The next step was to combine them. A diagram of the integrated temperature sensors and cooling fan can be seen below in Figure 7-14. Additionally, an image of the physical circuit can be seen in Figure 7-15.

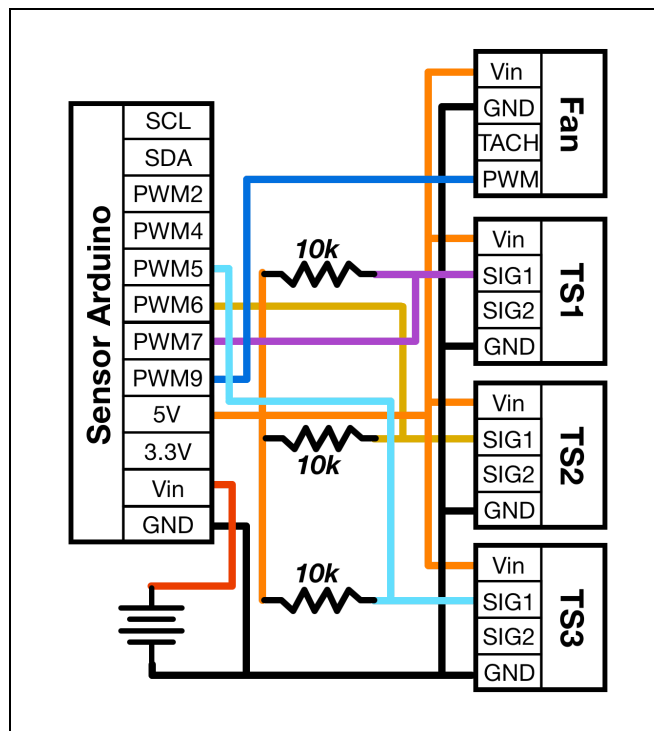


Figure 7-14: Circuit diagram of integrated temperature sensors and cooling fan

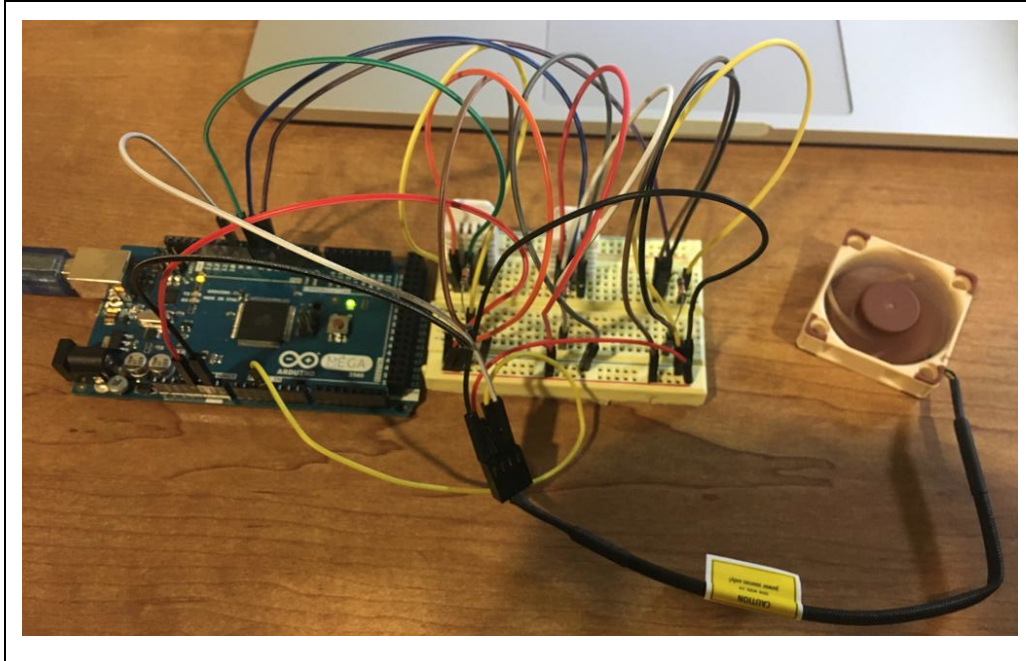


Figure 7-15: Temperature sensors and cooling Fan Integrated into the Thermal System

Once combined, the Arduino was programmed in such a way that the fan actuated in accordance with the average temperature readings from the temperature sensors. At higher temperatures the fan would function at full throttle, i.e at maximum capacity to bring down the internal temperature. At lower temperatures, the fan would be set to a lower speed to save power. The relationship between the average temperature and the PWM output to the fan was determined based on Table 7-5. The resulting output from the integrated system based on the imputed measured temperature can be shown in Figure 7-16.

Table 7-5: Arduino PWM output based on measured temperature

Temperature (Celcius)	PWM	Duty Cycle Max = 100%
0-15	85	33%
15-20	127	50%
20-25	170	66%
>25	255	100%

14	.....Average Temperature
85	.....Duty Cycle
.....	
18	.....Average Temperature
127	.....Duty Cycle
.....	
22	.....Average Temperature
170	.....Duty Cycle
.....	
26	.....Average Temperature
255	.....Duty Cycle
.....	
30	.....Average Temperature
255	.....Duty Cycle
.....	
34	.....Average Temperature
255	.....Duty Cycle

Figure 7-16: Output to fan from the Arduino based on the average temperature

### 7.2.2 Inertial Measurement Unit

To add odometry measurements to the sensor dashboard, the team used an IMU (the Pololu MinIMU-9). To get the odometry from the IMU, the team initially used the provided libraries recommended by the manufacturer [78]. These libraries solely use the IMU’s gyroscope. After extensive testing, the team determined the data obtained from this library was not accurate. As an alternative, the team implemented the Complementary filter. This section details the circuitry used by these methods and discusses their implementation.

#### Circuit

The IMU was wired into the Arduino to set up the I<sup>2</sup>C connection. The SDA pin on the IMU was connected to the SDA pin on our Arduino Mega (pin 20) and the SCL pin (pin 21) on the Arduino. The IMU circuit schematic is shown in Figure 7-17.

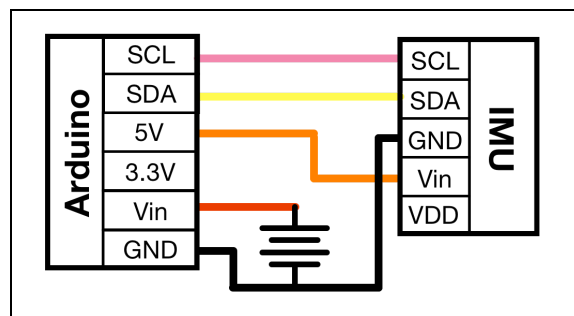


Figure 7-17 : IMU Circuit Schematic

#### Gyroscope

In order to calibrate the IMU to use the gyroscope, the team used the LSM303 library [7]. The library’s Calibrate.ino code (Appendix E) was used to obtain the Minimum and Maximum headings for the sensor. The Calibrate.ino script repeatedly polls, at 100ms, the compass for X, Y, Z values and replaces the globally defined vector values if they are ever less than the minimum or greater than the maximum. To cover all possible X, Y, Z header values, the IMU

was moved in all possible orientations to obtain the Maximum and Minimum values. The range of the calibrate poll data from the terminal is as shown in Figure 7-18. The maximum and minimum values obtained from this data are as shown in Table 7-6.

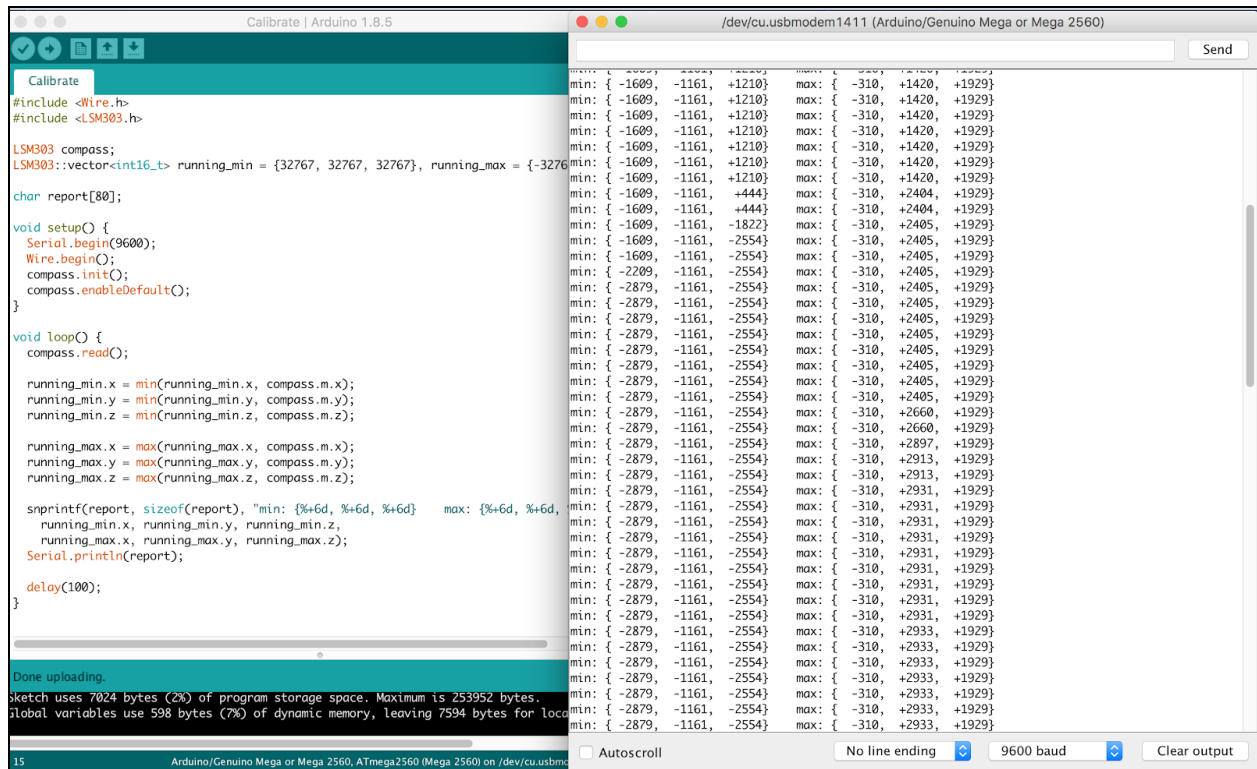


Figure 7-18: Calibration readings

Table 7-6 : Minimum and Maximum IMU sensor headings

Values	X	Y	Z
Minimum	-2879	-2435	-3591
Maximum	+2422	+2933	+2921

After calibrating the Pololu MinIMU-9, the roll, pitch, and yaw, were measured. To do this, the Gyro\_minimu.ino (Appendix E) was uploaded onto the Arduino. This script provided the X, Y, and Z values that corresponded to the orientation of the IMU. An example of the output from the script is shown in Figure 7-19.

```

/dev/cu.usbmodem1411 (Arduino/Genuino Mega or Mega 2560)
Send
G X: -12.35 Y: -111.13 Z: -27.74
G X: -7.94 Y: -107.71 Z: -48.78
G X: -23.54 Y: -113.40 Z: -40.45
G X: -30.03 Y: -117.75 Z: -25.97
G X: -12.15 Y: -109.56 Z: -30.77
G X: 63.03 Y: -97.19 Z: -13.85
G X: 67.23 Y: -98.60 Z: -13.82
G X: 132.00 Y: -103.47 Z: -13.35
G X: 134.84 Y: -101.48 Z: -13.38
G X: -39.71 Y: -99.79 Z: -2.89
G X: -74.16 Y: -94.53 Z: -4.67
G X: -41.23 Y: -96.85 Z: -6.79
G X: -36.29 Y: -98.47 Z: 2.38
G X: -27.96 Y: -102.50 Z: -6.56
G X: -26.97 Y: -104.08 Z: -6.10
G X: -27.11 Y: -102.39 Z: -6.69
G X: -23.98 Y: -48.79 Z: -5.30
G X: -26.70 Y: -15.54 Z: -7.23
G X: -27.73 Y: -11.15 Z: -9.22
G X: -30.61 Y: -22.93 Z: -12.88
G X: -31.63 Y: -151.78 Z: -12.91
G X: -34.11 Y: -187.09 Z: -12.18
G X: -28.45 Y: -172.36 Z: -12.41
G X: -27.49 Y: -104.47 Z: -30.34
G X: -28.01 Y: -101.54 Z: -32.25
G X: -23.76 Y: -101.75 Z: -0.42
G X: -19.72 Y: -78.92 Z: 72.81
G X: -28.66 Y: -76.21 Z: 77.34
G X: -28.24 Y: -73.47 Z: 77.12
G X: -30.48 Y: -76.27 Z: 43.96
G X: -3.01 Y: -113.32 Z: -84.57
G X: -12.53 Y: -118.40 Z: -127.15
G X: -10.95 Y: -118.21 Z: -127.57
G X: -17.85 Y: -101.62 Z: -105.52
G X: 19.82 Y: -54.80 Z: -68.10
G X: 21.73 Y: -44.65 Z: -65.86
G X: 23.17 Y: -42.08 Z: -65.61
G X: 25.51 Y: -39.96 Z: -65.01
G X: 26.42 Y: -38.88 Z: -65.50
G X: 27.23 Y: -38.11 Z: -66.17
G X: 27.96 Y: -37.46 Z: -66.88
G X: 28.62 Y: -36.97 Z: -67.73
Autoscroll No line ending 9600 baud Clear output

```

Figure 7-19: Resulting output from Gyro\_minimu.ino

Next, the accuracy of the roll, pitch were tested. The team performed various tests at different orientations about the X, Y and Z axis. The experiment was performed by first setting a base orientation and then seeing if a 90 degree rotation along an axis resulted in the gyroscope output to correctly shift by 90 degrees on the appropriate axis. The base case used in the experiment was the home/nose up position of the IMU that it was calibrated with. All orientations were referenced with respect to tise base case coordinates. The base orientation and global coordinate axis is as shown in Figure 7-20.

**Base case : In resting position (Nose up)**

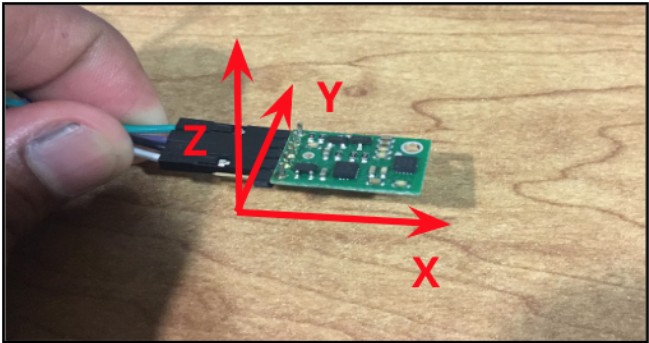
	<p>OUTPUT =</p> <p>G X: 376.25 Y: -540.24 Z: -173.43</p>
---	--

Figure 7-20: Base case, in resting position (Nose up)

The first orientation, as shown in Figure 7-21, was achieved by rotating about the X axis counterclockwise by 90 degrees. The theoretical output should have only changed in X axis by

-90 degrees. However, the actual outputted value along the X axis changed by -101.82 degrees (469.07 degrees - 367.25 degrees), as shown in Table 7-7. The 2.6 percent error was considered to be significant. But the team, continued with the experiment and performed a second test.

**Test 1 : Rotate X axis counterclockwise by 90**

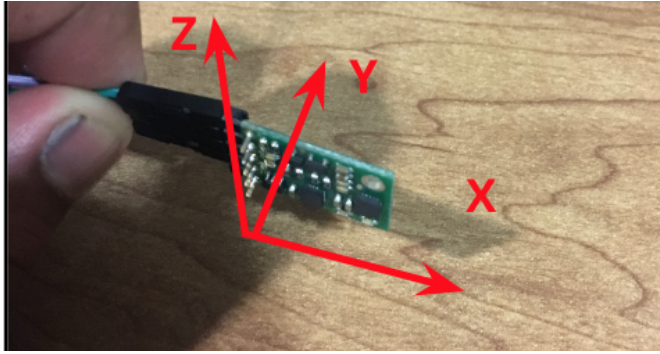
	<p>OUTPUT =</p> <p>G X: 469.07 Y: -561.84 Z: -176.36</p>
---	--

Figure 7-21: Test 1, rotate X axis counterclockwise by 90

Table 7-7: Difference between the Calculated and Actual readings from test 1

*All values are in degrees.	Base Orientation	Desired Rotation	Theoretical Result	Actual Result	Error (%)
X	376.25	90	457.25	469.07	2.585
Y	-540.24	0	-540.24	-561.84	3.998
Z	-173.43	0	-173.43	-176.36	1.747

For the second test, the IMU was rotated clockwise about the X axis and then again rotated clockwise about the Z axis. Next, the base position (Figure 7-20) and the Test 2 position (Figure 7-22) were compared and the expected output was then calculated. The output from the gyroscope for this new orientation in Test 2 was then compared to the calculated readings to calculate error and overall accuracy of the gyroscope component. The results of the error calculations are tabulated in Table 7-8.

## Test 2 : Rotate X Clockwise by 90 and rotate Z Clockwise by 90

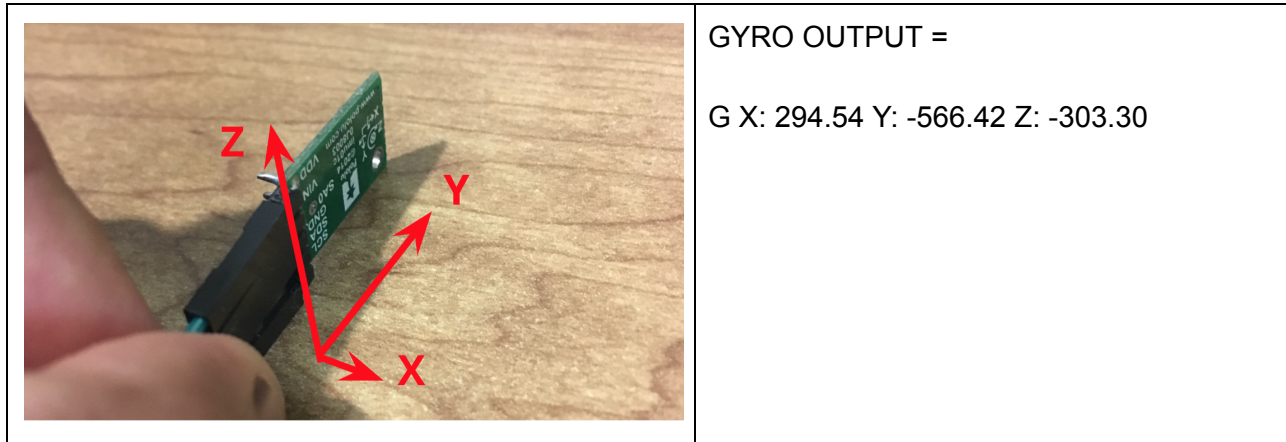


Figure 7-22: Test 2, rotate X Clockwise by 90 and rotate Z Clockwise by 90

Table 7-8: Difference between the Calculated and Actual readings from test 2

*All values are in degrees.	Base Orientation	Desired Rotation	Theoretical Result	Actual Result	Error (%)
X	376.25	-90	286.25	294.54	2.896
Y	-540.24	-0	-540.24	-566.42	4.848
Z	-173.43	-90	-263.43	-303.30	15.135

As with test one, there was a significant amount of error in the data. The team hypothesized that the error was caused by only taking advantage of the IMU's gyroscope, while neglecting the accelerometer and magnetometer. This led to the team to move to the complementary filter, which makes use of the gyroscope and the accelerometer.

### Complementary Filter

The complementary filter [79] is a sophisticated filter that measures both the long term and short term scenarios. The short time measurements taken by the filter, depend heavily on the data from the gyroscope. This is because the gyroscope is very precise for short term measurements and is not affected by outside forces. For the long term measurements, data from the accelerometer is preferred as it does not drift. The complementary filter in its simplest form is,

$$\text{Angle} = 0.98 * (\text{angle} + \text{gyroscope\_data} * dt) + 0.02 * (\text{accelerometer\_data})$$

Where, the gyroscope data is integrated every timestep with the current angle value and then, it is combined with the low-pass data obtained from the IMU's accelerometer. The

constants (0.98 and 0.02) have to add up to 1 but can of course be changed to tune the filter properly. Every iteration the pitch and roll angle values are updated with the new gyroscope values (gyroscope\_data) by means of integration over time. The filter then checks if the magnitude of the force seen by the accelerometer has a reasonable value that could be the real g-force vector. If the value is too small or too big, we know for sure that it is a disturbance we don't need to take into account. Afterwards, it will update the pitch and roll angles with the accelerometer data (accelerometer\_data) by taking 98% of the current value, and adding 2% of the angle calculated by the accelerometer. This will ensure that the measurement won't drift, but that it will be very accurate on the short term [77].

In order to implement the complementary filter, the compliment.ino library [Appendix E] was uploaded onto the Arduino and the gain values were modified with respect to X, Y, and Z values. The readings obtained from the complementary filter are shown in Figure 7-23.

```

/dev/cu.usbmodem1411 (Arduino/Genuino Mega or Mega 2560)
min: { -starting calibration
-3743.81
820.54
94.11
Accel Device ID3
GX: 6.36 GY: -0.15 GZ: -0.88 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 45.07 GY: -10.41 GZ: -127.76 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 66.86 GY: -9.43 GZ: -157.11 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 91.47 GY: -17.77 GZ: -162.44 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 112.34 GY: -17.78 GZ: -161.83 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 148.85 GY: -19.52 GZ: -170.49 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 175.46 GY: -20.26 GZ: -172.92 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 205.12 GY: -20.16 GZ: -176.71 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 234.67 GY: -20.23 GZ: -180.08 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 264.58 GY: -20.11 GZ: -182.83 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 293.64 GY: -20.15 GZ: -185.79 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 323.18 GY: -20.23 GZ: -188.84 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 351.12 GY: -20.07 GZ: -191.17 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 348.99 GY: 61.00 GZ: -190.98 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 384.08 GY: 60.79 GZ: -193.60 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 410.13 GY: 34.89 GZ: -212.10 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 436.73 GY: 39.36 GZ: -210.82 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 458.03 GY: 47.46 GZ: -187.71 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 490.79 GY: 18.61 GZ: -188.96 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 636.67 GY: 65.64 GZ: -205.26 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 755.10 GY: 27.50 GZ: -244.53 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 760.01 GY: 22.20 GZ: -245.79 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 647.19 GY: 28.17 GZ: -171.36 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 593.89 GY: 37.88 GZ: -172.99 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 592.87 GY: 4.26 GZ: -189.15 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 622.23 GY: 3.16 GZ: -192.49 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 723.95 GY: 35.01 GZ: -203.38 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 776.57 GY: 49.30 GZ: -194.21 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 809.98 GY: 51.25 GZ: -197.67 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 840.49 GY: 51.01 GZ: -200.82 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 870.49 GY: 50.79 GZ: -203.90 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 900.80 GY: 51.00 GZ: -207.19 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 932.27 GY: 50.08 GZ: -211.08 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 962.03 GY: 50.15 GZ: -214.15 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 991.64 GY: 49.92 GZ: -217.17 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 1021.33 GY: 50.82 GZ: -220.33 Ax = 0.00 Ay = 0.00 Az = 0.00
GX: 1050.82 GY: 50.64 GZ: -223.35 Ax = 0.00 Ay = 0.00 Az = 0.00
  
```

Figure 7-23: Terminal Showing Readings of Complementary Filter

After completing the setup of the complementary filter, the test performed in the gyroscope section were repeated.

### Test 1: Rotate X axis counterclockwise by 90 using the Complementary Filter

In order to analyse the efficiency of the complementary filter, it was put under a similar test as that of using a gyroscope. The IMU was rotated counterclockwise about the X axis and the X, Y, and Z output was recorded in Table 7-7, in which, the complementary filter can be seen



in comparison to the previously obtained readings using only the gyroscope, in Table 7-9. From Table 7-9 and Figure 7-24 below, we can see that errors calculated using a complementary filter are smaller than those calculated using the gyro. The errors are graphically displayed in Figure 7-24.

Table 7-9: Complementary filter measurements vs Gyroscope measurements in test 1 (in degrees)

All values are in degrees.	Calculated (GYRO)	Actual (GYRO)	Error (GYRO)	Calculated (COMP FILTER)	Actual (COMP FILTER)	Error (COMP FILTER)
X	457.25	469.07	11.82	450.00	445.67	4.33
Y	-540.24	-561.84	21.6	-540.00	-545.72	5.72
Z	-173.43	-176.36	2.93	-180.00	-179.23	0.77

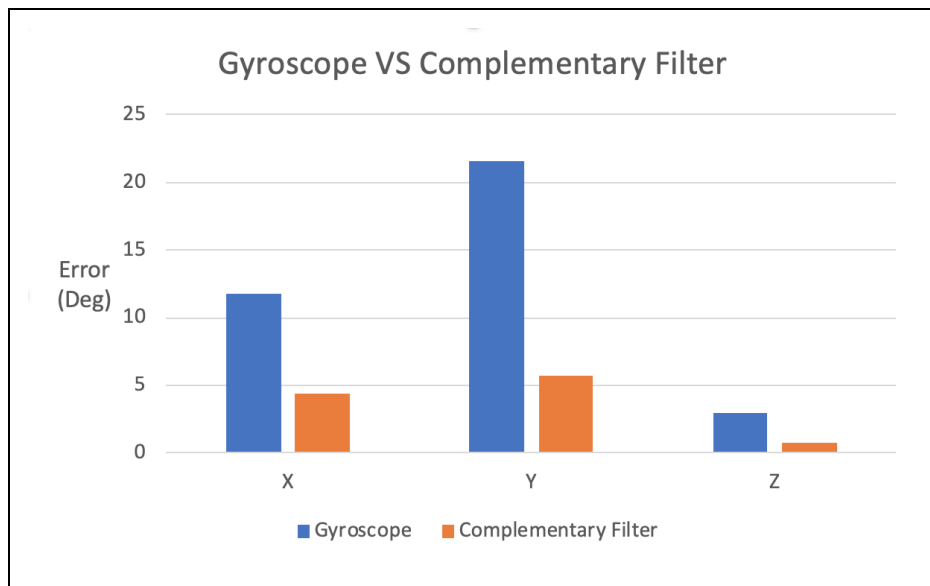


Figure 7-24: Comparing errors in Gyro vs Complementary filter for test 1

### Test 2 : Rotate X and Z Clockwise by 90 using the Complementary Filter

For test 2, the IMU is rotated clockwise by 90 degrees on the X axis and then rotated clockwise again by 90 degrees on the Z axis. The readings from using the complementary filter in comparison to the readings obtained from only using the gyroscope are tabulated in Table 7-10. Additionally, Table 7-10 shows the errors calculated using a complementary filter. Again, the errors produced by the complementary filter are smaller than the errors produced by the gyroscope. The errors are also graphically displayed in Figure 7-25.

Table 7-10: Complementary filter vs Gyroscope readings obtained in test 2

All values are in degrees.	Calculated (GYRO)	Actual (GYRO)	Error (GYRO)	Calculated (COMP FILTER)	Actual (COMP FILTER)	Error (COMP FILTER)
X	286.25	294.54	8.29	270.00	265.27	4.73
Y	-540.24	-566.42	26.18	-540.00	-532.71	7.29
Z	-263.43	-303.30	39.87	-270.00	-278.53	8.53

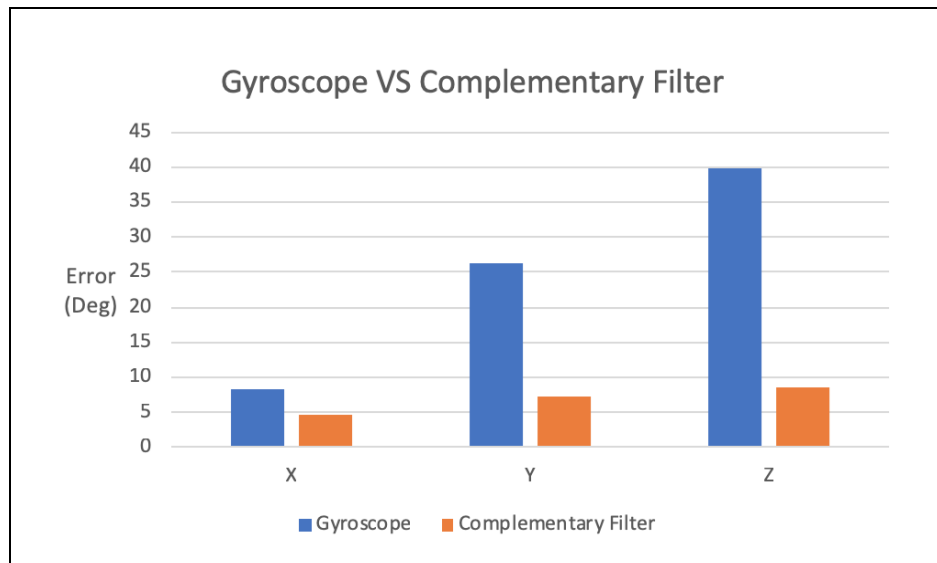


Figure 7-25: Comparing errors in Gyro vs Complementary filter for test 2

From the comparison of the gyroscope and the complementary filter, the team determined that the data obtained from the complementary filter was more accurate. Note that the Arduino scripts are in the Appendix E.

### 7.2.3 Tachometer

Using the KY-003 Hall-Effect sensor and neodymium magnets, a tachometer was created. Connected to pin 2 of the Arduino Mega, the hall-effect sensor would detect every instance where the magnet crossed its path. Following the code seen in Appendix G, the number of magnetic detections were counted per unit of time. Ultimately, this ratio, when appropriately converted, produced the revolutions per minute (RPM) of the drive wheels. While placement of the sensor and magnet required some trial and error, the sensor and magnet were ultimately integrated into the car as seen in Figure 7-26. Specifically, the Hall-Effect sensor was glued to the external siding of the differential housing. Furthermore, a stack of three neodymium magnets were glued to one of the adaptors to the rear axle. By strategically placing the sensor, the rotation of the magnets nearly touched the sensor per every revolution. Ultimately, this allowed for very responsive and accurate magnet detections.

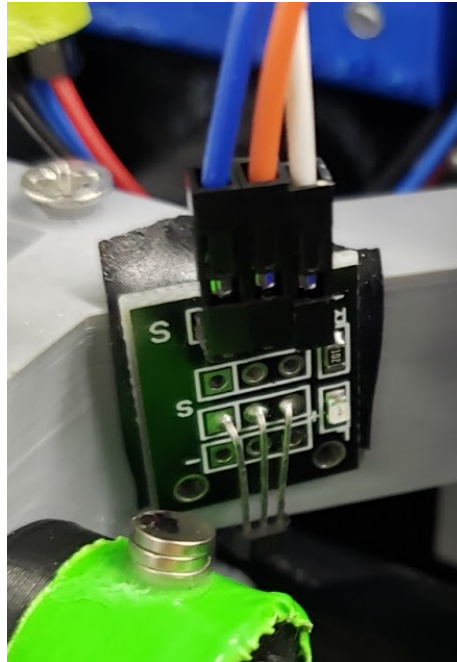


Figure 7-26: Hall Effect Sensor

### 7.3 Modularization of the Sensor System

After the team had implemented and selected sensors for specific sensor components, the next goal was to combine the sensors into a scalable and customizable system by making a PCB that uses all the sensors. Creating a PCB makes the sensor system scalable because once a PCB has been successfully designed, it can be replicated indefinitely. Creating a PCB also makes the sensor system customizable because not all the pins on the PCB need to be used.

To make the PCB, the team started by combining the individual circuit schematics described in previous section. Figure 7-27 shows the flow of the IMU, thermal system, and tachometer sensory information into the Arduino and Figure 7-28 shows the combined sensor circuit schematics.

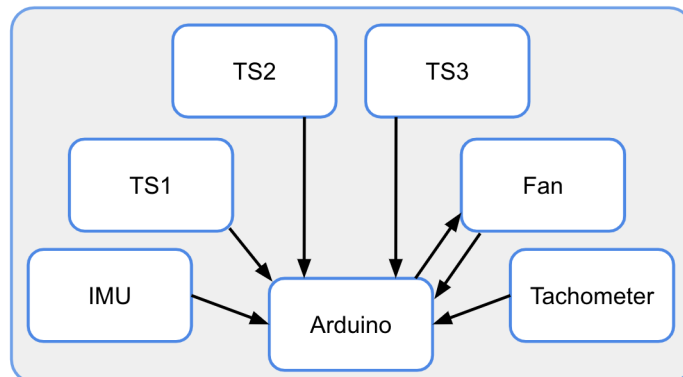


Figure 7-27: Flowchart of Sensor Dashboard Data

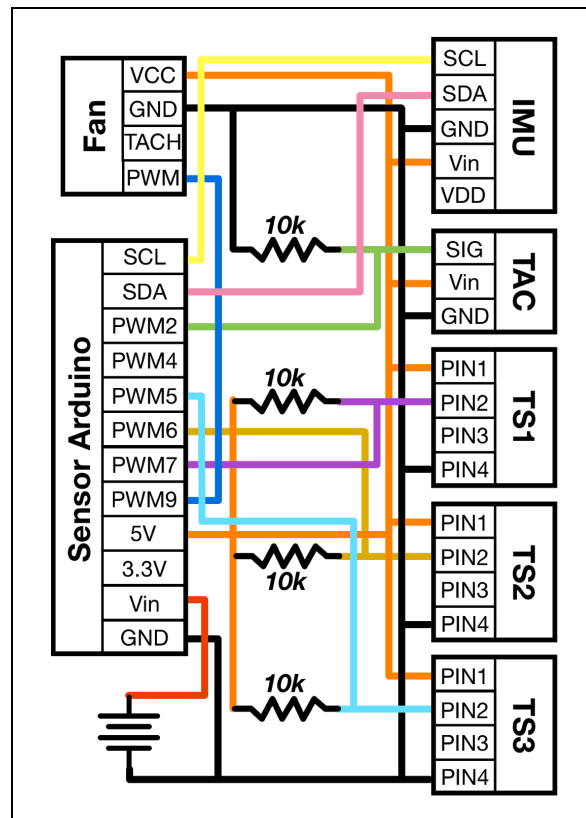


Figure 7-28: Combined sensor circuit schematics

After team had created the schematic of the combined circuit schematic and knew which pins that would be used by the devices, the Arduino scripts from each sensor were combined into a single Arduino script. Appendix G contains the resulting Arduino script. Once the sensors had been consolidated into one design, the team began designing a PCB based on the schematic in Figure 7-28. The PCB was designed using Altium 18.4 software [80] and was custom ordered from an external source, Oshpark LLC [81]. To design the PCB in Altium 18.4, a new PCB project was created in which, a PCB schematics library, a PCB Library, new schematics and new PCB were created. Initially, individual components for each sensor connector on the PCB were designed in the schematics library. The three temperature sensors components from the schematics library are shown in Figure 7-29. After adding all the individual components in the schematics library, they were included in the main PCB schematic and connections to the power and ground were added. Figure 7-30 shows the complete PCB schematics sketch for the sensor electronics PCB.

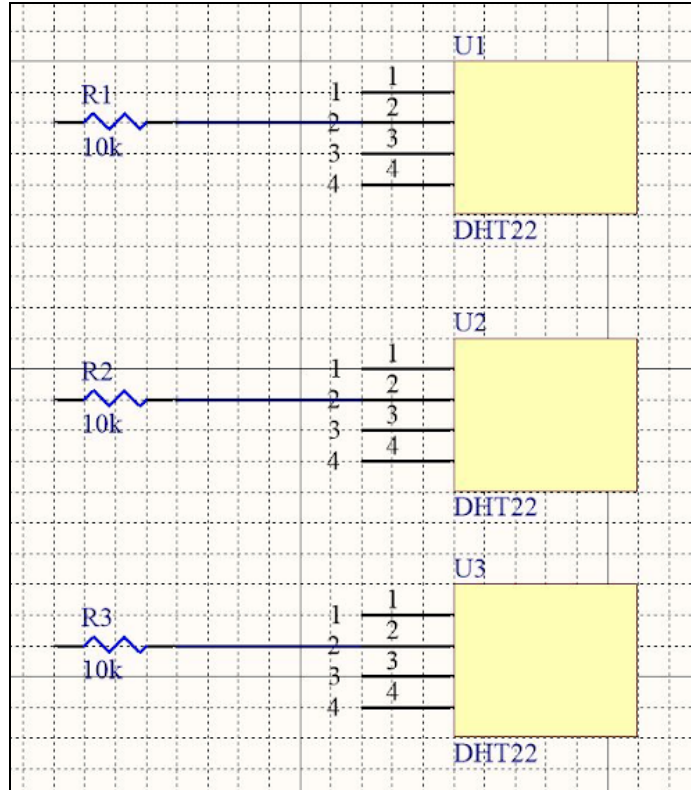


Figure 7-29: Temperature sensor components in Altium schematics Library

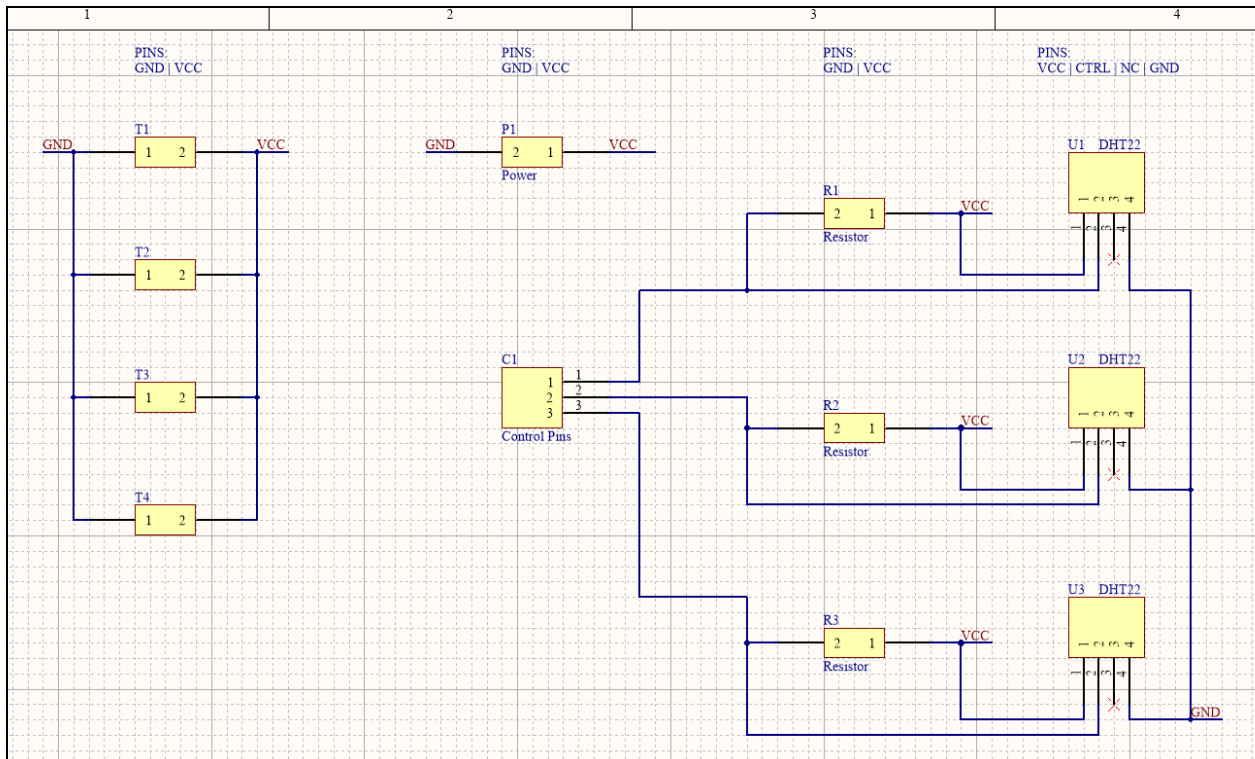


Figure 7-30: Final PCB schematics in Altium

Next, the PCB components, such as headers, were added to the PCB library and placed on the PCB layers. To design this custom PCB, a top and bottom layer were only required as most of the sensors were present externally and not required to be on the PCB itself. This was mainly because sensors such as temperature sensors are to be placed in proximity of heat sensitive units of a system for protection and not on the PCB itself. Additionally, the IMU is also preferred to be placed at the center of mass of a system for accurate readings and will therefore not be placed on the PCB either. Thus, the PCB was designed to only account for any external circuit required for the operation of a sensor and not account for the actual sensor installation. Figure 7-31 shows the top layer design of the custom PCB with DHT22 sensor components and Figure 7-32 shows the complete PCB design of the top and bottom layer in 2-D and 3-D view. The red wiring on the PCB, as shown in Figure 7-32, illustrates the top layer of the PCB whereas, the blue wiring illustrate the bottom layer.

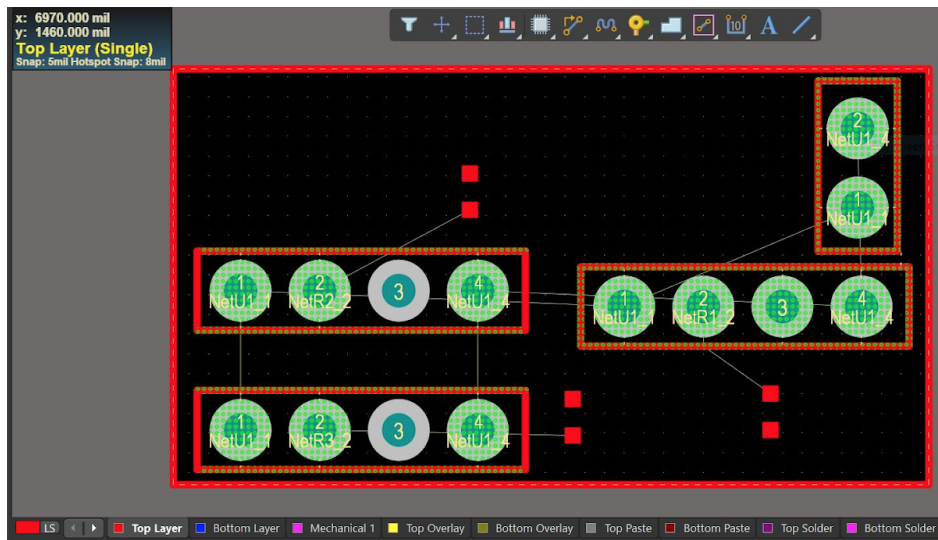


Figure 7-31: DHT22 sensor components on top layer in PCB design

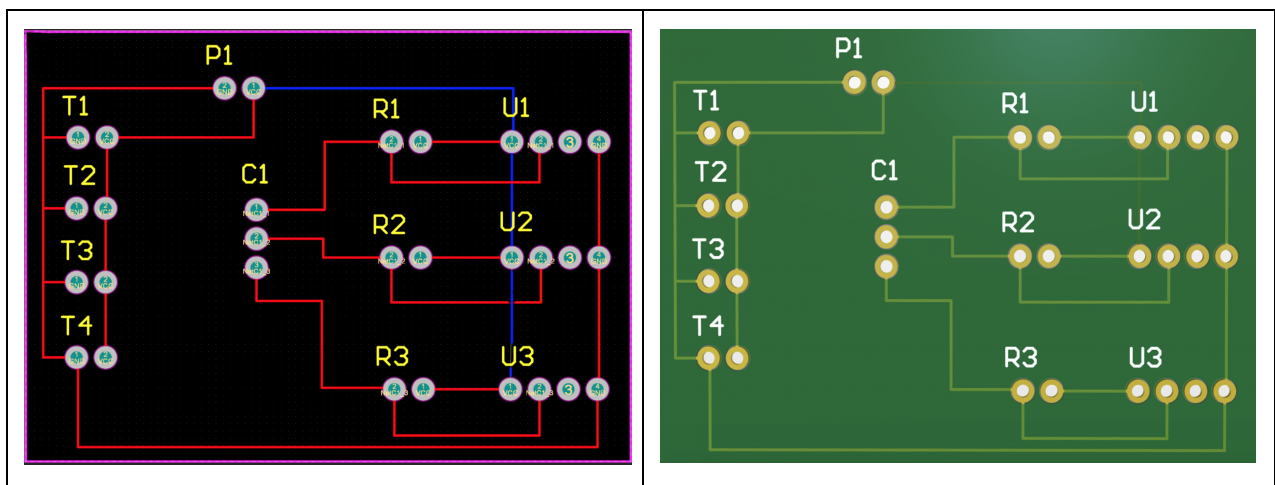


Figure 7-32: 2-D design, 3-D view of final PCB design

Lastly, Gerber files were generated and an order was placed to OshPark LLC for three custom PCBs for \$16. The delivered PCB was immediately ready to be integrated into the sensor system. Figure 7-33 shows the final custom PCB product. Due to time constraints, however, the PCB could not be integrated into the sensor system to replace breadboard circuitry. This has now become a future goal for the next team that will work on the sensor system.

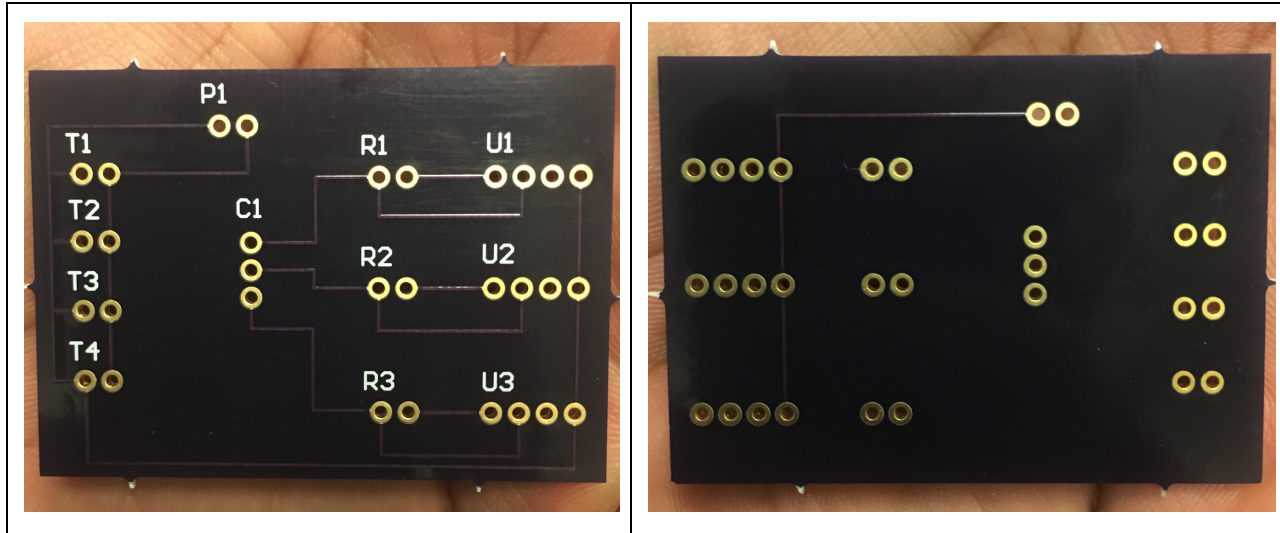


Figure 7-33: Final Printed Circuit Board for Sensor Dashboard

## 7.4 Data Collection

Although the team did not have sufficient time to implement the PCB described in the previous section, it was able to use a breadboard implementation of the circuit. Using this circuit, the team was able to obtain real-time data from each sensor on the custom car. Graphs of each of the data collected with the car running can be found below in Figures 7-34, 7-35, and 7-36.

Figure 7-34 illustrates the estimated RPM of the rear axle while the car is driving. While the test was conducted briefly, the spikes in the graphic illustrate moments of acceleration where the rear axle was increasingly torqued. Figure 7-35 illustrates the readings produced from the three temperature sensors while the car was driving. As expected, the temperature sensors located near the Raspberry Pis obtained the highest temperature readings. Overall, the nonvolatile and low magnitude temperature profile of the car gave little reason for thermal concerns. Finally, Figure 7-36 shows a sample reading from the inertial measurement unit. As the car was driven in circles during this data collection period, the primary fluctuations captured by the IMU related to the car's yaw orientation. While the car was driven on a flat surface, both roll and pitch had some inconsistency with their readings. Due to some body roll of the car's suspension and gyroscopic drift, however, said inconsistencies were marginal.

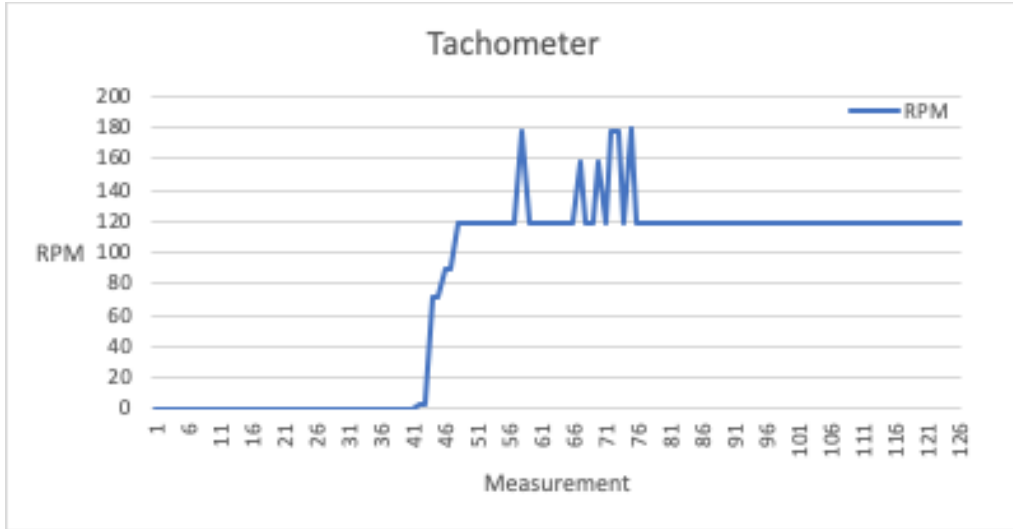


Figure 7-34: Data collected by the tachometer

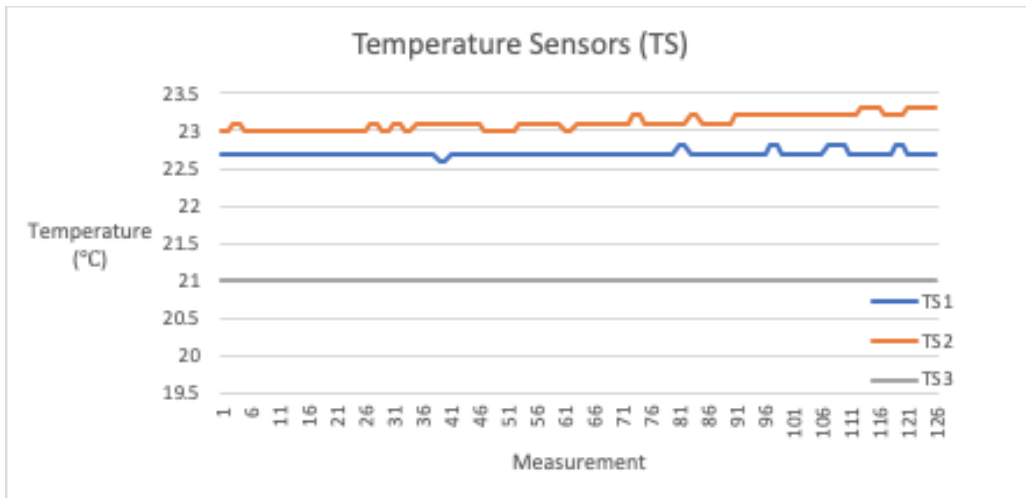


Figure 7-35: Data collected by Temperature sensors 1, 2 and 3



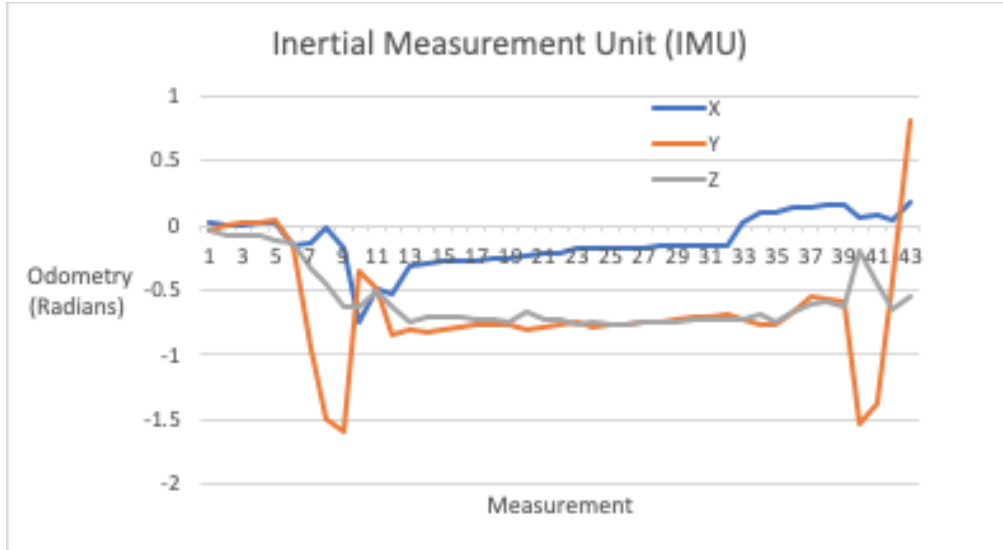


Figure 7-36: Data collected by the IMU

## 7.5 Sensor Dashboard

An important aspect of placing sensors on the car is to be able to view information in as readable and fast a way as possible. The goal of the modular sensor package was to bundle all of the information on a real time dashboard viewable on a webpage.

### 7.5.1 Display Server

In order to achieve this, the team connected the Arduino that was directly interfacing with the sensors to a Raspberry Pi. That Raspberry Pi captured the data sent by the Arduino over a serial connection and displayed that data on a simple web server.

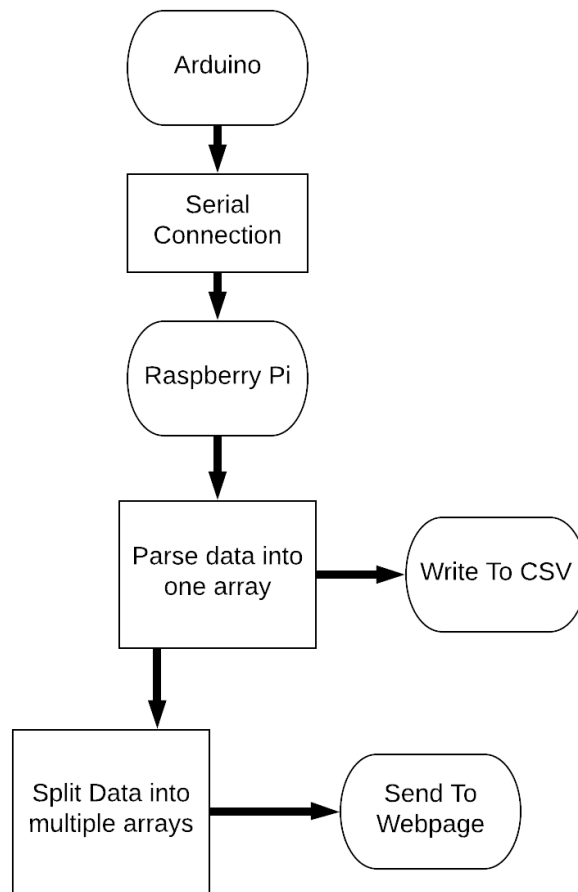


Figure 7-37: Data flow from Arduino to CSV and Webpage

As seen in Figure 7-37, data from the Arduino was sent over a serial connection as a string of comma separated values. That data was read in on the Raspberry Pi and parsed to be sent to the website. Since the data received by the Raspberry Pi was a string of comma separated values, the order of those values is constant. The received string was parsed from strings to a list of integers so it could be manipulated more effectively. Each sensor was separated into its own array of data and every time new data was sent to the Raspberry Pi, the most recent seventy five data points were sent through a socket to the website where each graph was updated. While the data is being sent to the website it is also being captured in a comma separated value file in order to allow for more intense analysis of the data over longer periods of time by the users of the sensor system. This system was designed to be extremely flexible with the only limit being the number of sensors that could be physically attached to the pins of the Arduino.

## 7.5.2 Graphics

The goal of the live data sensor module was to view data in real time. The webpage served by the Raspberry Pi is capable of displaying an unlimited amount of graphs. Some theoretical graphs are shown below. The three graphs included below include a sample temperature graph to show any ambient heat zones on the car, a sample graph from a tachometer to show the rotational speed of the shafts of the car, and finally a graph showing sample inertial measurement displaying the odometry of the car.

The collected data can be used to verify calculations and infer any issues with the car. For example, if the calculated rotational speed of a shaft is 180rpm at top speed, the users of the dashboard would be able to look at the real time data of the tachometer to either verify the car was operating inside normal operating parameters or if there was a malfunction causing the shaft to spin too quickly or too slowly.

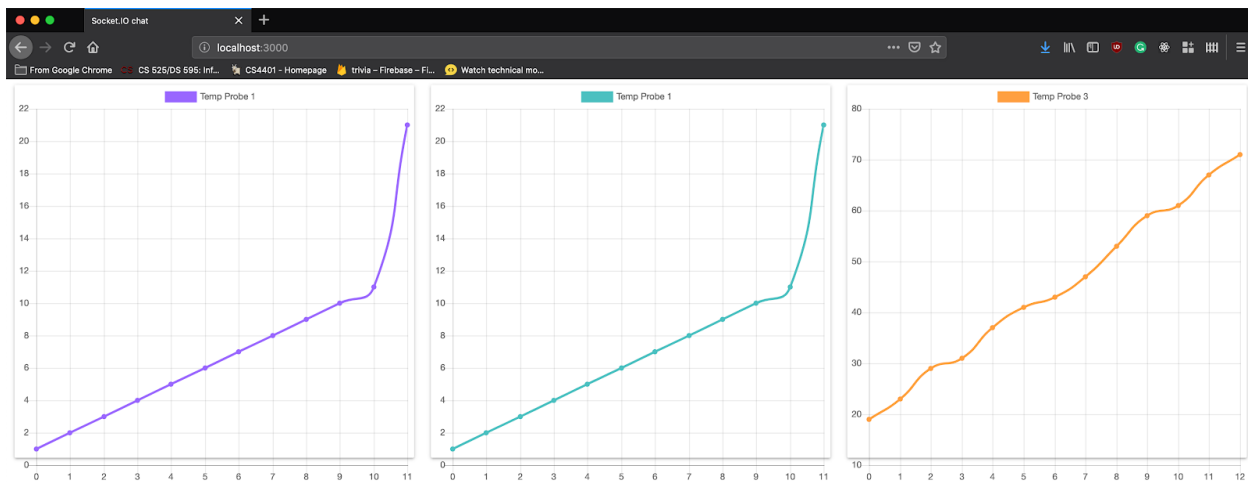


Figure 7-38: Mockup displays of sensor dashboard data using HTML/CSS/Javascript

In Figure 7-38, we have provided a mockup of what the dashboard could look like. Each graph is placed in its own card on the website to allow for simple dynamic repositioning. Each graph can be customized to contain as many data sets as desired. For example all of the temperature datasets could be placed in series on one chart to view all of the heat zones together. Similarly since these graphs can be generated using any visualization library the possibilities for what the data looks like on the website is endless.

## 7.6 Conclusion

In conclusion, the team made a modular sensor suite that could be utilized in WPI courses, specifically the Advance Engineering Design course (ME4320). This was done through the creation of a sensor dashboard that tracks a car's odometry, speed, internal temperatures, and axle rotation. The system was created by incorporating sensors such as an inertial measurement unit (IMU), multiple temperature sensors, and a tachometer. The sensor system

was made to be modular, scaleable, and customizable by integrating the circuits onto a PCB. Lastly, the data collected from the sensory components of the system was parsed and saved as a Comma Separated Values file (CSV file) and displayed in real-time on a webpage.

Future iterations of the project should continue with the implementation of a real-time display. Additionally, more sensors should be added to increase the sensor dashboard's capabilities. More specifically, strain gauges should be added to help perform analysis on mechanical systems. This chapter discussed the modular sensor system designed by this team, the next chapter will discuss the autonomous driving module and the methodology behind its design.

## 8.0 Autonomous Driving

This chapter will discuss the methodology behind the implementation of the autonomous driving module for the team's car. Specifically discussed is implementation of the AI, component selection, data collection, data formatting, data parsing, and the issuing of instructions back to the car itself.

### 8.1 Implementation

Figure 8-1 depicts the general flow of data when the car is operating under the control of the autonomous driving module. Data is collected from the camera and sent to the Raspberry Pi. That image is sent through an OpenCV masking function that reduces the dimensionality of the image and increases its computational effectiveness. The masked image is sent through a neural network running on the Raspberry Pi which outputs an instruction command. That command is sent to the Arduino connected to the Raspberry Pi which delegates the instructions to the relevant mechanical systems.

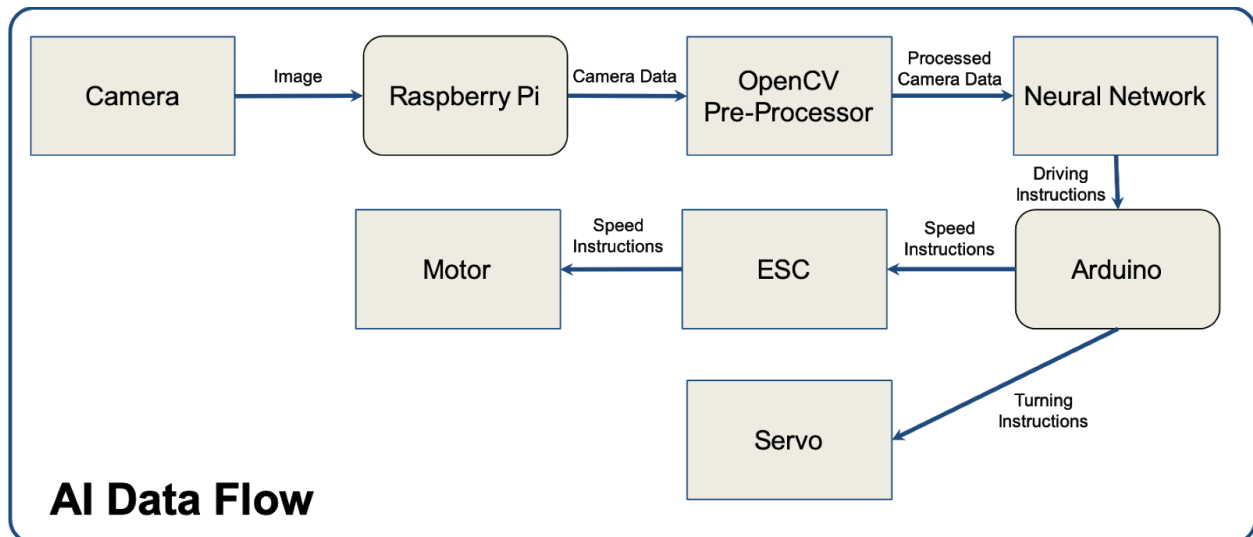


Figure 8-1: AI Data Flow

### 8.2 Collecting Data

When creating an artificial intelligence to perform a given task, you need to have a way to teach it. In this project, we taught our AI using supervised learning. Supervised learning is a method of creating an AI that utilizes labeled training data. Labeled training data is any data with corresponding ground truth values. In our case, we recorded data by driving our car around a race track. The video produced from the camera was the data, and the commands issued to the car from the controller were the labels (ground truths). Data was collected using the maximum resolution available from our camera, (640x480), and our controller inputs were recorded with up to 100 degrees of resolution.

### 8.2.1 Collection System

For the physical data collection, the team had to establish a system that read input data from the radio controller and match that to the recorded frames from the camera that gives the car its vision. The team used an Arduino and a Raspberry Pi to accomplish this task. First, the Arduino was connected to the radio controller in order to read the signal from the controller and pass that input along to the physical hardware on the vehicle. The Arduino took the data it was receiving and sent that through a serial connection to the Raspberry Pi.

The Raspberry Pi is responsible for receiving serial data from the Arduino and capturing that data to a text file. The Raspberry Pi collected and recorded the camera data to an avi file. The recording script on the Raspberry Pi is very straightforward, written below is pseudo-code for the process.

```
while(True):  
    frame = read_camera_data()  
    input = read_controller_data()  
  
    video_writer.write(frame)  
    file_writer.write(input)
```

The code on the Raspberry Pi starts by opening a serial connection to the Arduino. The code then loops indefinitely reading both controller data from the Arduino and sequential frames from the camera on the Raspberry Pi. Figure 8-2 illustrates a sample of the controller data written to the file with the format, Frame Number: Acceleration, Turning.

```
233: 111, 135
234: 111, 135
235: 111, 135
236: 111, 135
237: 111, 135
238: 111, 135
239: 111, 135
240: 111, 135
241: 111, 135
242: 111, 135
243: 111, 135
244: 111, 135
245: 111, 115
246: 111, 90
247: 111, 90
248: 111, 90
249: 111, 90
250: 111, 90
251: 111, 90
252: 111, 105
253: 111, 105
254: 111, 105
255: 111, 105
256: 111, 105
257: 111, 100
```

Figure 8-2: Sample of data written to file format FRAME\_NUM: ACCEL,TURN

### 8.3 Formatting Data

When giving training data to an AI, it is always important to attempt to make the data as basic as possible. If we input our raw recorded data into the AI for training, it might detect patterns that are less than ideal. For example, perhaps coincidentally there was a blue mark on a pillar in the background, that just happened to appear in view every time the car was to make a right U-turn. The AI may begin to associate blue marks on pillars with right U turns, when in reality that pillar and blue mark have nothing to do with why the car is turning right.

During our initial testing of the system, we used raw camera data and a camera placed on the front edge of the car, the resulting camera data was ultimately not useful for training for a number of reasons. Firstly, the image was mostly not relevant information meaning that the camera was capturing the view from directly in front of the car, only the bottom third of that image was useful, as seen in figure 8-3, since it contained the track.



Figure 8-3 Front positioned camera data

Without any preprocessing we tried to train an AI using that front positioned camera data. The resulting AI was unable to determine what information was relevant to driving as there was so much noise in the data. We realized that camera positioning was far more important than we first thought. Secondly, the quality of the incoming data was the next challenge the team had to address. When the camera was on the front of the car we were getting far more irrelevant data than relevant data. Our team changed the position of the camera to be elevated at the back of the car as seen in Figure 8-4, this increased the quality of the data we were receiving because it was only capturing relevant data around the car. The next step was filtering out noise from the image including the body of the car itself, the floor color variation, any information that was not a wall.

To solve this problem, it is essential that all non-important features are removed from the input data. The method used to remove these non-important features must be quick and efficient, since we will need to employ it in real time on the car while it is driving itself.

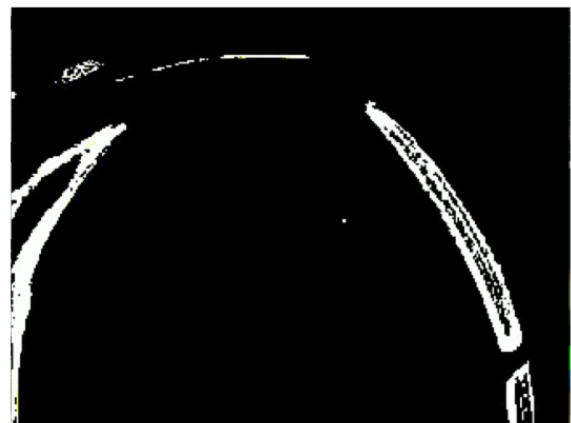
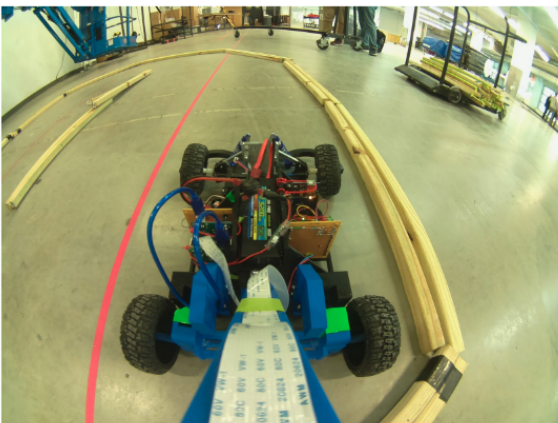


Figure 8-4: Raw versus Masked Image Data







Figure 8-6 Example rounded track

In Figures 8-5 and 8-6 there are two different track configurations that were tested. The goal of a track configuration was to have as many varieties of turns that a car could encounter on an unknown track. This included sharp and mild turns, when recording data the tracks were traversed in both directions to ensure that the training data could teach both left and right turns. Ultimately the car was trained more successfully on the switchback track than the rounded one.

## 8.4 Issuing Commands

Issuing commands to the car required a separate set of scripts than collecting data. The goal with issuing commands was to run the neural network on the Raspberry Pi and have that format commands over the same serial connection used for recording. The challenge was getting serial instructions to be issued correctly to the Arduino. Formatting them on the Raspberry Pi was simple, just creating a string, but getting the Arduino to interpret them correctly was a distinct challenge. The biggest issue involved with sending commands to the Arduino from the ANN was how the Arduino accepts information through its input buffer.

### 8.4.1 Arduino Serial Events

In order to successfully initialize a serial connection with an Arduino, it must send some sort of serial output to the monitoring computer. Our team chose to send a string in the required setup function of the Arduino. A print function is required in the setup of the Arduino code because that allows the monitoring computer to verify and establish a connection over the serial channel, the content of the printed string does not matter. The second step for issuing commands to the car is to ensure proper formatting of instructions. Since the serial channel on

the Arduino is a buffer communication is continuous and there needs to be a way to indicate that one instruction has ended. The instructions were sent in the format of a three digit number padded by zeros in the front if necessary, separated by a comma, and ended with a pipe character. This format was chosen specifically because it matches the type of data collected in the supervised learning as seen in Figure 8-2. The only difference between the collected data and the command instruction format is that commands are sent with a terminating | character. The pipe was chosen because it is a non-standard character that can be used to mark the separation of values in an input stream. In order to process data the group wrote a function that executes at the top of every call to the loop function. This function reads the incoming byte stream and as long as there are characters available to be read the function appends them to a string. As soon as the next incoming character is a “|” the function stops recording appending to the string and marks that input string as ready to be processed for execution. The input string is then processed and the resulting instruction is completed by the system.

#### **8.4.2 Neural Network Design**

Implementing a Neural Network requires a lot of testing. Training individual models took anywhere from 5 to 20 minutes based on the complexity of the model. Different factors to consider when constructing a model included: number of hidden layers, size of layers, whether or not to use fully connected layers or convolutional layers, activation functions, and many more. Below is a simple sample model:

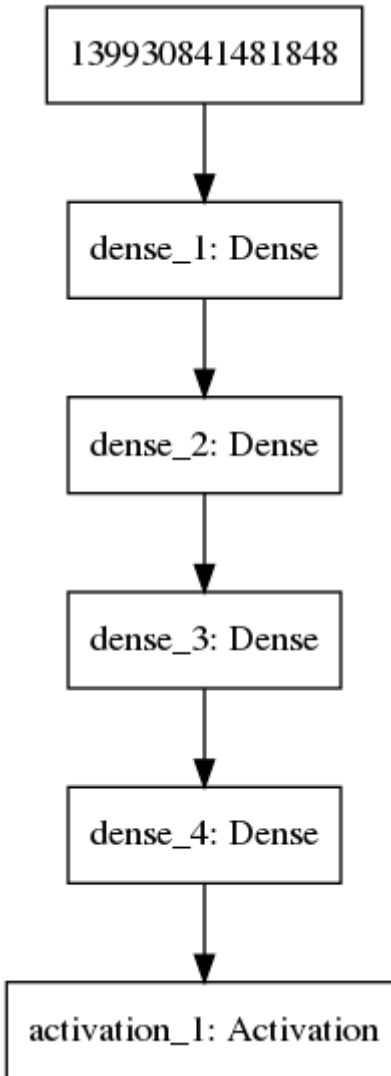


Figure 8-7: Simple Fully Connected Neural Network

In this model, we can see a simple fully connected neural network. We learned quickly that a model like this is not very effective. Each dense layer is fully connected to the subsequent dense layer, this can slow computation because the size of the network is dictated by the complexity of the network and the more connected nodes there are the larger the network. Dense layers by definition are fully connected, for our application there was no need for a fully connected network like this because our images did not have a lot of nuance. Below is a more nuanced model design that was heavily inspired by the EuroPilot open source self driving implementation:

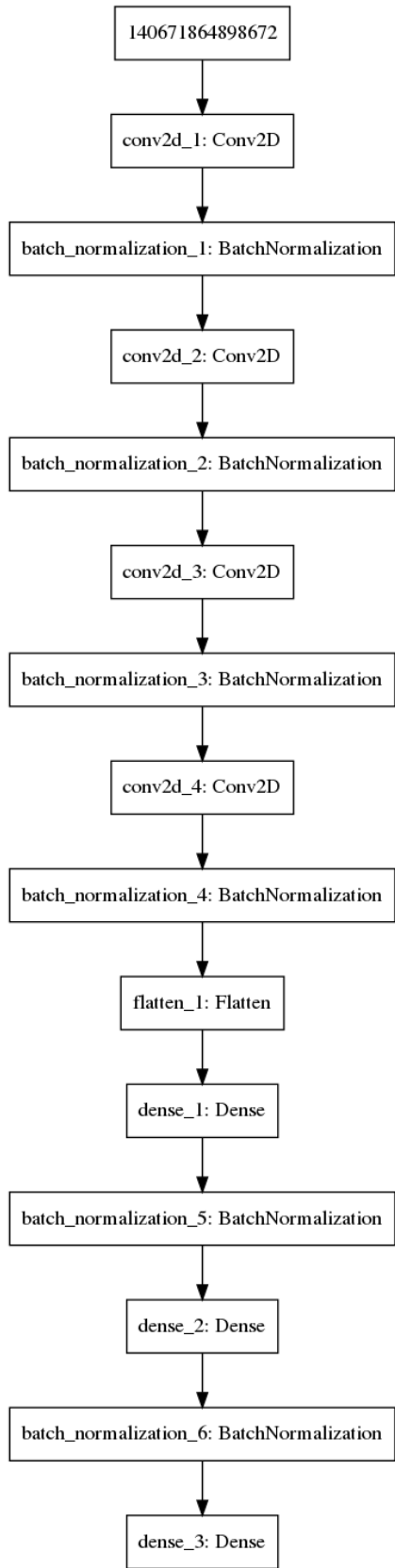


Figure 8-8 - Neural Network Model Inspired by EuroPilot [82]

This model utilizes batch normalization and convolutional layers, which are not fully connected. This network is far more effective at driving a car because the lack of full connection of nodes allows for more robust quick computations. Convolutional layers, Conv2D in the image, are layers that do not have to fully connect to the subsequent layer. This can increase computational effectiveness because they do not have fully connected paths that will never be activated. The batch normalization steps apply an algorithm that smooths images. The algorithm takes a pixel from the image and examines all of the surrounding pixels and sets that pixel to be the value most common value around it. This algorithm helps remove noise from image processing by ensuring that zones that are surrounded by white pixels, meaning walls, will be filled in to hopefully get stronger training classification.

## **8.5 Conclusion**

During this project we were able to prove that creating a cheap self driving solution using a neural network and a camera is completely feasible. We produced software to properly filter important features from our sensor, and integrated it with the neural network to drive the car autonomously. The only sensor required for our AI was a camera. We successfully integrated a data collection system onto an RC car that could control the car reasonably well. We were able to successfully send and receive communication from an Arduino through a serial channel. Overall we designed a modular autonomous driving system that can be integrated with any RC car platform. This chapter discussed the modular autonomous driving module designed by this team, the next chapter will discuss the conclusions and final thoughts from this project.

## 9.0 Conclusion

In summary, this report examined the feasibility of creating a 1:10 scale car with modular autonomy and sensory packages. Due to the iterative and interdisciplinary nature of the work, the project was broken down into 3 fundamental sections. First, the creation of a 1:10 scale car with independent front and rear suspension and corresponding analysis. Second, the creation of a modular sensor package which could collect performance and other car data. Third, the creation of a modular autonomy package leveraging a Raspberry Pi and single, front-facing camera.

### **Mechanical**

As a team we were able to design a full 1:10th scale RC car capable of driving reliably around a track. There were three main goals the car itself needed to accomplish. The car needed to have front and rear suspension, it needed to perform reliably, and it needed to be able to accommodate the sensor and autonomous packages. Ultimately, we were able to accomplish the first task by designing independent front and rear suspensions for the car. This task required us to stray from the standard designs of students in the course to design steering and transmission systems that can accommodate the different positions of the front and rear wheels. The second task was accomplished through a series of design alterations to improve car design and performance. Ultimately, our car was able to perform reliably through the long periods of driving required to collect data for AI training and the many crashes that resulted from testing different AI models. Third, we were able to accommodate the sensor and autonomous modules through the design of custom mounts.

In order to apply metrics to evaluate the reliability of the car, we performed analysis on the mechanical subsystems. By performing Finite Element Analysis, we were able to determine the weak spots on several of the components. By performing static analysis, we were able to determine the forces on each component, allowing us to better understand the loading on each component. By performing dynamic analysis we were able to determine how components would react to dynamic loading such as sudden starts and stops. All of these analyses were able to give us a broader sense of the strengths and weaknesses of the car.

### **Sensor Module**

For this project, we were able to successfully design and produce a modular and scalable sensor suite capable of collecting and displaying live car data. The groundwork provided for the creation of the sensor dashboard has been completed. The sensor dashboard that future teams implement could accurately display the car's orientation and speed and monitor its internal temperature. Tracking orientation and speed helps in supervising the car's behavior in various situations while regulating the heat flow is vital to protect various heat sensitive mechanical and electrical components.

While there was no time to leverage the data collected from the scale car, the data from the IMU could be used to enhance the autonomy of the vehicle. More specifically, the IMU data could potentially be able to correct the car's orientation whenever necessary and help sustain the desired direction of movement. This could also account for any mechanical play in the

system which might cause the car to deviate from the path determined by the artificial intelligence.

The modular sensor suites could also be utilized in other WPI courses which require collection of such data. Most specifically, the Advance Engineering Design course (ME4320) could potentially use these sensor suites to supervise scale car behaviors for performance analysis.

### **Artificial Intelligence**

During this project we were able to prove that creating a cheap self driving solution using a neural network and a camera is completely feasible. We produced software to properly filter important features from our sensor, and integrated it with the neural network to drive the car autonomously. The only sensor required for our AI was a camera. We are confident that with slightly more time investment, it will be possible to drastically improve the autonomous package and create a fully autonomous vehicle. In its current state, our neural network model is capable of driving the car through approximately 50% of an indoor wooden track.

### **Future Work**

For future work, we have organized our suggestions into three categories: AI, Mechanical, and Sensor Module. The suggestions are as follows:

#### **Mechanical**

- Mechanical design refinement
  - Improve front suspension
    - Better articulation
    - More responsive suspension
    - Softer springs
  - Improve Steering Linkage
    - Redesign to reduce toe out
    - Redesign to improve responsiveness
    - Redesign to better integrate with front suspension
  - Improve Rear Suspension
    - Better articulation
    - Softer springs
  - Transmission
    - Improve connection between differential and gearbox
    - Reduce friction in differential
    - Replace 3D printed axles
    - Move gearbox closer to front of car so that drive shaft can be closer
- Convert parts from 3d prints to machined versions
  - Replace parts that break or melt with more durable materials
- Mechanical Analysis
  - Synthesize new four bar for front and rear suspension



## **Sensor Module**

- Integrate the live dashboard
- Integrate the PCB
- Integrate strain gages on the car

## **AI**

- Create a more systematic training process for the AI
- Include statistical analysis to monitor, track, and evaluate AI metrics
- Train on more complex tracks with extremely varied design
- Train with other cars to learn car avoidance

## **Reflection**

Highly interdisciplinary in nature, this project called for the integration of mechanical engineering, electrical engineering, robotics engineering and computer science. For mechanical engineering, the construction of the car required immense knowledge of manufacturing, SolidWorks, and dynamical analysis. Highly iterative, the mechanical design and manufacturing of the car required tremendous trial and error and modification. Furthermore, while the manufacturing of the car could have been more systematic, the difficulty in design illustrated the gap that can form between theory and practice.

For electrical engineering, the work primarily revolved around controls and signals engineering. Focusing on timing, microcontrollers, and communication, the electrical engineering requirements of this project emphasized precision and speed. Among many lessons, one of the most important was the importance of synchronization between systems.

Integrating electrical and mechanical principles, robotics engineering describes the general nature of the whole project. As such, the robotics work was highly dependent on kinematics, controls, and sensor knowledge. The discipline that symbolized the interdisciplinary nature of the project, the robotics work was particularly helpful in showing the importance of sensors and their affiliation with the surrounding world.

Finally, for computer science, the work primarily revolved around timing, communication, and artificial intelligence. Producing both the control and autonomy systems of the car, all computer science work was pivotal towards achieving the project's fundamental goals. Among many lessons learned, one of the most important was the highly sensitive nature of neural networks. Highly dependent and sensitive to an abundance of variables, the creation of effective neural networks required iterative and systematic designs.

## **Societal Impacts**

As technology becomes increasingly digitized, data becomes more prevalent, hardware becomes more powerful, and artificial intelligence improves, the autonomous car will play an increasingly important role in society. Evidenced by the work of Tesla, a leader in car autonomy, Level 5 autonomy cars will arrive in the not too distant future. While the thrill and reward of autonomous navigation has sparked a nearly global competition among major companies, it has also invoked academic intrigue. It is for this reason that this report is written.

While the work of legal, full-sized autonomous vehicles is far beyond the scope of this project, this work illustrates many of the challenges associated with driverless technology. Avoiding the tremendous legal, financial, moral, and societal constraints of full-sized autonomous technology, this project provided an agnostic view on the pros and cons of autonomous navigation. Instead, this project hopes to both inspire and educate those who wish to pursue the academic challenges of driverless cars.

## References

- [1] OFNA .Com - OFNA Racing Radio Controlled R/C (RC) Products, [ofna.com/gtv2e-rtr.php](http://ofna.com/gtv2e-rtr.php).
- [2] "Raspberry Pi 3 Model B+." Raspberry PI Foundation, <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>.
- [3] Autopilot. (n.d.). Retrieved April 22, 2019, from <https://www.tesla.com/autopilot>
- [4] "Donkey Car." Donkey Car, [www.donkeycar.com/](http://www.donkeycar.com/).
- [5] Vasilash, Gary S. "Argo AI and Getting Ford to Level 4 Autonomy." Autoblog, Autoblog, 8 Sept. 2018, [www.autoblog.com/2018/09/08/argo-ai-ford-level-4-autonomy-self-driving-car/](http://www.autoblog.com/2018/09/08/argo-ai-ford-level-4-autonomy-self-driving-car/).
- [6] Berk, Christina Cheddar, and Berkeley Lovelace Jr. "General Motors Shares Soar as SoftBank Invests \$2.25 Billion in Automaker's Self-Driving Vehicles." CNBC, CNBC, 31 May 2018, [www.cnbc.com/2018/05/31/softbanks-vision-fund-to-invest-2-point-25-billion-in-general-motors-self-driving-vehicles-unit.html](http://www.cnbc.com/2018/05/31/softbanks-vision-fund-to-invest-2-point-25-billion-in-general-motors-self-driving-vehicles-unit.html).
- [7] Jonathan Dyble. "Understanding SAE Automated Driving – Levels 0 to 5 Explained." Cloud Computing | GigaBit, Ben Mouncer, 24 Apr. 2018, [www.gigabitmagazine.com/ai/understanding-sae-automated-driving-levels-0-5-explained](http://www.gigabitmagazine.com/ai/understanding-sae-automated-driving-levels-0-5-explained).
- [8] Walker, Jon. "The Self-Driving Car Timeline – Predictions from the Top 11 Global Automakers | Emerj - Artificial Intelligence Research and Insight." Emerj, Emerj, [emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/](http://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/).
- [9] Wang, R., & Crimmins, R. C. (2016). Autonomous Ground Vehicle Prototype via Steering-, Throttle-, and Brake-by Wire Modules. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/3183>
- [10] Tyson, Jeff. "How Radio Controlled Toys Work." HowStuffWorks, HowStuffWorks, 18 Dec. 2000, [electronics.howstuffworks.com/rc-toy.htm](http://electronics.howstuffworks.com/rc-toy.htm).
- [11] "Extreme Machines Ford Mustang Tri-Band 1:10 RTR RC Car." HobbyTron.com, [www.hobbytron.com/ExtremeMachinesFordMustangTriBand110RTRRCar.html?gclid=Cj0KCQiAmafBRDUARIsACOKERPYO\\_o-HJXWh6q\\_UNb9QtRXIXeTlclM2d5G8ObgS12ZQEh9JN3lp0aAr4KEALw\\_wcB](http://www.hobbytron.com/ExtremeMachinesFordMustangTriBand110RTRRCar.html?gclid=Cj0KCQiAmafBRDUARIsACOKERPYO_o-HJXWh6q_UNb9QtRXIXeTlclM2d5G8ObgS12ZQEh9JN3lp0aAr4KEALw_wcB).
- [12] "Hobby Motor - Medium Torque." Jaycar, [www.jaycar.com.au/hobby-motor-medium-torque/p/YM2707](http://www.jaycar.com.au/hobby-motor-medium-torque/p/YM2707).

[13] Woodford, Chris. "How Do Electric Motors Work?" Explain That Stuff, 4 Apr. 2018, [www.explainthatstuff.com/electricmotors.html](http://www.explainthatstuff.com/electricmotors.html).

[14] "Transmission Basics: How Does a Transmission Work?" Transmission Repair Guy, [transmissionrepairguy.com/how-does-a-transmission-work/](http://transmissionrepairguy.com/how-does-a-transmission-work/).

[15] "A Look at Belt, Chain and Gear Drive Technology," A Look at Belt, Chain and Gear Drive Technology | Power Transmission Blog. [Online]. Available: <https://www.powertransmission.com/blog/a-look-at-belt-chain-and-gear-drive-technology/>. [Accessed: 12-Oct-2018].

[16] "G2 Axle & Gear Driveshaft for 07-18 Jeep Wrangler JK." Mopar 52124170AA Wheel Lug Nut Wrench for 07-18 Jeep Wrangler JK | Quadratec, [www.quadratec.com/p/g2-axle-gear/driveshaft-07-16-jeep-wrangler-and-wrangler-unlimited-jk](http://www.quadratec.com/p/g2-axle-gear/driveshaft-07-16-jeep-wrangler-and-wrangler-unlimited-jk).

[17] "Drive Shaft." Internal Combustion Engine - Energy Education, [energyeducation.ca/encyclopedia/Drive\\_shaft](http://energyeducation.ca/encyclopedia/Drive_shaft).

[18] Nice, Karim. "How Differentials Work." HowStuffWorks, HowStuffWorks, 2 Aug. 2000, [auto.howstuffworks.com/differential2.htm](http://auto.howstuffworks.com/differential2.htm).

[19] GrabCAD: Design Community, CAD Library, 3D Printing Software, [grabcad.com/library/open-differential-mechanism-1](http://grabcad.com/library/open-differential-mechanism-1).

[20] "Complete Guide to Car Suspension," Carbibles, 17-Sep-2018. [Online]. Available: <https://www.carbibles.com/guide-to-car-suspension/>. [Accessed: 12-Oct-2018].

[21] "Coil Springs," MOOG Parts | Steering, Suspension & Drivetrain Parts. [Online]. Available: <https://www.moogparts.com/parts/suspension/coil-springs.html>. [Accessed: 12-Oct-2018].

[22] K. Neil, "Deaver 1-1.5' Lift 10 Leaf Springs (set) 2005-2015 Tacoma," BilsteinLifts.com, 03-Oct-2016. [Online]. Available: <https://bilsteinlifts.com/shop/toyota-tacoma/2005-2015/deaver-1-5-inch-lift-10-leaf-springs-set-for-2005-2015-toyota-tacoma/>. [Accessed: 12-Oct-2018].

[23] S. Connection, "Double Convolute Air Spring - Ride Height 5.5'-7.0' / Max. Diameter 7.52' / 1/4 NPT Air Inlet - EACH," Firestone 6859 - Double Convolute Air Spring - Ride Height 5.5"-7.0" / Max. Diameter 7.52" / 1/4 NPT Air Inlet - EACH | SuspensionConnection.com. [Online]. Available: <https://www.suspensionconnection.com/6859-air-bags.html>. [Accessed: 12-Oct-2018].

[24] "0.96' Torsion Bar Deluxe Kit - Rubber - B & E Body," Performance Suspension Technology. [Online]. Available:

<https://p-s-t.com/i-23157551-0-96-torsion-bar-deluxe-kit-rubber-b-e-body.html>. [Accessed: 12-Oct-2018].

[25] Peter, "Monroe® 58620 - Sensa-Trac™ Load Adjusting Rear Shock Absorbers," CARiD.com, 26-Apr-2017. [Online]. Available: <https://www.carid.com/monroe/sensa-trac-rear-load-adjusting-shock-mpn-58620.html>. [Accessed: 12-Oct-2018].

[26] "ST Suspension Mustang Front & Rear Anti-Sway Bars 52050 (94-04 GT, V6) - Free Shipping," AmericanMuscle.com. [Online]. Available: <https://www.americanmuscle.com/stsuspensions-swaybar-kit-9404gtv6.html>. [Accessed: 12-Oct-2018].

[27] W. Harris, "How Car Suspensions Work," HowStuffWorks, 11-May-2005. [Online]. Available: <https://auto.howstuffworks.com/car-suspension.htm>. [Accessed: 12-Oct-2018].

[28] "Your Vehicle's Steering Linkage," BlueStar Inspections. [Online]. Available: [https://www.bluestar.com/get\\_informed/article/your-vehicles-steering-linkage](https://www.bluestar.com/get_informed/article/your-vehicles-steering-linkage). [Accessed: 12-Oct-2018].

[29] "Rack and Pinion." Wikipedia, Wikimedia Foundation, 15 Dec. 2018, [en.wikipedia.org/wiki/Rack\\_and\\_pinion](en.wikipedia.org/wiki/Rack_and_pinion).

[30] Leonard, Derrick. "Light-Duty Steering Systems - Ppt Video Online Download." SlidePlayer, SlidePlayer, 12 July 2017, <slideplayer.com/slide/4640593/>.

[31] Aguilar, Mike. "Suspensions 101: Diagnosing the Two Basic Types of Front Suspensions." RacingJunk News, 30 Nov. 2016, [www.racingjunk.com/news/2015/08/27/suspensions-101-diagnosing-the-two-basic-types-of-front-suspensions/](http://www.racingjunk.com/news/2015/08/27/suspensions-101-diagnosing-the-two-basic-types-of-front-suspensions/).

[32] GrabCAD: Design Community, CAD Library, 3D Printing Software, [grabcad.com/library/formula-car-full-chassis](http://grabcad.com/library/formula-car-full-chassis).

[33] "How Does a Car Chassis Work?" The Pros and Cons of Saltwater Pools | DoItYourself.com, DoItYourself.com, 1 Aug. 2010, [www.doityourself.com/stry/how-does-a-car-chassis-work](http://www.doityourself.com/stry/how-does-a-car-chassis-work).

[34] "MST MS-01D PRO 1/10 Scale 4WD Electric Drift Car Chassis Kit 532017 RC Cars Cheap RC Cars RC Cars Cheap RC Cars Cheap RC Cars Online with \$549.72/Piece on Yunkun3's Store." Multi Color Artificial Ceramic Creative Succulents Potted With Flower Plant For Home Garden Wedding Table Calconies Decorations Table Decorations Potted Succulent Plants

Artificial Garden Plants Online with \$3.83/Piece on Tmos's Store | DHgate.com,  
[www.dhgate.com/store/product/mst-ms-01d-pro-1-10-scale-4wd-electric-drift/383339044.html](http://www.dhgate.com/store/product/mst-ms-01d-pro-1-10-scale-4wd-electric-drift/383339044.html).

[35] "Brushed vs Brushless Motors." Think RC, [www.thinkrc.com/faq/brushless-motors.php](http://www.thinkrc.com/faq/brushless-motors.php).

[36] "Everything You Need to Know About RC Batteries." RC Car Action, 31 July 2018,  
[www.rccaraction.com/everything-need-know-rc-batteries/](http://www.rccaraction.com/everything-need-know-rc-batteries/).

[37] "Servo Motor – 5Kg Torque." PotentialLabs, [potentiallabs.com/cart/servo-motor-5kg-india](http://potentiallabs.com/cart/servo-motor-5kg-india).

[38] "How Do Servo Motors Work?" How It Works: Xbox Kinect,  
[www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html](http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html).

[39] "FVT CBWI120A ESC /Brushless Speed Controller For 1/10 and 1/8Series RC Cars." Alex NLD,  
[alexnl.com/product/fvt-cbwi120a-esc-brushless-speed-controller-for-1-10-and-1-8series-rc-cars/](http://alexnl.com/product/fvt-cbwi120a-esc-brushless-speed-controller-for-1-10-and-1-8series-rc-cars/).

[40] "Introduction to Electronic Speed Control (ESC) Working and Applications." EIProCus -  
Electronic Projects for Engineering Students, [www.elprocus.com/](http://www.elprocus.com/).

[41] "Understanding How a Voltage Regulator Works." Analog Devices,  
<https://www.analog.com/en/technical-articles/how-voltage-regulator-works.html>.

[42] "The Zener Diode." Electronics Tutorials,  
[https://www.electronics-tutorials.ws/diode/diode\\_7.html](https://www.electronics-tutorials.ws/diode/diode_7.html).

[43] D. Hart, Power Electronics. McGraw-Hill Higher Education, 2011.

[44] How Radar Works. (n.d.). Retrieved April 14, 2019, from  
[https://www.bom.gov.au/australia/radar/about/what\\_is\\_radar.shtml](https://www.bom.gov.au/australia/radar/about/what_is_radar.shtml)

[45] US Department of Commerce, & National Oceanic and Atmospheric Administration. (2012,  
October 01). What is LIDAR. Retrieved April 14, 2019, from  
<https://oceanservice.noaa.gov/facts/lidar.html>

[46] GISGeography. (2018, February 23). Trilateration vs Triangulation – How GPS Receivers  
Work. Image Retrieved from <https://gisgeography.com/trilateration-triangulation-gps/>

[47] Groves, P. D. (2013). Principles of GNSS, inertial, and multisensor integrated navigation  
systems. Boston, MA: Artech house. pp. 13, 23-136, 137-162.

[48] Djuknic, G. M., & Richton, R. E. (2001). Geolocation and assisted GPS. Computer, (2),  
123-125.

- [49] How do Self-Driving Cars Work? 2018, October 1, from <https://www.iotforall.com/how-do-self-driving-cars-work/>.
- [50] "Lidar vs. Cameras for Self Driving Cars - What's Best?" AutoPilot Review, 9 Mar. 2019, <https://www.autopilotreview.com/lidar-vs-cameras-self-driving-cars/>.
- [51] "How Tesla Is Ushering in the Age of the Learning Car." Fortune, <https://fortune.com/2015/10/16/how-tesla-autopilot-learns/>.
- [52] Li, C., Zhang, S., Cao, Y. (2013). One new onboard calibration scheme for gimballed IMU, Measurement, Volume 46, Issue 8, Pp. 2359-2375, ISSN 0263-2241.
- [53] Vu, H., Palacios, A., In, V., Longhini, P., & Neff, J. D. (2011). A drive-free vibratory gyroscope. Chaos (Woodbury, N.Y.). 21. 013103. doi:10.1063/1.3532802.
- [54] Hadj said, Mohamed & Tounsi, Farès & Gkotsis, P & Mezghani, Brahim & Francis, Laurent A.. (2017). A Resonant Microstructure Tunability Analysis for an Out-of-plane Capacitive Detection MEMS Magnetometer. Microsystem Technologies. 23. 2599-2608. 10.1007/s00542-016-3093-y.
- [55] Roetenberg, D., Luinge, H. J., Baten, C. T., & Veltink, P. H. (2005). Compensation of magnetic disturbances improves inertial and magnetic sensing of human body segment orientation. IEEE Transactions on neural systems and rehabilitation engineering, 13(3), 395-405.
- [56] Titterton, D., and Weston, J. L.(2004). Strapdown inertial navigation technology. Reston, Virginia: The American Institute of Aeronautics. pp. 56.
- [57] Introduction to Digital Tachometer Circuit Working with 8051 and Types. (2018, October 13). Retrieved April 21, 2019, from <https://www.elprocus.com/introduction-to-digital-tachometer-circuit-working-with-8051/>
- [58] Artificial Intelligence. (n.d.). Retrieved March 18, 2019, from [https://www.merriam-webster.com/dictionary/artificial intelligence](https://www.merriam-webster.com/dictionary/artificial%20intelligence)
- [59] Valkov, V., & Valkov, V. (2017, May 19). Creating a Neural Network from Scratch-TensorFlow for Hackers (Part IV). Retrieved April 23, 2019, from <https://medium.com/@curiously/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8>
- [60] A Basic Introduction to Neural Networks. (n.d.). Retrieved April 23, 2019, from <http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>

- [61] Qiao, Junqing. (2016). Semi-Autonomous Wheelchair Navigation With Statistical Context Prediction. WORCESTER POLYTECHNIC INSTITUTE, [web.wpi.edu/Pubs/ETD/Available/etd-053016-140503/unrestricted/Thesis.pdf](http://web.wpi.edu/Pubs/ETD/Available/etd-053016-140503/unrestricted/Thesis.pdf).
- [62] Garrison Hefter, Mitch Read, Dylan Roncati, Md Shamsur, & Rahman Saikat. (2018, April 26). Autonomous Campus Mobility Platform. WORCESTER POLYTECHNIC INSTITUTE
- [63] "Arduino Mega 2560 Rev3." Arduino, Retrieved April 22, 2019 from <https://store.arduino.cc/usa/arduino-mega-2560-rev3>.
- [64] Intel® Compute Stick STK1A32SC. (n.d.). Retrieved April 23, 2019, from <https://www.intel.com/content/www/us/en/products/boards-kits/compute-stick/stk1a32sc.html>
- [65] Bringing the Power of AI to Millions of Devices. (n.d.). Retrieved April 23, 2019, from <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/>
- [66] XU4 Special Price. (n.d.). Retrieved April 23, 2019, from <https://www.hardkernel.com/shop/odroid-xu4-special-price/>
- [67] Banggood.com. (n.d.). US\$14.28 JLB Racing CHEETAH 1/10 Brushless RC Car Differential Set EA1057 RC Parts from Toys Hobbies and Robot on banggood.com. Retrieved April 25, 2019, from [https://www.banggood.com/JLB-Racing-CHEETAH-110-Brushless-RC-Car-Differential-Set-EA1057-p-1079202.html?rmmds=buy&cur\\_warehouse=CN](https://www.banggood.com/JLB-Racing-CHEETAH-110-Brushless-RC-Car-Differential-Set-EA1057-p-1079202.html?rmmds=buy&cur_warehouse=CN)
- [68] Amadget Wide Angle Fish-eye Camera Module. (n.d.). Retrieved April 25, 2019, from [https://www.amazon.com/gp/product/B07FDB7S7F/ref=ppx\\_yo\\_dt\\_b\\_asin\\_title\\_o02\\_s00?ie=UTF8&psc=1](https://www.amazon.com/gp/product/B07FDB7S7F/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1)
- [69] Karnopp, D., Margolis, D. L., & Rosenberg, R. C. (2012). System dynamics: Modeling, simulation, and control of mechatronic systems. Hoboken (New Jersey): Wiley.
- [70] DHT11: Image.dfrobot.com (2019) Retrieved April 22, 2019 from, <https://image.dfrobot.com/image/data/KIT0003/DHT11%20datasheet.pdf>
- [71] Thomas, L. (2019). DHT22 Retrieved April 22, 2019 from, <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>
- [72] Cityos-air.readme.io. (2019). 5. DHT22 - Digital Temperature and Humidity Sensor Retrieved April 24, 2019 from, <https://cityos-air.readme.io/docs/4-dht22-digital-temperature-humidity-sensor>



- [73] Noctua.at. (2019). Noctua Cooling Fan Retrieved 25 April, 2019 from, [https://noctua.at/media/blfa\\_files/infosheet/noctua\\_nf\\_a4x10\\_5v\\_pwm\\_datasheet\\_en.pdf](https://noctua.at/media/blfa_files/infosheet/noctua_nf_a4x10_5v_pwm_datasheet_en.pdf)
- [74] Pololu.com. (2019). Pololu - MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carrier). Retrieved April 24, 2019 from <https://www.pololu.com/product/2468>
- [75] Superdroidrobots.com (2019). Pololu MinIMU-9 v2 Gyro, Accelerometer and Compass (L3GD20 and LSM303DLHC Carrier). Retrieved April 25, 2019, from <https://www.superdroidrobots.com/shop/item.aspx/pololu-minimu-9-v2-gyro-accelerometer-and-compass-l3gd20-and-lsm303dlhc-carrier/1361/>
- [76] KY-003 Hall Magnetic Sensor Module. (2018, November 19). Retrieved April 25, 2019, from <https://arduinomodules.info/ky-003-hall-magnetic-sensor-module/>
- [77] 6mm X 6mm N35 Strong Disc Cylinder Round Rare Earth Neodymium Magnets Nickel Plated 6mm Magnets - UMAGNETS. (n.d.). Retrieved April 25, 2019, from <https://www.umagnets.com/p/disc-magnet-n50-6mm-x-6mm-nickel-plated/>
- [78] GitHub. (2019). pololu/lsm303-arduino. Retrieved 25 Apr. 2019 from, <https://github.com/pololu/lsm303-arduino>.
- [79] Pieter-jan.com. (2019). Reading a IMU Without Kalman: The Complementary Filter | Pieter-Jan.com. Retrieved April 24, 2019 from <https://www.pieter-jan.com/node/11>
- [80] Altium.com. (2019). PCB Design Software & Tools | Altium. Retrieved April 24, 2019 from <https://www.altium.com/>
- [81] Oshpark.com. (2019). OSH Park ~. Retrieved April 24, 2019 from <https://oshpark.com/>
- [82] Marsauto, "marsauto/europilot," *GitHub*, 21-Mar-2018. Retrieved April 24, 2019 from <https://github.com/marsauto/europilot>.
- [83] 3D Printing Software | Simplify3D. (n.d.). Retrieved April 26, 2019, from <https://www.simplify3d.com/>

# Appendices

## Appendix A: Finite Element Analysis

Table A-1: Parameters for FEA on Front Lower Suspension Link

Fixture Type	Location	
Fixed Geometry	Rear Pin	
Fixed Geometry	Front Pin	
Loading Type		
Force	0.2248	lbf
Results		
Max Von Mises Stress	315600	Pa
Maximum Displacement	0.00494	mm
Maximum Strain	0.001034	

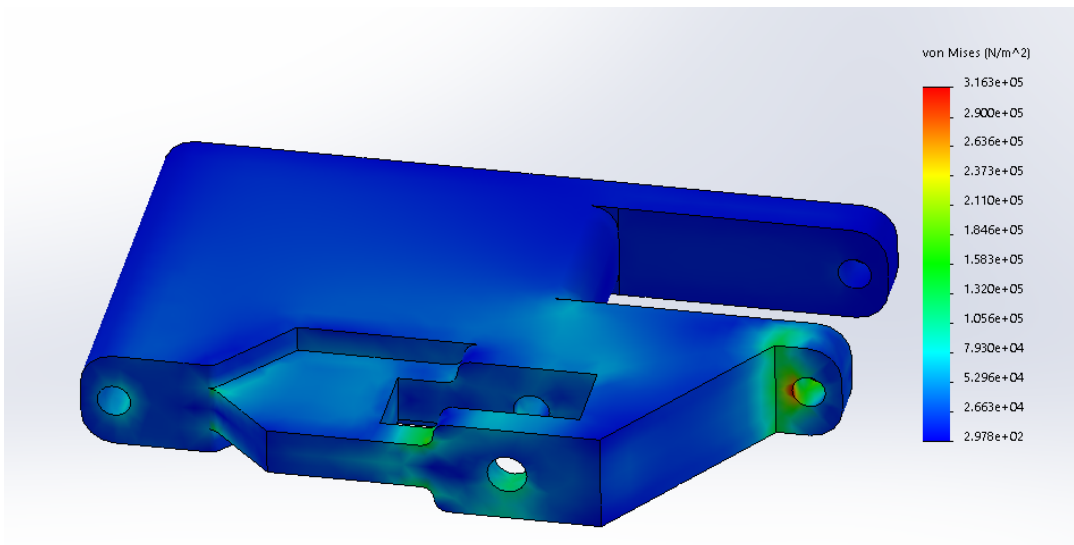


Figure A-1: Von Mises Stress Analysis for Front Suspension Lower Link

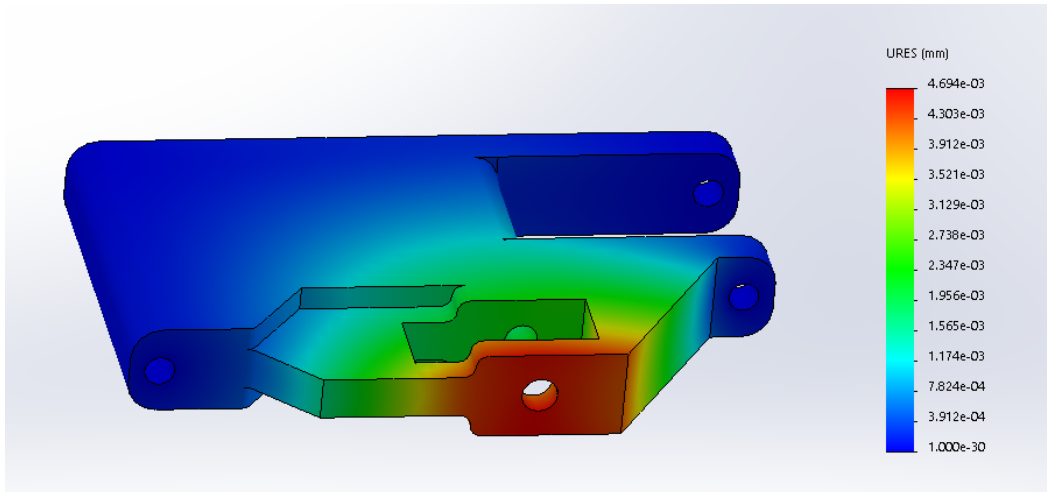


Figure A-2: Displacement Analysis for Front Suspension Lower Link

Table A-2: Parameters Used for FEA on Servo Horn

<b>Material Used</b>	<b>ABS</b>	
Elastic Modulus	2000000000	Pa
Poisson's Ratio	0.394	
Mass density	1020	kg/m <sup>3</sup>
Tensile Strength	30000000	Pa
Shear Modulus	318900000	Pa
<b>Fixture Type</b>	<b>Location</b>	
Fixed Geometry	Countersunk Screw Hole	
Fixed Geometry	Pin Hole for Steering Arm Connection	
<b>Applied Load</b>		
Torque	146	oz-in
<b>Results</b>		
Max Von Mises Stress	24700000	Pa
Maximum Displacement	0.2317	mm
Maximum Strain	0.0147	

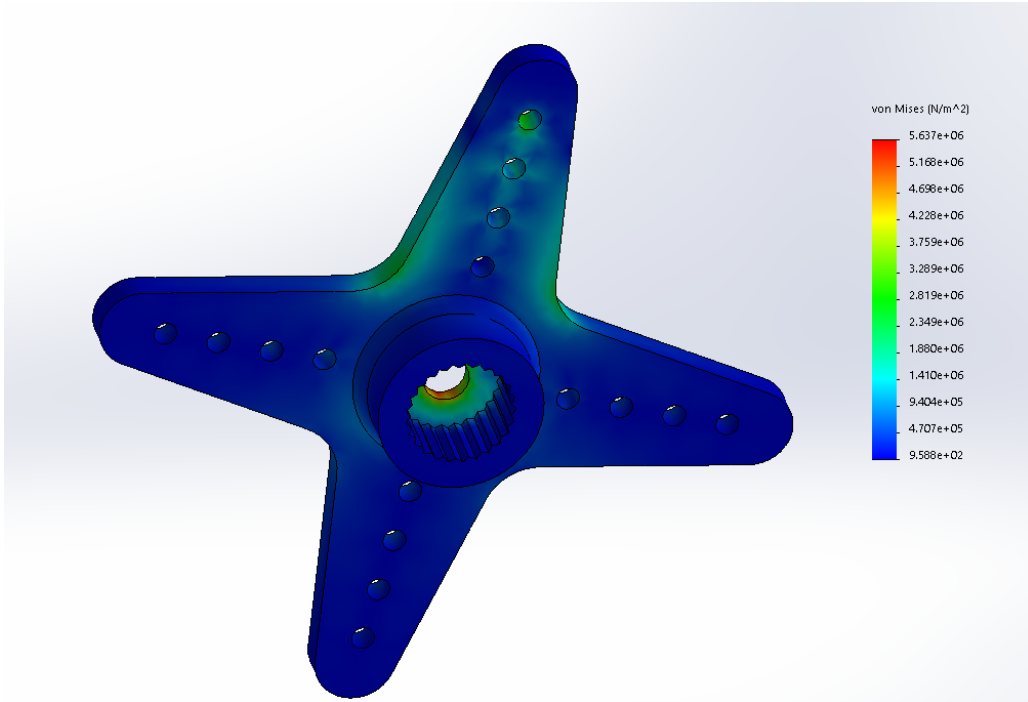


Figure A-3: Von Mises Stress Analysis for Servo Horn (Bottom View)

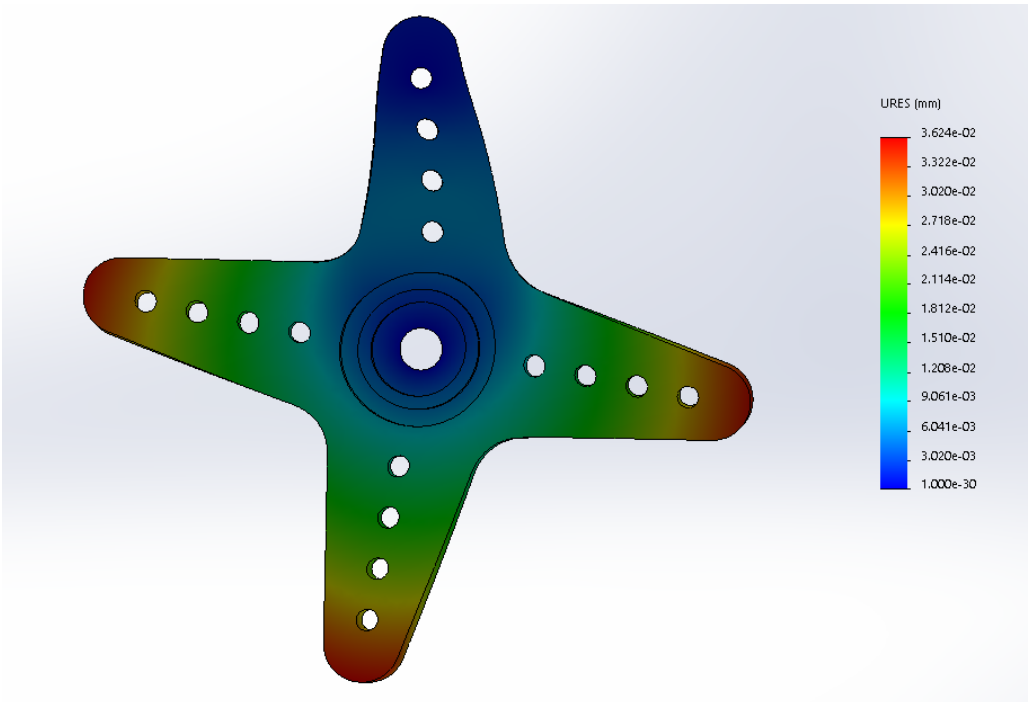


Figure A-4: Displacement Analysis for Servo Horn (Top View)

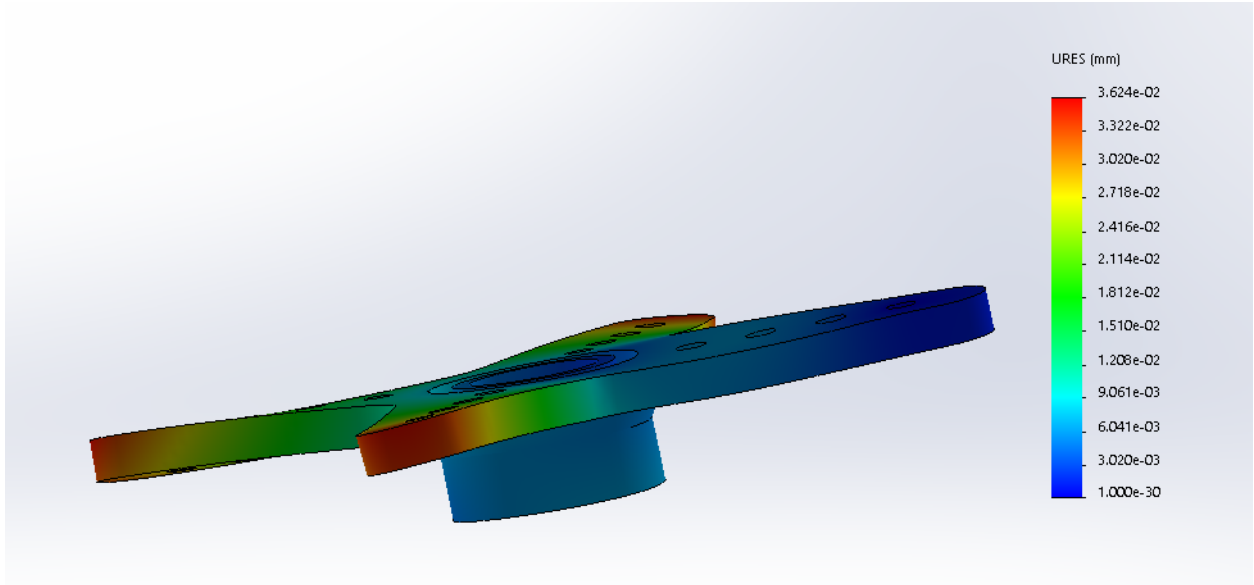


Figure A-5: Displacement Analysis for Servo Horn (Side View)

## **Appendix B: Design for Manufacturability Manual**

This document is a collection of design for manufacturability practices that can be used to help make your parts and assemblies more manufacturable. This guide is split up into sections based on general practices as well as method specific recommendations.

### General Design Considerations

#### **1. Understand the functional requirements**

The first step in designing a part that will be manufacturable is to understand the functional requirements of the part. In addition, the part's interaction with other components, geometry requirement, tolerances, etc. are all important considerations.

#### **2. Incorporate manufacturing into the design process**

A key part of design for manufacturability is understanding the processes available to produce the part. Each process has its own design challenges and recommendations. For example, a part produced using a milling machine requires different design considerations than a 3D printed part. These design considerations should be incorporated into the design from the beginning as it is easier to design a part with manufacturability in mind than to modify an existing part to be more manufacturable.

Some suggestions and guidelines for common practices can be found below.

#### **3. Reduce design complexity**

More complex designs usually means that more operations are needed in order to make your part. This translates to increased manufacturing lead time and cost. . Reducing design complexity can include limiting the number of features on a part to only those that are necessary, but it can also include things like using standard hole sizes, and using standard parts instead of custom parts. By reusing parts, a lot of time spent in setting up manufacturing operations can also be reduced.

#### **4. Buy off the shelf when available**

It is almost always better to buy commercially made, standard parts instead of making them if those parts meet all of the functional requirements. This saves time and possibly cost in the process.

### **5. Work with lab staff early**

If you are going to be using a lab facility to make your part, schedule a consultation with lab staff early and have them critique your design. For example, if you want to make a part using the CNC machines in Wsahburn shops, scheduling a meeting with one of the Sr. Instructional Lab Technicians, Ian Anderson or James Loisselle. By meeting with them early, you can get feedback on your design, order material, and get an estimate of how much time it will take.

Scheduling early is the most important part of this. It is unfair to the lab staff to come in last minute and expect them to drop what they're doing to help you make your part. Plan ahead and work with the lab staff from the beginning. Similarly, if you plan to have parts printed in Foisie's prototyping lab, expect a few days delay and check with lab staff to see the expected wait time.

### **6. Test your design early**

There is nothing more frustrating than having to redesign a part of assembly over and over again because of a minor design flaw. In your CAD models, make sure to model everything, including any hardware you may be using to attach components together. While it might take more time initially, it can save you time in redesigns. The solidworks toolbox can be a great resource for finding models of standard hardware.

Design Considerations for Milled Parts:

***Design Limitations for Milled Parts:***

Square Internal Corners

It is not possible to create square internal corners using standard milling tools. . Internal corner radii are required as shown in the figures below.

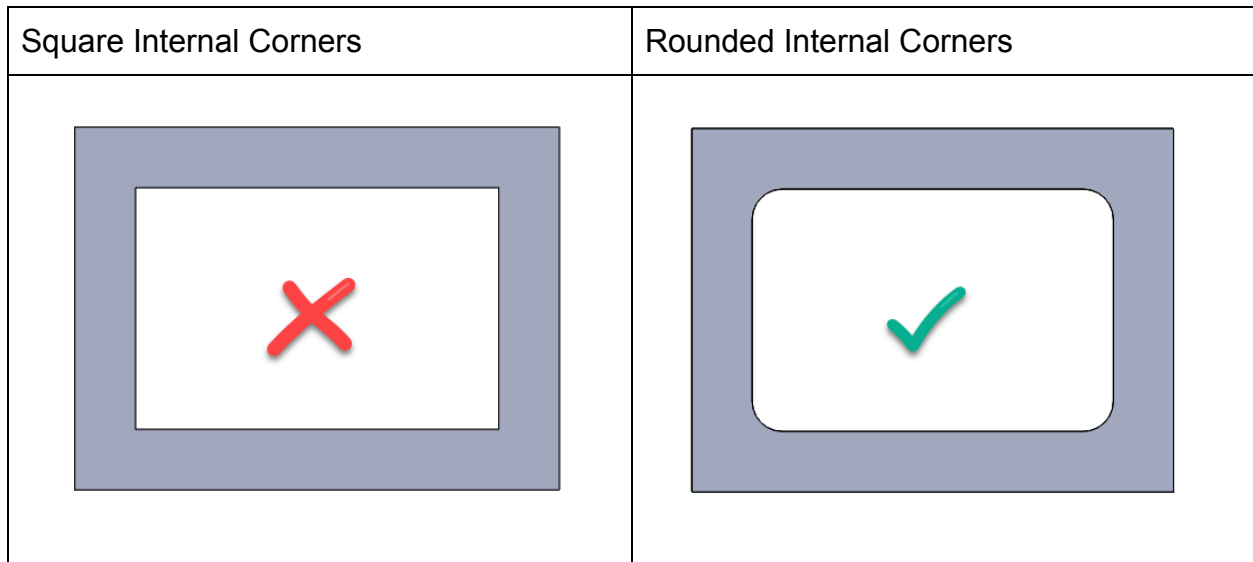


Figure B-1: Square vs. Rounded Internal Corners

One important thing to note when adding radii to internal corners, is that the radius of the corner should be greater than or equal to the radius of the tool used to create the feature. Larger tools remove material more quickly and are much more rigid, but smaller tools can make smaller radii. Some common diameters of tools available in Washburn Shops are  $\frac{3}{8}$  ",  $\frac{1}{4}$  ", and  $\frac{1}{8}$  ".

Fillets on Everything

While Fillets look nice, the machining process involved can be laborious . Each radius of fillet needs a custom tool, meaning that a  $\frac{1}{4}$ " fillet tool can only make  $\frac{1}{4}$ " fillets. Chamfers are better alternatives, as one chamfer tool can make many different size and depth chamfers, An image showing both fillets and chamfers can be seen below.



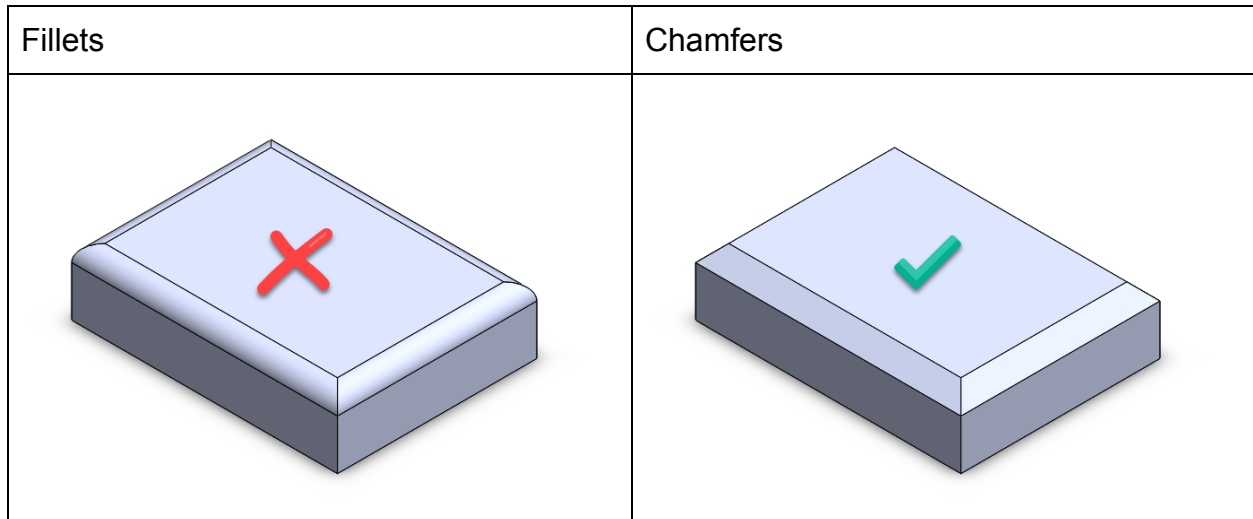


Figure B-2: Filletts vs. Chamfers

If filletts are absolutely needed, please use standard size filletts like 1/8" or 1/4".

#### Thin Walled Parts:

Structural integrity of the work piece is very important during machining operation. Additionally, the machining process itself exerts a force on the workpiece that can cause deflection. For this reason, It is often very difficult to machine thin walled features. The depth of the feature can also affect the structural integrity, so a shallow pocket can tolerate thinner walls than a deeper pocket.

#### Fixturing

An extremely important thing to remember when thinking about part design, is how your part will be held in the machine. Often, each face that you have features on means that you have to do an entirely new set up for the part. Since the tool used in a 3-axis mill, like those in Washburn Shops, can only move in the Z direction, they won't be able to reach features on all sides of the part. On a part like the one pictured below, there are features on three sides of the part, meaning that there would need to be at least three machining setups in order to make this part.

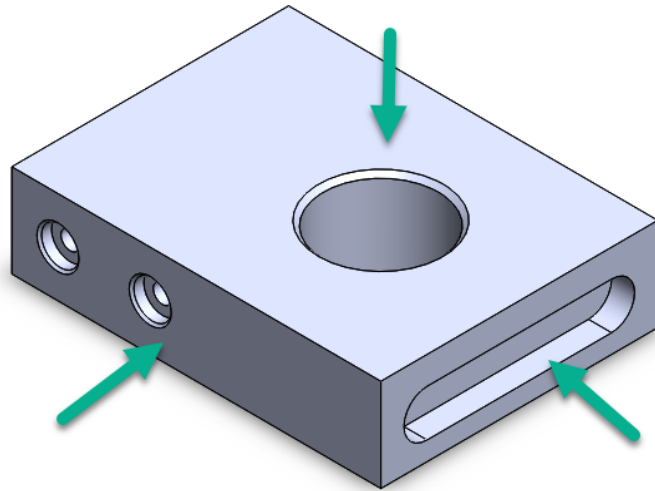


Figure B-3: Part Fixturing

As you can see, this part would need to be fixtured three times in order to reach all of the features. If the part could be redesigned to only have features on one of two sides, that would greatly reduce the setup and machining time.

Another difficulty when it comes to fixturing is having unusual outside geometry. In order to ensure that you have enough clamping pressure, you'll want your fixture to closely match the shape of your part. If the fixture doesn't closely match the shape of the part you can risk your part becoming unfixtured. In the figure below, you can see an example of a fixture (blue) closely matching the part geometry (red), and example of a fixture not closely matching the part geometry

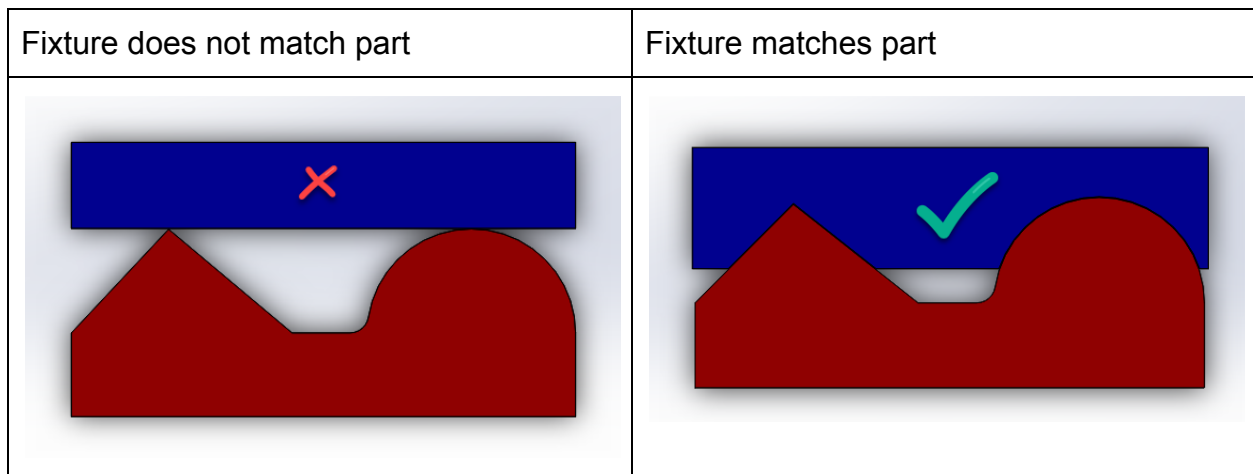


Figure B-4: Fixture Matching

As you can see above, the fixture in blue is designed to match the features on the part to ensure that the part has adequate clamping force. This adds an extra step to the manufacturing process, and makes it more difficult to manufacture the part.

When possible, you should reduce the number of weirdly shaped outside features, to make fixturing easier.

#### Hole Sizes and the Tap Drill Chart

One of the easiest ways to make life easier for yourself or the machinist you're working with is to size holes correctly for the hardware you want to use. Every piece of hardware, like a screw, bolt, or pin, has a different size hole it's meant to go in. Additionally, if you want your part to have threads, there are certain sizes of drilled holes you need to make, so that the tap that will cut the threads has the right amount of material. This information is widely available but I will share two great ways to find this information, the tap drill chart, and the hole wizard.

The tap drill chart below shows the drill sizes that should be using when making tapped features. An excerpt from a tap drill chart can be seen below.

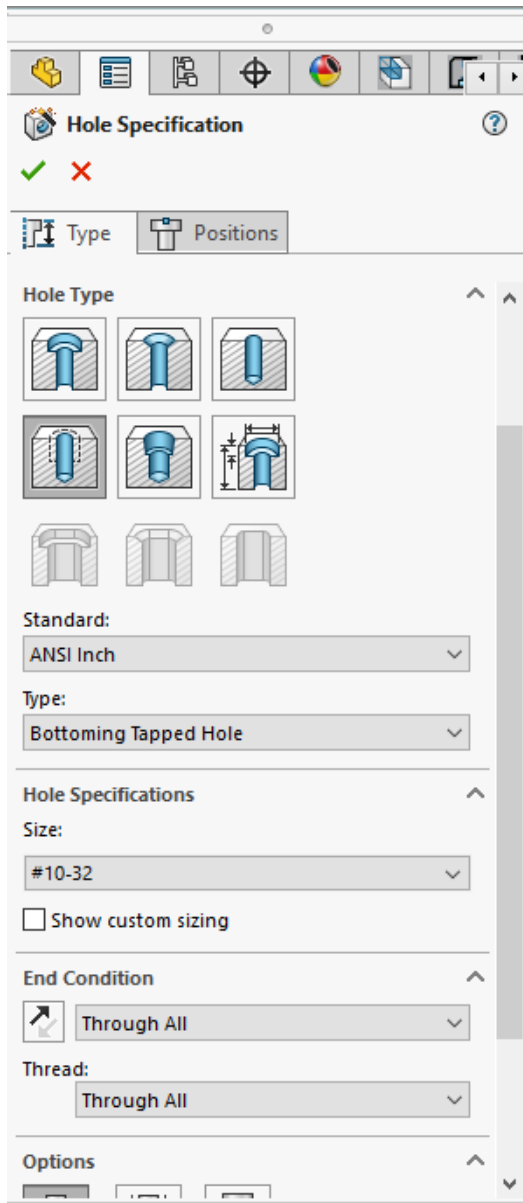
Screw Size	Major Diameter	Threads Per Inch	Minor Diameter	Tap Drill				Clearance Drill			
				75% Thread for Aluminum, Brass, & Plastics		50% Thread for Steel, Stainless, & Iron		Close Fit		Free Fit	
				Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.	Drill Size	Decimal Equiv.
0	.0600	80	.0447	3/64	.0469	55	.0520	52	.0635	50	.0700
1	.0730	64	.0538	53	.0595	1/16	.0625	48	.0760	46	.0810
		72	.0560	53	.0595	52	.0635				
2	.0860	56	.0641	50	.0700	49	.0730	43	.0890	41	.0960
		64	.0668	50	.0700	48	.0760				
3	.0990	48	.0734	47	.0785	44	.0860	37	.1040	35	.1100
		56	.0771	45	.0820	43	.0890				
4	.1120	40	.0813	43	.0890	41	.0960	32	.1160	30	.1285
		48	.0864	42	.0935	40	.0980				
5	.125	40	.0943	38	.1015	7/64	.1094	30	.1285	29	.1360
		44	.0971	37	.1040	35	.1100				
6	.138	32	.0997	36	.1065	32	.1160	27	.1440	25	.1495
		40	.1073	33	.1130	31	.1200				
8	.1640	32	.1257	29	.1360	27	.1440	18	.1695	16	.1770
		36	.1299	29	.1360	26	.1470				
10	.1900	24	.1389	25	.1495	20	.1610	9	.1960	7	.2010
		32	.1517	21	.1590	18	.1695				
12	.2160	24	.1649	16	.1770	12	.1890	2	.2210	1	.2280
		28	.1722	14	.1820	10	.1935				
		32	.1777	13	.1850	9	.1960				
1/4	.2500	20	.1887	7	.2010	7/32	.2188	F	.2570	H	.2660
		28	.2062	3	.2130	1	.2280				
		32	.2117	7/32	.2188	1	.2280				

Figure B-5: Tap Drill Chart

Let's say that you want to make a clearance hole for a 10-32 screw in a part. The first step would be to find the correct row for a 10-32 screw by first looking in the screw size column and then finding the row for 32 threads per inch (green arrows). Once you've found the row you want, you would decide whether you want a close fit, or a free fit on your clearance hole. A close fit will more closely match the outer diameter of your part, but a free fit leaves more space and is good when the accuracy of the hole isn't as important.

If you wanted to make a tapped 10-32 hole, you would look for the tap drill size instead of the clearance drill size. The tap drill size will always be smaller than the clearance hole. The first thing to know is the type of material you will be tapping into, as this will control the percentage of thread you will be using. For softer materials, a higher percentage of thread is better but on hard materials it's better to use a slightly larger drill and smaller percentage of threads, so that tap doesn't have to remove as much material. Once you know what material and size you are using, all you need to do is find

the correct drill size. For a 10-32 tapped hole in aluminum, we would use either a #21 drill or a 0.1590" drill (purple arrows).



When using the hole wizard in SolidWorks, the program does all of the same steps that you would do if you were using a tap drill chart. A picture showing the hole wizard can be seen to the left.

Under the hole type tab, you can see a lot of common types of holes including, plain holes, tapped holes, countersunk holes, and counterbored holes. Select the type you want, then move on to the “standard” drop down. This is where you can select whether you want to use standard english or metric sizes. You can then adjust the type of hole if you want. If you are using a non- tapped hole, you will have a lot of options under the type drop down. Some of the most useful ones are drill sizes, dowel sizes,

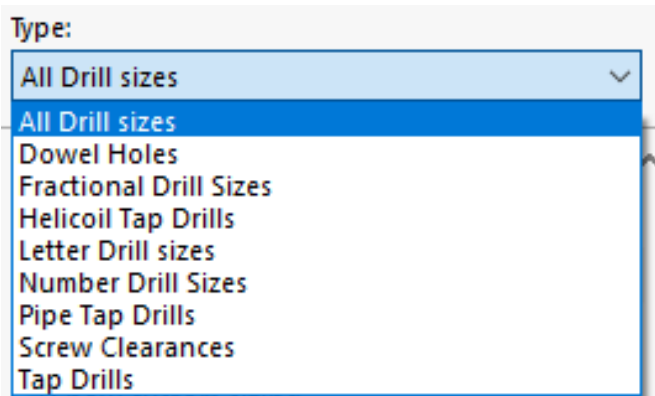


Figure B-6: SolidWorks Hole Wizard

screw clearances, and tap drills. A picture of the drop down can be seen below

Next, under the hole specifications tab, you can select the size of the hole you want to make, or the screw that you are designing around. Once you’ve done this, all you have

to tell the program is the end condition of your hole, whether your hole goes all the way through or if it stops at a certain depth.

The last step, is to go to the positions tab at the top and tell the program where to place your holes and you're done!

Why should you use the hole wizard instead of extrude cutting the hole you want? First, by using the hole wizard, you can ensure that the hole and the corresponding hardware matches.. Second, the hole wizard makes it extremely easy to change hole sizes if you need to change the hardware.. Third, by using the hole wizard you ensure that the hole size you want to make actually matches a standard drill size. This is extremely important because a drill will make a much more accurately sized hole than a milled pocket, and will do so much more quickly.

#### More Common Tap Problems

Other common tapping problems you may run into are trying to make threads to the very end of a blind hole. The problem with this, is that when a hole is tapped, you create chips that build up underneath the tap, and when you get to the bottom of a hole, there's nowhere for those chips to go, so they exert a force on the tap and cause it to break.

Other common problems that can occur, is when you have to parts that you want to tap together by putting threads in both of the parts. For a lot of reasons this is a bad idea. First, in order for that to work, the end of threads from the first piece need to line up perfectly with start of the threads of the second piece. Second, the holes need to be perfectly coaxial, otherwise the screw will not go in, even if the threads happen to line up. If you can manage to do both of those things, you also need to make sure that our parts will never be taken apart, as you likely won't be able to put them back together accurately enough that your screw will go in. Instead, one of the parts can have a clearance hole, and the other can have a tapped hole. This gives you much more reliability and makes the manufacturing easier.

#### Internal Features

Internal features are features that are inside of other features and are often extremely difficult or impossible to make. An example of an internal feature can be seen below.

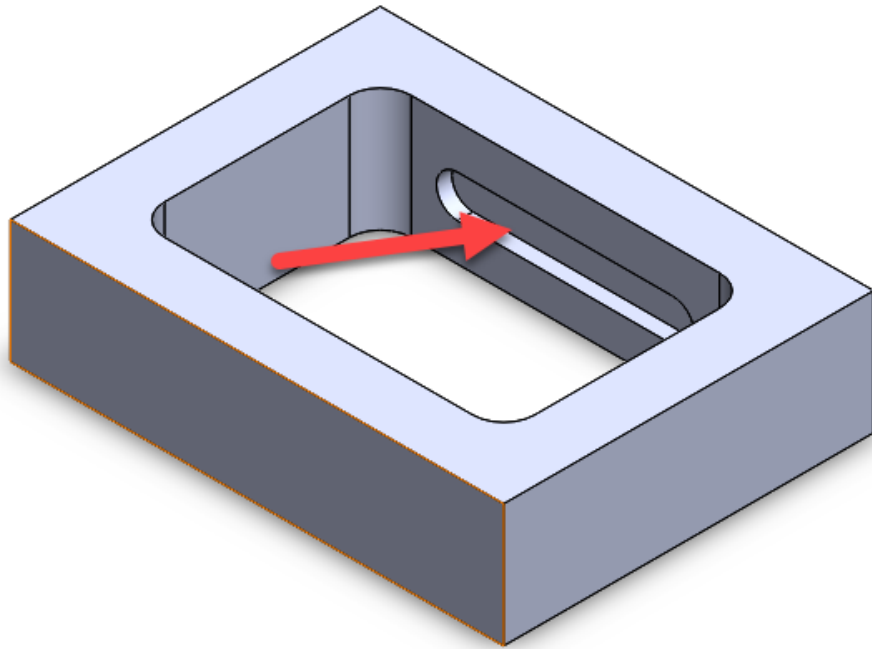


Figure B-7: Internal Part Features

The main problem with internal features is that it is extremely difficult or impossible to get tooling inside of the larger pocket to be able to make this part.

#### Making One Complicated Part v. Making Several Easy Parts

Slightly contrary to what may be obvious, is that it is often much easier to make several easy parts than to try and make one difficult part. If we wanted to make the part pictured above in the internal features section, what we could do instead, is split the part into two halves that fit together. A figure of what this might look like can be seen below

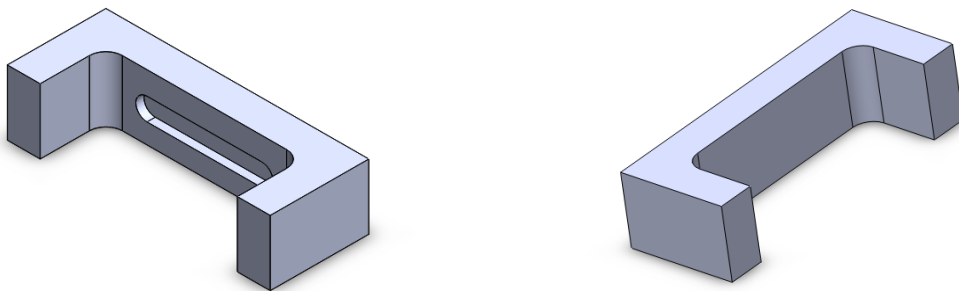


Figure B-8: Simple vs. Hard Parts

## Mixing English and Metric Units

The challenging thing about mixing english and metric parts is that particularly in shops like Washburn, the vast majority of our tooling is in standard english sizes. We do have a small amount of metric tooling, but not nearly the same variety and quantity that we have of english tooling. The other problem with mixing english and metric units is that the machines in Washburn Shops are american made HAAS machines and run, by default, in english units.

## Material Choices

Some materials are easier to work with than others. Some materials like aluminum, brass, and steel are commonly used in machine shops like the one in Washburn. These materials are easy to work with and can make functional parts. A list of good material choices can be found below.

### Good Material Choices for CNC machining:

- Most Aluminum Alloys
- Some Steel Alloys
- Most Brass Alloys
- Some Copper Alloys
- Some plastics (Delrin, ABS, Nylon 6/6, HDPE, Polycarbonate)

### Bad Material Choices for CNC machining:

- Extremely hard materials (Ex. Hardened steel)
- Titanium
- Fiberglass
- Carbon Fiber
- Pure Copper

## Choosing and Buying Stock

Before trying to manufacture a part, you need to understand what type of stock you will need in order to make your part. Stock is the material that you start from and machine away to get your finished part. For smaller parts, stock can be ordered online from places like MSC direct or McMaster Carr. For larger quantities or specialized materials, it may be worth it to consult with Lab Staff so that they can help you find the right distributor.



## Design Considerations for Turned Parts

### Part Length

When making parts to be made in a lathe, overall length is a critical design consideration. Due to the cutting forces that occur in a lathe, part diameter to part length should not exceed a 1:3 ratio for parts that will be unsupported, or a 1:5 ratio for parts that will be supported by with a tailstock. Examples showing the difference between supported and unsupported cuts can be seen below.

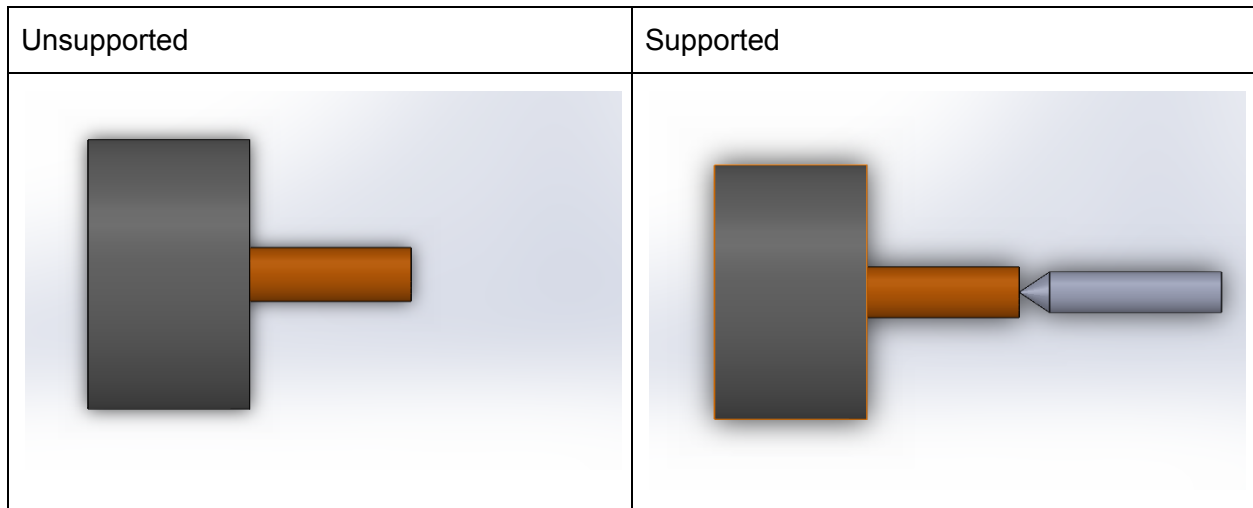


Figure B-9: Unsupported vs. Supported Cuts

### Stock Size

Each of the lathes in Washburn Shops has a maximum size of stock that it can accommodate. The smaller lathes can accommodate stock up to about 2" in diameter, where the larger lathe can accommodate up to about 10" in some situations.

### Fixturing

While the lathes can accommodate a number of different sizes of stock, there are limitations to the way in which the stock is held. If you plan to use a collet chuck, try to keep the outside diameter of your stock to standard english fractional sizes ( $\frac{1}{8}$ ,  $\frac{1}{4}$ , etc.). If you plan to use a jaw chuck, you can accommodate many intermediate sizes but there are additional steps required during setup.

## Design Considerations for 3D Printed Parts

When considering designs for parts that are to be 3D Printed it is important to consider many different things than would be considered for traditionally manufactured parts.

### 1. Types of Additive Manufacturing:

There are many different types of additive manufacturing for plastic parts. For this paper, we will be discussing the two major types of plastic 3D Printing which are material extrusion and vat polymerization. Two types of vat polymerization are SLA and DLP which both cure a resin and FDM which extrudes plastic one layer at a time. SLA 3D Printing is known as stereolithography and uses a moving laser beam to cure a vat of resin into the geometry specified by the software for the printer. DLP or Digital Light Projection uses an LCD screen to project each layer onto the vat of uncured resin to cure the layers faster than SLA. This process does not have the same resolution as SLA printing as the pixel size of the LCD screen determines the resolution of the X and Y dimensions of the part. FDM or Fused Deposition Modelling is the most commonly used in hobbyist and professional 3D Printing. This method for 3D Printing uses a filament of plastic which is passed through a heating element and heated to the glass transition state of the plastic, allowing the heated plastic to bond with other plastic but not be extruded as a liquid. This extruder builds up the model by extruding one layer at a time. The rest of this informative paper will highlight printing using an FDM printer.

### 2. Material Choice:

Recently with advances in additive manufacturing, there are now even more choices for materials. The two major types of filament used in FDM machines are PLA and ABS. PLA is generally easier to print with and is a harder material while ABS has more challenges when printing but is a much sturdier plastic (the same plastic used for manufacturing LEGO bricks). Other materials that are gaining popularity are flexible filaments, nylon, and composites. Each one comes with its own positives and drawbacks, nylon has less friction between parts but like ABS is challenging to print, flexible filaments allow for flexible parts but have difficulty extruding and adhering to the build plate, and composite filaments are generally a mixture of PLA with other particulates of either wood or metals.

### 3. Layer Lines:

For additive manufacturing, the way in which the lines are deposited for the individual layer matters greatly. For FDM machines, all exposed surfaces are deposited first at a slower speed to improve the surface quality. The internal structure that is printed afterwards is either a hatching pattern or an infill pattern. This hatching pattern is printed in a way to keep all parts of the layer at a consistent temperature to allow for the filament to properly fuse to itself while preventing heat build up and melting.

### 4. Layer Thickness:

Layer thickness plays a major role in strength and resolution in 3D printed parts. The smaller the individual layers, the higher the resolution. However, when the layer lines are thinner the strength of the part is weaker because the weakest point of the part

is between the layers. Each printer has a different acceptable range for layer heights meaning that there is no one perfect layer height for prints.

#### 5. Part Orientation:

Orienting the part on the bed can change the entire structural integrity of the part and prevent the need for support structures or otherwise tricky geometries to print. In general, to avoid overhangs any angle that the surface makes with the vertical over 45 degrees would require support structure. By reorienting the part, one can reduce the amount of support material and save on plastic. In addition to this, the weakest part of the print are where the layers are fused to each other, so by changing the orientation, one can make the print as strong as possible in the proper ways. An example of how to optimize orientations can be shown in the below picture.



Figure B-10: 3D Printing Orientation. (Reproduced as is from [83]).

#### 6. Support:

Support material is generally required for any portions of the part that are in midair. Support structures and properties can be changed in most slicer settings. These changes allow the user to find the perfect balance between supports that easily come off and also leave a good surface finish on the part. Certain 3D printers are able to print a dissolvable support structure in addition to the main part material.

#### 7. Overhangs:

Overhangs are when the part has a portion of it above the build surface and is only connected to the main part by one side. As previously mentioned, overhangs need support material when they are over 45 degrees from the vertical to reduce misprints. An example of overhangs on a perfectly calibrated overhang bridge is shown.



Figure B-11: 3D Printing Overhangs. (Reproduced as is from [83]).

#### 8. Bridging:

Bridging is printing anything in a part that is supported by both sides but not the bottom of the part. In order to print bridges without support sufficient cooling is needed. Otherwise it is best to have supports on the bridge. Many 3D Printers and reviews of 3D printers will state the maximum length of a bridge before support material is needed. Below is shown a picture of what happens to the material as larger and larger bridges are printed.

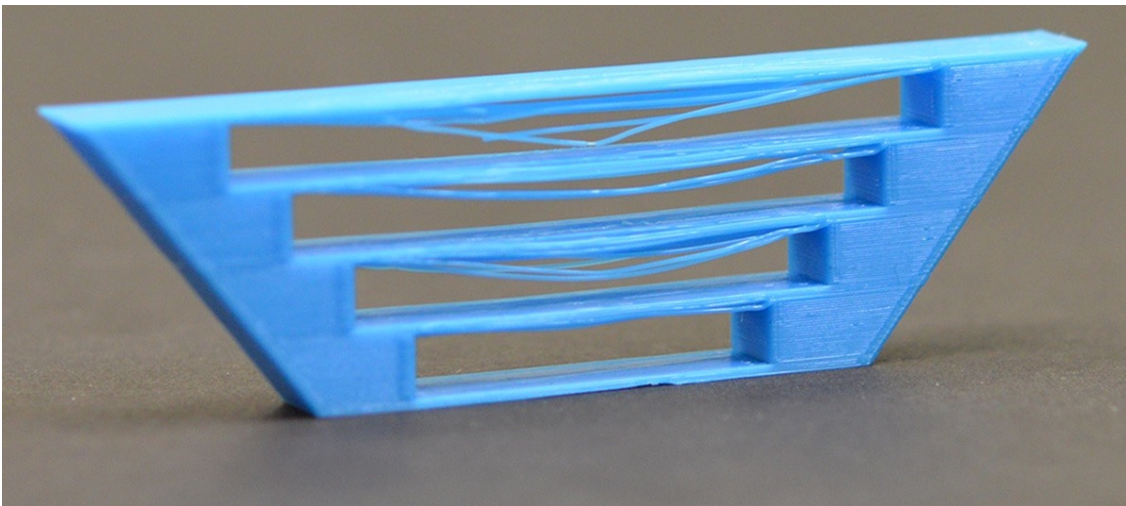


Figure B-12: 3D Printing Bridging. (Reproduced as is from [83]).

## 9. Vertical Towers:

When printing parts, geometries matter not only to make challenging prints easier but also to make sure that the parts have structural integrity. For vertical towers, a chamfer or fillet is used to build up the layers gradually to reduce strain on the part where the tower connects to the base. This is shown in the following diagram:

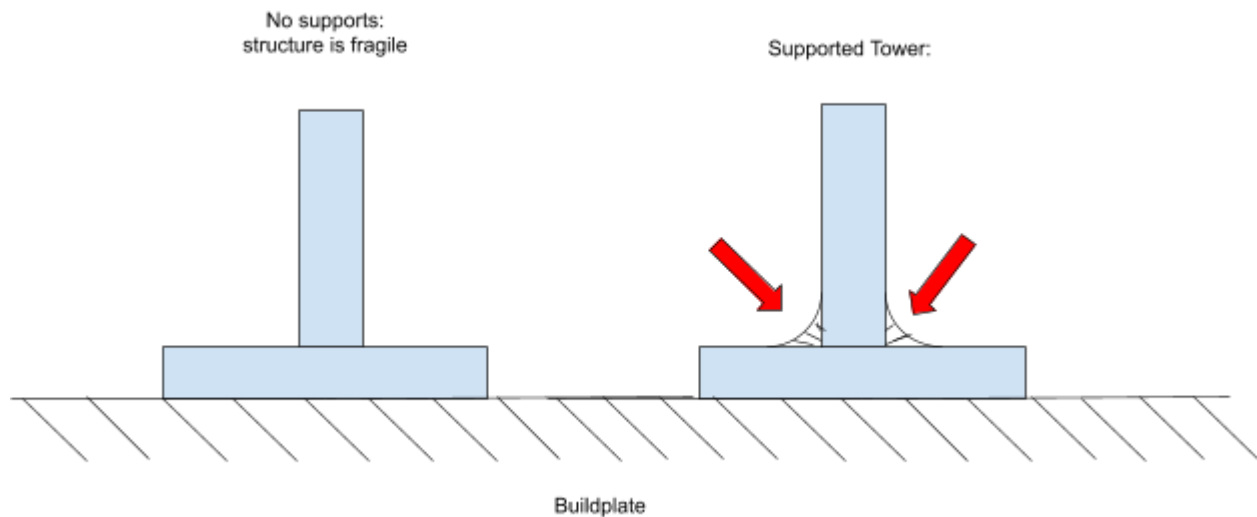


Figure B-13: 3D Printing Vertical Towers

## 10. Infill:

Infill is the percentage of plastic inside of the part. This is done so that parts that do not need to be as strong do not need to waste extra plastic and time printing. Infill should be between 20-70% to ensure a good print. When the Infill percentage is too low, the top face of the part will be unsupported by the infill underneath it. When the infill percentage is above 70% there is a good chance that the part will cool too fast between layers preventing proper layer adhesion. Most slicer software allows for changing the infill pattern to give the part structure in specific dimensions.

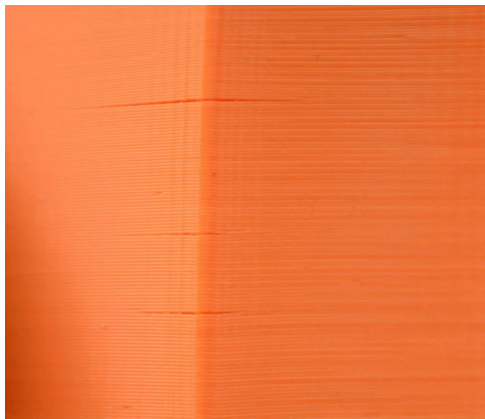


Figure B-14: 3D Printing Infill. (Reproduced as is from [83]).

**11. Wall thickness:**

Wall or shell thickness is the thickness of the outer wall of the part which determines print quality and structure. Wall thickness is something that can be changed in the slicer software and it is recommended that the user selects a size thickness that is at least two extruder widths across. This ensures that the outer geometry of the part actually gets printed as some slicer softwares disregard geometries that are smaller than the extruder diameter.

**12. Warping:**

Certain materials, like ABS, when 3D printing are very prone to warping. Warping happens when the part cools unevenly. The way to reduce warping is to enclose the print area and to have a heated build surface. After the print finishes, the user must allow the print to cool in a slow and controlled manner.

**13. Bed Adhesion:**

There are many different types of beds for 3D printing, the most common types are slightly textured material and glass plates. The best FDM build plates are heated top allow for better adhesion to the bed. Some tricks for better adhesion when the heated bed is not enough is to use hairspray or a glue stick for glass beds and blue painters tape for some of the textured surfaces.

**14. Tolerances:**

Tolerancing 3D prints is completely dependent on the printer involved. Bed leveling and initial layer height are also very important as the first layer can be oversized due to the squishing between the extruder and the build plate. In 3D printing communities, this term is known as “elephant’s foot”. For solving tolerance issues it is best to go onto a forum or other community of users of the same or similar printers to find what is best for the user’s 3D printer.

## Appendix C: Arduino Scripts

### Signal\_Conductor\_5.ino

```
/*
 * Signal_Conductor_5.ino - Code for controlling RC Car with a receiver or Raspberry Pi
 * Created by Mitchell Curbelo, March 16, 2019.
 *
 * Code for Remote Controller modeled by Circuit Basics
 * Link: http://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/
 */

// Include LED, Servo, and Remote Libraries
#include "LEDController.h"
#include <Servo.h>
#include <IRremote.h>

// Declare servos
Servo servo1; // Motor
Servo servo2; // Turning Servo

//Master comman (true - connect and write to servos)
// True - connect and write to servos
// False - don't connect and write to servos
boolean execute = true;

// Output and Input Signal Data -> {Pins, Signal Calibration (min, steady, max), measued/output value, signal step, lastvalue}
float motorReceiver[] = {22, 5.52, 9.355, 12.80, 9.355, 0.03, 9.355}; // Signal from the motor channel of the receiver
float turnReceiver[] = {23, 6.12, 9.245, 12.78, 9.25, 0.06, 9.25}; // Signal from the turning channel of the receiver
float motorTransmitter[] = {4, 55, 90, 125, 90, 2, 90}; // Signal to the motor
float turnTransmitter[] = {5, 47, 90, 133, 90, 2, 90}; // Signal to the turing servo

// Variables to prevent noise in the input signal from the receiver into steps (not currently in use)
// NumberOfSamples = The number of samples before resetting the samples array
// SwitchMinimum = The number of samples at the same value that need to be taken before updating the input value
// When the SwitchMinimum is meet, the samples array is reset
// Sample Count = Number of samples taken
// DegreeStep = the number of degrees between each step
// motorDegreeStep is constant but turnDegreeStep can be changed while running the program
// Samples = An array that contains the number of times each type of sample has been measured.
// Motor Variables
int motorNumberOfSamples = 2;
int motorSwitchMinimum = 1;
int motorSampleCount = 0;
float motorDegreeStep;
int * motorSamples;
// Turning Servo Variables
int turnNumberOfSamples = 2;
int turnSwitchMinimum = 1;
int turnSampleCount = 0;
float turnDegreeStep;
int * turnSamples;

long iterationStart = 0; // Set at the start of each loop to tracks the time to execute each iteration
int iterationsCounter = 0; // Iterations counter, sent to Raspberry Pi during training to determine if there are any skipped commands

// Variables for input from Raspberry Pi
String inputString = ""; // String read to from Raspberry Pi
bool stringComplete = false; // Boolean to determine if the entire instruction was received
int INPUT_SIZE = 2; // Number of inputs that are supposed to be recieved from the Raspberry

// Variables for the LEDs that indicate the pins, output types and output details
LEDController * ledController;
const int numberOfLEDS = 3; // Number of LEDs being controlled by the LEDController Library
const int ledPins[] = {24, 25, 26}; // Pins: {Yellow, Blue, Red}

// Set types of outputs that will be used by the LEDs
const int outputTypeZeroDetails[] = {1}; // Output Type: None, Details: {arraySize}
const int outputTypeOneDetails[] = {2, 1000}; // Output Type: Pulse, Details: {arraySize, highDelay}
const int outputTypeTwoDetails[] = {3, 200, 200}; // Output Type: Blink, Details: Details {arraySize, highDelay, lowDelay}
const int outputTypeFourDetails[] = {6, 3, 0, 300, 200, 1000}; // Output Type: Numbered Blink Before Delay and Repeat, Details:
{arraySize, blinkNumber, blinkCount, blinkNumberHighDelay, blinkNumberLowDelay, lowDelayBetweenNumberedBlinks}
```

```

// Variables controlled by the Remote Controller
int channel = 0; // Channel 0 = collect training data, Channel 1 = receive commands from Raspberry Pi, Channel 3 = set the number of
turning steps
int level = 0;
boolean play = false; // Run or stop the car
const int RECV_PIN = 27; // Pin that receives data from the remote controller
IRrecv irrecv(RECV_PIN);
decode_results results; // Decoded results from the remotecontroller
unsigned long lastKeyValue = 0; // Last value set from the remote controller
int turnSteps = 7; // Initial number of turning steps
int motorSteps = 7; // Initial number of motor speed steps

/*
 * Setup variables and libraries with initial values
 *
 * 1. Receiver input pins
 * 2. Initial Values
 * 3. Attach motor and turning servos
 * 4. Setup LEDs
 * 5. Setup Remote controller
 */
void setup() {
    // Open serial connection for communication with Raspberry Pi or debugging
    Serial.begin(9600);

    // Set both receiver input pins as inputs
    pinMode(motorReceiver[0], INPUT);
    pinMode(turnReceiver[0], INPUT);

    iterationsCounter = 0;

    // Calculate the initial degree steps
    motorDegreeStep = (float)(motorTransmitter[3] - motorTransmitter[1] + 1) / (float)motorSteps;
    turnDegreeStep = (float)(turnTransmitter[3] - turnTransmitter[1] + 1) / (float)turnSteps;

    // Initialize motor and turn sample arrays
    motorSamples = new int[motorNumberOfSamples];
    for (int counter = 0; counter < motorNumberOfSamples; counter++) {
        motorSamples[counter] = 0;
    }

    turnSamples = new int[17];
    for (int counter = 0; counter < 17; counter++) {
        turnSamples[counter] = 0;
    }

    // Initialize inputString
    inputString.reserve(200);

    Serial.println("Initialize");

    // Attach servos
    if (execute) {
        servo1.attach((int)(motorTransmitter[0])); // Motor
        servo2.attach((int)(turnTransmitter[0])); // Turn
    }

    // LED Setup
    {
        // Set Constructor Variables
        long lastChangeTime[] = {0, 0, 0}; // Set the last time each LEDs were set to zero
        int outputType[] = {0, 1, 1}; // Set the initial output types of the LEDs, Yellow = None, Blue = Pulse, Red = Pulse
        boolean state[] = {false, false, false}; // Set the initial state of the LEDs, true = on, false = off

        // Set Initial LED Output Details
        int *outputTypeDetails[] = {outputTypeZeroDetails, outputTypeOneDetails, outputTypeOneDetails};

        // Convert Constructor Variables to Pointers
        int *ledPinsPointer = ledPins;
        long *lastChangeTimePointer = lastChangeTime;
        int *ledOutputTypePointer = outputType;
        boolean *ledStatePointer = state;
        int **outputTypeDetailsPointer = outputTypeDetails;

        // Construct LEDController
        ledController = new LEDController(numberOfLEDs, ledPinsPointer, lastChangeTimePointer, ledOutputTypePointer, ledStatePointer,
outputTypeDetailsPointer);

```



```

    ledController->setLED(channel, HIGH); // Set the LED that indicates indicating the initial channel
}

// Setup Remote Control
irrecv.enableIRIn();
irrecv.blink13(true);
}

/*
 * Repeat the code within this function indefinitely
 */
* Loop Steps
* 1. Get the input from the remote controller
* 2. Run receiver code if channel is 0 and AI code if channel is 1
* 3. Update LEDs
* 4. Output ", (millis() - iterationStart), turnSteps\n" if channel != 2 or "(millis() - iterationStart), turnSteps\n" if channel == 2
*/
void loop() {
    iterationStart = millis(); // Note time at start of loop

    remoteControls(); // Get input from remote controller (< 0ms)

    // Determine channel
    if (channel == 0) {
        receiverControl(); // Get input from Receiver (~80ms)
    } else if (channel == 1) {
        AIControl(); // Get input from Raspberry Pi (~80ms)
    }

    ledController->updateLEDs(); // Update LEDs (< 0ms)

    // Serial.println(" Channel: " + String(channel) + " Level: " + String(level) + " Play: " + String(play)); // Debugging code, used to
    // determine the programs state
    if (channel != 2) {
        Serial.print(",");
    }

    // Print loop duration, the number of turning steps, and a '\n'
    Serial.println(String(millis() - iterationStart) + ", " + String(turnSteps));
}

// AI Functions

/*
 * Code for Raspberry Pi (channel = 1)
 * Reads input from the Raspberry Pi through a serial connection, sets the servo values, and sends the servo values back to the Raspberry
Pi
 *
 * If stringComplete is false then no input was read from the raspberry pi then don't set the servo values or sends values to the
Raspberry Pi
 * If execute is false then the commands from the Raspberry Pi are not send to the servos
 * If execute and play are true then the commands from the Raspberry Pi is send to the servos
 * If execute is true and play is false then the turning servo is set from the Raspberry Commands but the motor servo is not. The motor
servo is set to 90 (stop)
 *
 * Output (Received by Raspberry Pi or Debugger)
 * "motorTransmitter[4], turnTransmitter[4]"
 */
void AIControl() {
    serialEvent();
    if (stringComplete) {

        char input[inputString.length()];
        motorTransmitter[4] = (int) atoi(inputString.substring(0, 3).c_str()); //sets the value to be the integer form of the first 3
characters of the incoming string
        turnTransmitter[4] = (int) atoi(inputString.substring(4, 7).c_str()); //sets the value to be the integer form of the last 3 characters
of the incoming string

        //reset incoming string information
        inputString = "";
        stringComplete = false;

        if (execute) {
            servo2.write(turnTransmitter[4]);
            if (play) {

```

```

        servo1.write(motorTransmitter[4]);
    } else {
        servo1.write(90);
    }
}

// Send back the commands that were received
Serial.print(motorTransmitter[4]);
Serial.print(",");
Serial.print(turnTransmitter[4]);
}
}

/*
 * Get input from the Raspberry Pi Serial connection
 *
 * If the serial connection is not available then leave stringComplete = false
 * If the serial connection is available then leave read the sent characters into inputString
 * If no newline is read before the serial connection becomes unavailable then leave stringComplete = false
 * If a newline is read before the serial connection then leave set stringComplete equal to true
 */
void serialEvent() {
    while (Serial.available()) {
        // get the new byte:
        char inChar = (char)Serial.read();
        // add it to the inputString:
        if (inChar == '\n') {
            stringComplete = true;
            return;
        }

        inputString += inChar;
        // if the incoming character is a newline, set a flag so the main loop can
        // do something about it:]
    }
}

// Receiver Functions

/*
 * Code for Receiver (channel = 0)
 * Reads input from the receiver, sets the servo values, and sends the new servo values to the raspberry pi
 *
 * If execute or play are false then the servos are not set
 *
 * Output (Received by Raspberry Pi or Debugger)
 * If execute is true then the output "iterationsCounter, motorTransmitter[4] + "," + String(turnTransmitter[4])"
 * If execute is false then the output "motorTransmitter[4] + "," + String(turnTransmitter[4])"
 */
void receiverControl() {
    iterationsCounter++; // increment the number of iterations that have occurred

    // Get the input from the receiver
    getSignal(motorReceiver);
    getSignal(turnReceiver);

    // Convert the data from PWM to degrees
    scale(motorReceiver, motorTransmitter);
    scale(turnReceiver, turnTransmitter);

    // Remove noise from input signals
    signalNoiseFilter(motorTransmitter, motorDegreeStep, motorSteps, motorSwitchMinimum, motorNumberOfSamples, motorSamples,
&motorSampleCount);
    signalNoiseFilter(turnTransmitter, turnDegreeStep, turnSteps, turnSwitchMinimum, turnNumberOfSamples, turnSamples, &turnSampleCount);

    // Send the commands from the Receiver to the servos if execute and play are true
    // Print the iteration count
    if (execute) {
        if (play) {
            servo1.write((int)round(motorTransmitter[4])); // Motor
            servo2.write((int)round(turnTransmitter[4])); // Turn
        }

        Serial.print(iterationsCounter);
        Serial.print(",");
    }
}

```

```

// Print the commands sent to the servos (Received by Raspberry Pi or Debugger)
Serial.print(String(motorTransmitter[4]) + "," + String(turnTransmitter[4]));
}

/*
 * Reads the input from the Receiver using the getDutyCycle(float* input) function
 * Parameters
 * float* input = detailed information about the input signal: {Pins, Signal Calibration (min, steady, max), measured/output value, signal
step, lastvalue}
 *
 * If the input from the pin is not within the signal bounds specified in input (input[1] < x < input[3]) then the new input value
(input[4]) is set to the signals steady state value (input[3])
 * If the input from the pin is within the signal bounds specified in input (input[1] < x < input[3]) then the new input value (input[4])
is set to the measured duty cycle
 */
void getSignal(float* input) {
    float signalValue = getDutyCycle(input[0]);

    //Check Signal Bounds
    if (signalValue < 1) {
        signalValue = input[1];
    } else if (signalValue < input[1]) {
        signalValue = input[1];
    } else if (signalValue > input[3]) {
        signalValue = input[3];
    }

    input[4] = signalValue;
}

/*
 * Removes noise from the Receiver input
 * float* input = detailed information about the input signal: {Pins, Signal Calibration (min, steady, max), measured/output value, signal
step, lastvalue}
 * float degreeStep = the number of degrees between each step
 * int steps, int switchMinimum = The number of samples at the same value that need to be taken before updating the input value
 * int numberOfSamples = The number of samples before resetting the samples array
 * int* samples = An array that contains the number of times each type of sample has been measured.
 * int * sampleCount = Number of samples already taken
 *
 * Return
 * int the "new" input value (input[4])
 *
 * If the minimum number of samples has been met then the input (input[4]) is set to the new value, the last input value (input[6]) is set
to the new value, the sampleCount is set to 0, and the values in the samples array are set to 0
 * If the minimum number of samples has not been met the "new" input value (input[4]) is set to the last input value that was set
(input[6])
 * If the minimum number of samples has not been met and numberOfSamples >= sampleCount then values in the samples array are set to 0
 */
int signalNoiseFilter(float* input, float degreeStep, int steps, int switchMinimum, int numberOfSamples, int* samples, int * sampleCount)
{
    int steppedInput = (int)((float)(input[4] - input[1]) / (float)degreeStep); // Convert the input into a stepped input

    samples[steppedInput]++; // Increment samples at that input

    if (samples[steppedInput] == switchMinimum) {
        *sampleCount = numberOfSamples; // Ensure sampleCount is reset
        input[4] = (int)((float)steppedInput * ((float)(input[3] - input[1]) / (float)(steps - 1))) + input[1]; // Set new input value
        input[6] = input[4]; // Set last input value
    } else {
        input[4] = input[6]; // Set input value to last input value
    }

    (*sampleCount)++;

    // Reset values in samples array to zero if sampleCount is >= numberOfSamples
    if (*sampleCount >= numberOfSamples) {
        for (int counter = 0; counter < steps; counter++) {
            samples[counter] = 0;
            *sampleCount = 0;
        }
    }

    return input[4];
}

/*

```

```

* Get the the duty cycle of a signal from a pin by measuring the low time and high time
* Parameters
* int pin - the pin that the duty cycle is being read from
*
* Returned Value
* float dutyCycle = the duty cycle of the signal with the range of 0 to 100
*
* RunTime ~ 40ms (two duty cycles)
*
*/
float getDutyCycle(int pin) {
    long startTime = millis();
    float dutyCycle = 0;
    long timePeriod = 0;
    int low = 0;
    int high = 0;

    if (digitalRead(pin)) {
        low = pulseIn(pin, LOW, 25000);
        bool timeout = false;

        while (high == 0 && !timeout && ((millis() - startTime) < 100)) {
            if (!digitalRead(pin)) {
                high = pulseIn(pin, HIGH, 25000);
                timeout = true;
            }
        }
    } else {
        high = pulseIn(pin, HIGH, 25000);
        bool timeout = false;

        while (low == 0 && !timeout && ((millis() - startTime) < 100)) {
            if (digitalRead(pin)) {
                low = pulseIn(pin, LOW, 25000);
                timeout = true;
            }
        }
    }

    if (high != 0 && low != 0) {
        timePeriod = high + low;
        // Serial.print(String(timePeriod) + ",");
        dutyCycle = 100 * (float)high / (float)timePeriod;
    }

    if (dutyCycle <= 100) {
        return dutyCycle;
    } else {
        return 0;
    }
}

/*
* Scale the input from the input range to the ouput in the ouput range
* float* input = detailed information about the input signal: {Pins, Signal Calibration (min, steady, max), measued/output value, signal
step, lastvalue}
* float* output = detailed information about the output signal: {Pins, Signal Calibration (min, steady, max), measued/output value,
signal step, lastvalue}
* RunTime ~ 40ms
*/
void scale(float* input, float* output) {
    if (input[4] > input[2]) {
        double inputDelta = input[4] - input[2];
        double inputScale = input[3] - input[2];
        double inputRatio = inputDelta / inputScale;

        double outputScale = output[3] - output[2];

        output[4] = output[2] + (float)(outputScale * inputRatio);
    } else if (input[4] < input[2]) {
        double inputDelta = input[4] - input[2];
        double inputScale = input[2] - input[1];
        double inputRatio = inputDelta / inputScale;

        double outputScale = output[2] - output[1];

        output[4] = output[2] + (float)(outputScale * inputRatio);
    }
}

```

```

} else {
    output[4] = output[2];
}

if (output[4] > output[3] - output[5]) {
    output[4] = output[3];
} else if (output[4] < output[1] + output[5]) {
    output[4] = output[1];
} else if (((output[4] - output[5] <= output[2]) && (output[4] >= output[2])) || ((output[4] + output[5] >= output[2]) && (output[4] <=
output[2]))) {
    output[4] = output[2];
}
}

// Remote Controller Functions

/*
* Get signal from remote controller and set channel, level, and play
* Button actions...
* CH+: Increment channel or wrap to channel 0 if channel = 2, update the LED output types and details , set play = false, and set level =
0
* CH-: Decrement channel or wrap to channel 2 if channel = 0, update the LED output types and details , set play = false, and set level =
0
* >| : play = !play, update the LED output types and details
* + : Increment level, if channel = 3 then turnSteps += 2, update LED output types, and details
* - : Decrement level, if channel = 3 then turnSteps -= 2, update LED output types, and details
*/
void remoteControls() {
    if (irrecv.decode(&results)) {
        if (results.value == 0xFFFFFFFF) {
            results.value = lastKeyValue;
        } else {
            switch (results.value) {
                case 0xFFA25D: // CH-

                    if (channel == 0) {
                        channel = 2;
                    } else {
                        channel--;
                    }
                }

                for (int counter = 0; counter < numberOfLEDs; counter++) {
                    ledController->setOutputType(counter, 0, outputTypeOneDetails);
                    if (counter != channel) {
                        ledController->setOutputType(counter, 0, outputTypeOneDetails);
                        ledController->setLED(counter, LOW);
                    } else {
                        ledController->setOutputType(counter, 0, outputTypeZeroDetails);
                        ledController->setLED(counter, HIGH);
                    }
                }

                level = 0;
                play = false;
                break;
            case 0xFF629D: // CH

                break;
            case 0xFFE21D: // CH+

                if (channel == 2) {
                    channel = 0;
                } else {
                    channel++;
                }

                // Update LED output types and led output details
                for (int counter = 0; counter < numberOfLEDs; counter++) {
                    ledController->setOutputType(counter, 0, outputTypeOneDetails);
                    if (counter != channel) {
                        ledController->setOutputType(counter, 0, outputTypeOneDetails);
                        ledController->setLED(counter, LOW);
                    } else {
                        ledController->setOutputType(counter, 0, outputTypeZeroDetails);
                        ledController->setLED(counter, HIGH);
                    }
                }
            }
        }
    }
}

```

```

level = 0;
play = false;

break;
case 0xFF22DD: // >>|

break;
case 0xFF02FD: // >>|

break ;
case 0xFFC23D: // >|
play = !play;
if (play) {
if (channel != 2) {
ledController->setOutputType(channel, 2, outputTypeTwoDetails);
} else {
int * outputTypeDetails = new int[outputTypeFourDetails[0]];
ledController->copyArray(outputTypeFourDetails, outputTypeDetails, outputTypeFourDetails[0]);
outputTypeDetails[1] = (int)(turnSteps / 2);
ledController->setOutputType(channel, 4, outputTypeDetails);
}
} else {
ledController->setOutputType(channel, 0, outputTypeZeroDetails);
ledController->setLED(channel, HIGH);
}
break ;
case 0xFFE01F: // -
if (channel != 2) {
if (level != 0) {
level--;
}
} else {
if (play) {
if (turnSteps != 3) {
turnSteps -= 2;
turnDegreeStep = (float)(turnTransmitter[3] - turnTransmitter[1] + 1) / (float)turnSteps;
int * outputTypeDetails = new int[outputTypeFourDetails[0]];
ledController->copyArray(outputTypeFourDetails, outputTypeDetails, outputTypeFourDetails[0]);
outputTypeDetails[1] = (int)(turnSteps / 2);
ledController->setOutputType(channel, 4, outputTypeDetails);
}
}
}
break ;
case 0xFFA857: // +
if (channel != 2) {
level++;
} else {
if (play) {
if (turnSteps != 17) {
turnSteps += 2;
turnDegreeStep = (float)(turnTransmitter[3] - turnTransmitter[1] + 1) / (float)turnSteps;
int * outputTypeDetails = new int[outputTypeFourDetails[0]];
ledController->copyArray(outputTypeFourDetails, outputTypeDetails, outputTypeFourDetails[0]);
outputTypeDetails[1] = (int)(turnSteps / 2);
ledController->setOutputType(channel, 4, outputTypeDetails);
}
}
}
break ;
case 0xFF906F: // EQ

break ;
case 0xFF6897: // 0

break ;
case 0xFF9867: // 100+

break ;
case 0xFFB04F: // 200+

break ;
case 0xFF30CF: // 1

break ;
case 0xFF18E7: // 2

```

```

        break ;
    case 0xFF7A85: // 3

        break ;
    case 0xFF10EF: // 4

        break ;
    case 0xFF38C7: // 5

        break ;
    case 0xFF5AA5: // 6

        break ;
    case 0xFF42BD: // 7

        break ;
    case 0xFF4AB5: // 8

        break ;
    case 0xFF52AD: // 9

        break ;
    }

    // Serial.println(" Channel: " + String(channel) + " Level: " + String(level) + " Play: " + String(play));
}

lastKeyValue = results.value;
irrecv.resume();
}
}
}

```

## LEDController.h

```

/*
 * LEDController.h - Library for controlling LED outputs.
 * Created by Mitchell Curbelo, March 16, 2019.
 *
 * Variables...
 * int _numberOfLEDs; // Number of LEDs being controlled
 * int * _pins; // Array of LED pins
 * long * _lastChangeTimes; // Last time the LEDs were updated
 * int * _outputTypes; // Array of LED output types
 * boolean * _states; // Array of current state of LEDs
 * int ** _outputTypeDetails; // Array of LED output details
 *
 * Output Types...
 * 0: None, Details: {arraySize}
 * 1: Pulse, Details: {arraySize, highDelay};
 * 2: Blink, Details: {arraySize, highDelay, lowDelay}
 * 3: Blink with Delay Cycle, Details: {arraySize, startIndex,
 * delayOne, delayTwo, delayThree, ...}
 * 4: Numbered Blink Before Delay and Repeat, Details: {arraySize, blinkNumber,
 * blinkCount, blinkNumberHighDelay, blinkNumberLowDelay,
 * lowDelayBetweenNumberedBlinks}
 * Else: Print error
 */

#ifndef LEDController_h
#define LEDController_h

#include "Arduino.h"

class LEDController {
public:
    LEDController(int numberOfLEDs, int * pins, long * lastChangeTimes, int * outputTypes, boolean * states, int ** theOutputTypeDetails);
    void updateLEDs();
    void setLED(int pinIndex, boolean state);

    boolean* getStates();
    int getOutputType(int index);
    void setOutputType(int index, int value, int * theOutputTypeDetails);

    void copyArray(int* src, int* dst, int len);
}

```

```

void copyArray(long* src, long* dst, int len);
void copyArray(boolean* src, boolean* dst, int len);

void printLEDStates();
void printLEDs();
private:
int _numberOfLEDs;
int * _pins;
long * _lastChangeTimes;
int * _outputTypes;
boolean * _states;
int ** _outputTypeDetails;

void setupLEDPins();

String LEDController::stringAppendSpaces(String theString, int theLength, char theChar);
String LEDController::stringReplaceBeginning(String originalString, String replacingString);
};

#endif

```

## LEDController.c

```

#include "Arduino.h"
#include "LEDController.h"

/*
 * Constructor
 * int numberOfLEDs = Number of LEDs being controlled
 * int * pins = Array of LED pins
 * long * lastChangeTimes = Last time the LEDs were updated
 * int * outputTypes = Array of LED output types
 * boolean * states = Array of current state of LEDs
 * int ** theOutputTypeDetails = Array of LED output details
 */
LEDController::LEDController(int numberOfLEDs, int * pins, long * lastChangeTimes, int * outputTypes, boolean * states, int **
theOutputTypeDetails) {
    _numberOfLEDs = numberOfLEDs;

    // Create arrays
    _pins = new int[numberOfLEDs];
    _lastChangeTimes = new long[numberOfLEDs];
    _outputTypes = new int[numberOfLEDs];
    _states = new boolean[numberOfLEDs];
    _outputTypeDetails = new int*[numberOfLEDs];

    //Copy Arrays
    copyArray(pins, _pins, numberOfLEDs);
    copyArray(lastChangeTimes, _lastChangeTimes, numberOfLEDs);
    copyArray(outputTypes, _outputTypes, numberOfLEDs);
    copyArray(states, _states, numberOfLEDs);

    for (int counter = 0; counter < numberOfLEDs; counter++) {
        _outputTypeDetails[counter] = new int[theOutputTypeDetails[counter][0]];
        copyArray(theOutputTypeDetails[counter], _outputTypeDetails[counter], theOutputTypeDetails[counter][0]);
    }

    //Setup LEDs
    setupLEDPins();
}

/*
 * Update LEDs based on the last time they were updated, their output types and their details
 * Output Types...
 * 0: None, Details: {arraySize}
 * 1: Pulse, Details: {arraySize, highDelay};
 * 2: Blink, Details: {arraySize, highDelay, lowDelay}
 * 3: Blink with Delay Cycle, Details: {arraySize, startIndex,
 * delayOne, delayTwo, delayThree, ...}
 * 4: Numbered Blink Before Delay and Repeat, Details: {arraySize, blinkNumber, blinkCount, blinkNumberHighDelay,
 * blinkNumberLowDelay, lowDelayBetweenNumberedBlinks}
 * Else: Print error
 */
void LEDController::updateLEDs() {
    for (int counter = 0; counter < _numberOfLEDs; counter++) {

```



```

// Update LEDs based on their output type
switch (_outputTypes[counter]) {
  case 0: // None - Details {arraySize};
    break;
  case 1: // // Pulse - Details {arraySize, highDelay};
    if (_states[counter]) {
      if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][1]) {
        setLED(counter, LOW);
        _lastChangeTimes[counter] = millis();
      }
    }
    break;
  case 2: // Blink - Details {arraySize, highDelay, lowDelay};
    if (_states[counter]) {
      if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][1]) {
        setLED(counter, LOW);
        _lastChangeTimes[counter] = millis();
      }
    } else {
      if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][2]) {
        setLED(counter, HIGH);
        _lastChangeTimes[counter] = millis();
      }
    }
    break;
  case 3: // Blink with Delay Cycle - Details {arraySize, startIndex, delayOne, delayTwo, delayThree, ...};
    if (_outputTypeDetails[counter][1] < 2) {
      _outputTypeDetails[counter][1] = 2;
    }

    //      Serial.println(_outputTypeDetails[counter][_outputTypeDetails[counter][1]]);

    if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][_outputTypeDetails[counter][1]]) {
      setLED(counter, !_states[counter]);

      _lastChangeTimes[counter] = millis();

      if (_outputTypeDetails[counter][1] + 1 >= _outputTypeDetails[counter][0]) {
        _outputTypeDetails[counter][1] = 2;
      } else {
        _outputTypeDetails[counter][1] = _outputTypeDetails[counter][1] + 1;
      }
    }
    break;
  case 4: // Numbered Blink Before Delay and Repeat - Details {arraySize, blinkNumber, blinkCount, blinkNumberHighDelay,
    // blinkNumberLowDelay, lowDelayBetweenNumberedBlinks}
    if (_outputTypeDetails[counter][2] < 0) {
      _outputTypeDetails[counter][2] = 0;
    }

    if (_outputTypeDetails[counter][2] < _outputTypeDetails[counter][1]) {
      if (_states[counter]) {
        if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][3]) {
          setLED(counter, LOW);
          _lastChangeTimes[counter] = millis();
        }
      } else {
        if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][4]) {
          _outputTypeDetails[counter][2]++;
          _lastChangeTimes[counter] = millis();

          if (_outputTypeDetails[counter][2] == _outputTypeDetails[counter][1]) {
            setLED(counter, LOW);
          } else {
            setLED(counter, HIGH);
          }
        }
      }
    }
    } else {
      if (millis() - _lastChangeTimes[counter] > _outputTypeDetails[counter][5]) {
        setLED(counter, HIGH);
        _outputTypeDetails[counter][2] = 0;
      }
    }
    break;
  default: // Print error for unknown case

```

```

        Serial.println("Error -> Invalid LED update case for LED: " + String(counter) + " with case: " + _outputTypes[counter]);
        break;
    }
}

/*
 * Set the state of an LED at a specified index and updates the last change time of the LED
 * int pinIndex = index of the LED being changed
 * boolean state = the new state of the LED
 */
void LEDController::setLED(int pinIndex, boolean state) {
    digitalWrite(_pins[pinIndex], state);
    _states[pinIndex] = state;
    _lastChangeTimes[pinIndex] = millis();
}

/*
 * Set all the pinModes of the pins specified in the pins array OUTPUT
 */
void LEDController::setupLEDPins() {
    for (int counter = 0; counter < _numberOfLEDs; counter++) {
        pinMode(_pins[counter], OUTPUT);
    }
}

/*
 * Copy a source int array to a destination int array
 * int* src = source boolean array
 * int* dst = destination int array
 * int len = length of src and dst arrays
 */
void LEDController::copyArray(int* src, int* dst, int len) {
    for (int counter = 0; counter < len; counter++) {
        memcpy(dst++, src++, sizeof(src[0]));
    }
}

/*
 * Copy a source long array to a destination long array
 * long* src = source long array
 * long* dst = destination long array
 * int len = length of src and dst arrays
 */
void LEDController::copyArray(long* src, long* dst, int len) {
    for (int counter = 0; counter < len; counter++) {
        memcpy(dst++, src++, sizeof(src[0]));
    }
}

/*
 * Copy a boolean array from a source array to a destination array
 * boolean* src = source boolean array
 * boolean* dst = destination boolean array
 * int len = length of src and dst arrays
 */
void LEDController::copyArray(boolean* src, boolean* dst, int len) {
    for (int counter = 0; counter < len; counter++) {
        memcpy(dst++, src++, sizeof(src[0]));
    }
}

/*
 * Print LED information
 * Format...
 * LEDController:
 * Pin   Type   State   Last Change   Type Details
 *
 *           Tabulated Values
 */
void LEDController::printLEDs() {
    String lineSpacer = "\n";
    String outputString = "LEDController:" + lineSpacer;

    outputString += stringAppendSpaces("Pin", 6, ' ');
}

```

```

outputString += stringAppendSpaces("Type", 6, ' ');
outputString += stringAppendSpaces("State", 7, ' ');
outputString += stringAppendSpaces("Last Change", 13, ' ');
outputString += stringAppendSpaces("Type Details", 20, ' ');
outputString += lineSpacer;

for (int counter = 0; counter < _numberOfLEDS; counter++) {
    outputString += stringAppendSpaces(String(_pins[counter]), 6, ' ');
    outputString += stringAppendSpaces(String(_outputTypes[counter]), 6, ' ');
    outputString += stringAppendSpaces(String(_states[counter]), 7, ' ');
    outputString += stringAppendSpaces(String(_lastChangeTimes[counter]), 13, ' ');

    String detailsString = "{";
    for (int detailsCounter = 0; detailsCounter < _outputTypeDetails[counter][0]; detailsCounter++) {
        if (detailsCounter == 0) {
            detailsString += _outputTypeDetails[counter][detailsCounter];
        } else {
            detailsString += "," + String(_outputTypeDetails[counter][detailsCounter]);
        }
    }
    detailsString += "}";
    outputString += stringAppendSpaces(detailsString, 20, ' ');

    outputString += lineSpacer;
}

Serial.println(outputString);
}

/*
 * Add a char to the end of the string to get a string with a specific length
 * String theString = the string with the chars being added onto the end
 * int theLength = length of the returned string
 * char theChar = character being added to the end of theString
 */
String LEDController::stringAppendSpaces(String theString, int theLength, char theChar) {
    String originalString = "";
    for (int counter = 0; counter < theLength; counter++) {
        originalString += theChar;
    }

    return stringReplaceBeginning(originalString, theString);
}

/*
 * Replaces the beginning part of a string with another string
 *
 * Parameters
 * String originalString = The string that is having the beginning replaced
 * String replacingString = the string that is replacing the beginning of originalString
 *
 * Exceptions
 * replacingString Length < replacingString length
 */
String LEDController::stringReplaceBeginning(String originalString, String replacingString) {
    String returnedString = "";

    if (replacingString.length() < originalString.length()) {
        returnedString = replacingString + originalString.substring(replacingString.length(), originalString.length());
    } else {
        Serial.println("Error -> LEDController::stringReplaceBeginning(String originalString, String replacingString);
replacingString.length()=" + String(replacingString.length()) + " > originalString.length()=" + String(originalString.length()));
    }

    return returnedString;
}

/*
 * Returns the states of the LEDs
 */
boolean* LEDController::getStates() {
    return _states;
}

/*
 * Returns the output type of the LED at the specified index
 */

```

```

int LEDController::getOutputType(int index) {
    return _outputTypes[index];
}

/*
 * Change the output type and output details of an LED
 * index - index of LED in arrays
 * value - the new output type of the LED
 * theOutputTypeDetails - the new outputDetails of the LED
 */
void LEDController::setOutputType(int index, int value, int * theOutputTypeDetails) {
    _outputTypes[index] = value;
    _outputTypeDetails[index] = new int[theOutputTypeDetails[0]];
    copyArray(theOutputTypeDetails, _outputTypeDetails[index], theOutputTypeDetails[0]);
}

/*
 * Print the current states of the LEDs
 */
void LEDController::printLEDStates() {
    for (int counter = 0; counter < _numberOfLEDs; counter++) {
        Serial.print(_states[counter]);
        if (counter != _numberOfLEDs - 1) {
            Serial.print(",");
        } else {
            Serial.println();
        }
    }
}
}

```

## Appendix C: Bond Graph State Equations and Derivations

### Gearbox

$$\begin{aligned}
 \tau_2 &= \tau_R = \tau_1 \\
 \tau_3 &= D_1 \omega_3 = \left( \frac{D_1 N_A}{N_B J_3} \right) h_3 \\
 \tau_5 + \tau_4 &= J_1 \left( \frac{d}{dt} (\omega_3) \right) + J_2 \left( \frac{d}{dt} (\omega_3) \right) = \left( \frac{d}{dt} (\omega_3) \right) (J_1 + J_2) = \left( \frac{N_A}{N_B} \right) \left( \frac{d}{dt} (\omega_4) \right) (J_1 + J_2) = \left( \frac{N_A}{N_B} \right) \left( \frac{h'_3}{J_3} \right) (J_1 + J_2) = \left( \frac{N_A}{N_B} \right) \left( \frac{J_1 + J_2}{J_3} \right) h'_3 \\
 \tau_6 &= \tau_5 + \tau_4 + \tau_3 + \tau_2 = \left( \frac{N_A}{N_B} \right) \left( \frac{J_1 + J_2}{J_3} \right) h'_3 + \left( \frac{D_1 N_A}{N_B J_3} \right) h_3 + \tau_R \\
 \tau_7 &= \left( \frac{N_B}{N_A} \right) \tau_6 = \left( \frac{N_B}{N_A} \right) \left( \left( \frac{N_A}{N_B} \right) \left( \frac{J_1 + J_2}{J_3} \right) h'_3 + \left( \frac{N_A}{N_B} \right) \left( \frac{D_1}{J_3} \right) h_3 + \tau_R \right) = \left( \frac{J_1 + J_2}{J_3} \right) h'_3 + \left( \frac{D_1}{J_3} \right) h_3 + \left( \frac{N_B}{N_A} \right) \tau_R \\
 \tau_8 &= D_4 \omega_4 = D_4 \frac{h_3}{J_3} = \frac{D_4}{J_3} h_3 \\
 \tau_9 &= k_4 \theta_4 \\
 \tau_{11} + \tau_{10} &= J_5 \left( \frac{d}{dt} (\omega_5) \right) + J_4 \left( \frac{d}{dt} (\omega_5) \right) = \left( \frac{d}{dt} (\omega_5) \right) (J_5 + J_4) = \left( \frac{N_D}{N_C} \right) \left( \frac{d}{dt} (\omega_6) \right) (J_5 + J_4) = \left( \frac{N_D}{N_C} \right) \left( \frac{d}{dt} \left( \frac{h_6}{J_6} \right) \right) (J_5 + J_4) = \left( \frac{N_D h'_6}{N_C J_6} \right) (J_5 + J_4) \\
 \tau_{13} &= \left( \frac{N_C}{N_D} \right) \tau_{12} = \left( \frac{N_C}{N_D} \right) (\tau_{11} + \tau_{10} + \tau_9) = \left( \frac{N_C}{N_D} \right) \left( \left( \frac{N_D h'_6}{N_C J_6} \right) (J_5 + J_4) + k_4 \theta_4 \right) = \left( \frac{J_5 + J_4}{J_6} \right) (h'_6) + \left( \frac{N_C}{N_D} \right) (k_4 \theta_4) \\
 \tau_{14} &= k_7 \theta_7 \\
 \tau_{15} &= D_7 \omega_M \\
 \omega_3 &= \left( \frac{N_B}{N_A} \right) \omega_4 = \left( \frac{N_A}{N_B J_3} \right) h_3 \\
 \omega_4 &= \frac{h_3}{J_3} \\
 \omega_5 &= \left( \frac{N_C}{N_D} \right) \omega_6 = \left( \frac{N_C}{N_D J_6} \right) h_6 \\
 \omega_6 &= \frac{h_6}{J_6} \\
 \omega_7 &= \omega_M \\
 \theta'_7 &= \omega_7 - \omega_6 = \omega_M - \frac{h_6}{J_6} \\
 h'_6 &= \tau_{14} - \tau_{13} = k_7 \theta_7 - \left( \frac{J_5 + J_4}{J_6} \right) (h'_6) - \left( \frac{N_C}{N_D} \right) (k_4 \theta_4) = \frac{k_7 \theta_7 - \left( \frac{N_C}{N_D} \right) (k_4 \theta_4)}{\left( \frac{J_5 + J_4}{J_6} \right) + 1} \\
 \theta'_4 &= \omega_5 - \omega_4 = \left( \frac{N_C}{N_D J_6} \right) h_6 - \frac{h_3}{J_3} \\
 h'_3 &= \tau_9 - \tau_8 - \tau_7 = k_4 \theta_4 - \frac{D_4}{J_3} h_3 - \left( \frac{J_1 + J_2}{J_3} \right) h'_3 - \left( \frac{D_1}{J_3} \right) h_3 - \left( \frac{N_B}{N_A} \right) \tau_R = \frac{k_4 \theta_4 - \frac{D_4}{J_3} h_3 - \left( \frac{D_1}{J_3} \right) h_3 - \left( \frac{N_B}{N_A} \right) \tau_R}{\left( \frac{J_1 + J_2}{J_3} \right) + 1}
 \end{aligned}$$

## Suspension

$$\begin{aligned}
x'_1 = v_5 = \frac{F_2}{D_1} &= (D_1)^{-1}(F_1 - F_3) = (D_1)^{-1}(F_4 - k_1x_1) = (D_1)^{-1}(F_5 + F_6 - k_1x_1) = (D_1)^{-1}(k_2x_{2a} + D_2v_6 - k_1x_1) \\
&= (D_1)^{-1}(k_2x_{2a} + D_2(v_1 - v_{2a}) - k_1x_1) = (D_1)^{-1}\left(k_2x_{2a} + D_2\left((v_{N1} - v_5) - ((v_{2a} - v_2) + v_2)\right) - k_1x_1\right) \\
&= (D_1)^{-1}\left(k_2x_{2a} + D_2\left((v_{N1} - x'_1) - \left(-\omega_2R_a + \frac{p_2}{m_2}\right)\right) - k_1x_1\right) \\
&= (D_1)^{-1}\left(k_2x_{2a} + D_2\left((v_{N1} - x'_1) + \omega_2R_a - \frac{p_2}{m_2}\right) - k_1x_1\right) \\
&= (D_1)^{-1}\left(k_2x_{2a} + D_2\left((v_{N1} - x'_1) + \frac{h_2}{J_2}R_a - \frac{p_2}{m_2}\right) - k_1x_1\right) \\
&= (D_1)^{-1}\left(k_2x_{2a} + D_2v_{N1} - D_2x'_1 + D_2\frac{h_2}{J_2}R_a - \frac{D_2p_2}{m_2} - k_1x_1\right) \\
&= \left(\frac{k_2}{D_1}x_{2a} + \frac{D_2}{D_1}v_{N1} - \frac{D_2}{D_1}x'_1 + \frac{D_2R_a}{D_1J_2}h_2 - \frac{D_2}{m_2}p_2 - \frac{k_1}{D_1}x_1\right) = \frac{\left(\frac{k_2}{D_1}x_{2a} + \frac{D_2}{D_1}v_{N1} + \frac{D_2R_a}{D_1J_2}h_2 - \frac{D_2}{m_2}p_2 - \frac{k_1}{D_1}x_1\right)}{\left(1 + \frac{D_2}{D_1}\right)}
\end{aligned}$$

$$\begin{aligned}
x'_{2a} = v_6 = \frac{F_6}{D_2} &= (D_2)^{-1}(F_4 - F_5) = (D_2)^{-1}(F_1 - k_2x_{2a}) = (D_2)^{-1}((F_2 + F_3) - k_2x_{2a}) = (D_2)^{-1}((D_1v_5 + k_1x_1) - k_2x_{2a}) \\
&= (D_2)^{-1}(D_1(v_{N1} - v_1) + k_1x_1 - k_2x_{2a}) = (D_2)^{-1}\left(D_1(v_{N1} - (v_6 + v_{2a})) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} - \left(x'_{2a} + ((v_{2a} - v_2) + v_2)\right)\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} - \left(x'_{2a} + \left(-\omega_2R_a + \frac{p_2}{m_2}\right)\right)\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} - \left(x'_{2a} + \left(-\frac{h_2}{J_2}R_a + \frac{p_2}{m_2}\right)\right)\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} - \left(x'_{2a} - \frac{h_2}{J_2}R_a + \frac{p_2}{m_2}\right)\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} + \left(x'_{2a} + \frac{h_2}{J_2}R_a - \frac{p_2}{m_2}\right)\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1\left(v_{N1} + x'_{2a} + \frac{h_2}{J_2}R_a - \frac{p_2}{m_2}\right) + k_1x_1 - k_2x_{2a}\right) \\
&= (D_2)^{-1}\left(D_1v_{N1} + D_1x'_{2a} + \frac{D_1h_2}{J_2}R_a - \frac{D_1p_2}{m_2} + k_1x_1 - k_2x_{2a}\right) \\
&= \left(\frac{D_1}{D_2}v_{N1} + \frac{D_1}{D_2}x'_{2a} + \frac{D_1R_a}{D_2J_2}h_2 - \frac{D_1}{D_2m_2}p_2 + \frac{k_1}{D_2}x_1 - \frac{k_2}{D_2}x_{2a}\right) = \frac{\left(\frac{D_1}{D_2}v_{N1} + \frac{D_1R_a}{D_2J_2}h_2 - \frac{D_1}{D_2m_2}p_2 + \frac{k_1}{D_2}x_1 - \frac{k_2}{D_2}x_{2a}\right)}{\left(1 - \frac{D_1}{D_2}\right)}
\end{aligned}$$

$$\begin{aligned}
x'_3 = v_8 = \frac{F_{16}}{D_4} &= (D_4)^{-1}(F_{14} - F_{15}) = (D_4)^{-1}(F_{11} - k_3x_3) = (D_4)^{-1}(F_{12} + F_{13} - k_3x_3) = (D_4)^{-1}(k_3x_{2b} + D_3v_7 - k_3x_3) \\
&= (D_4)^{-1}(k_3x_{2b} + D_3(v_3 - v_{2b}) - k_3x_3) = (D_4)^{-1}\left(k_3x_{2b} + D_2\left((v_{N2} - v_8) - ((v_{2b} - v_2) + v_2)\right) - k_3x_3\right) \\
&= (D_4)^{-1}\left(k_3x_{2b} + D_3\left((v_{N2} - x'_3) - \left(\omega_2R_b + \frac{p_2}{m_2}\right)\right) - k_3x_3\right) \\
&= (D_4)^{-1}\left(k_3x_{2b} + D_3\left((v_{N2} - x'_3) - \omega_2R_b - \frac{p_2}{m_2}\right) - k_3x_3\right) \\
&= (D_4)^{-1}\left(k_3x_{2b} + D_3\left((v_{N2} - x'_3) - \frac{h_2}{J_2}R_b - \frac{p_2}{m_2}\right) - k_3x_3\right) \\
&= (D_4)^{-1}\left(k_3x_{2b} + D_3v_{N2} - D_2x'_3 - D_3\frac{h_2}{J_2}R_b - \frac{D_3p_2}{m_2} - k_3x_3\right) \\
&= \left(\frac{k_3}{D_4}x_{2b} + \frac{D_3}{D_4}v_{N2} - \frac{D_3}{D_4}x'_3 - \frac{D_3R_b}{D_4J_2}h_2 - \frac{D_3}{m_2}p_2 - \frac{k_3}{D_4}x_3\right) = \frac{\left(\frac{k_3}{D_4}x_{2b} + \frac{D_3}{D_4}v_{N2} - \frac{D_3R_b}{D_4J_2}h_2 - \frac{D_3}{m_2}p_2 - \frac{k_3}{D_4}x_3\right)}{\left(1 + \frac{D_3}{D_4}\right)}
\end{aligned}$$

$$\begin{aligned}
x'_{2b} = v_7 &= \frac{F_{13}}{D_3} = (D_3)^{-1}(F_{11} - F_{12}) = (D_3)^{-1}(F_{14} - k_3 x_{2b}) = (D_3)^{-1}((F_{16} + F_{15}) - k_3 x_{2b}) = (D_3)^{-1}((D_4 v_6 + k_4 x_3) - k_3 x_{2b}) \\
&= (D_3)^{-1}(D_4(v_{N2} - v_3) + k_4 x_3 - k_3 x_{2b}) = (D_3)^{-1}(D_4(v_{N2} - (v_7 + v_{2b})) + k_4 x_3 - k_3 x_{2b}) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} - \left(x'_{2b} + ((v_{2b} - v_2) + v_2)\right)\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} - \left(x'_{2b} + \left(\omega_2 R_b + \frac{p_2}{m_2}\right)\right)\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} - \left(x'_{2b} + \left(\frac{h_2}{J_2} R_b + \frac{p_2}{m_2}\right)\right)\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} - \left(x'_{2b} + \frac{h_2}{J_2} R_b + \frac{p_2}{m_2}\right)\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} + \left(x'_{2b} - \frac{h_2}{J_2} R_b - \frac{p_2}{m_2}\right)\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4\left(v_{N2} + x'_{2b} - \frac{h_2}{J_2} R_b - \frac{p_2}{m_2}\right) + k_4 x_3 - k_3 x_{2b}\right) \\
&= (D_3)^{-1}\left(D_4 v_{N2} + D_4 x'_{2b} - \frac{D_4 h_2}{J_2} R_b - \frac{D_4 p_2}{m_2} + k_4 x_3 - k_3 x_{2b}\right) \\
&= \left(\frac{D_4}{D_3} v_{N2} + \frac{D_4}{D_3} x'_{2b} - \frac{D_4 R_b}{D_3 J_2} h_2 - \frac{D_4}{D_3 m_2} p_2 + \frac{k_4}{D_3} x_3 - \frac{k_3}{D_3} x_{2b}\right) = \frac{\left(\frac{D_4}{D_3} v_{N2} - \frac{D_4 R_b}{D_3 J_2} h_2 - \frac{D_4}{D_3 m_2} p_2 + \frac{k_4}{D_3} x_3 - \frac{k_3}{D_3} x_{2b}\right)}{\left(1 - \frac{D_4}{D_3}\right)}
\end{aligned}$$

$$\begin{aligned}
p'_2 = F_9 + F_7 + F_{10} &= F_9 + F_4 + F_{11} = F_9 + (F_5 + F_6) + (F_{12} + F_{13}) = F_9 + (x_{2a} k_2 + D_2 v_6) + (x_{2b} k_2 + D_3 v_7) \\
&= F_9 + (x_{2a} k_2 + D_2 x'_{2a}) + (x_{2b} k_2 + D_3 x'_{2b}) \\
&= F_9 + \left(x_{2a} k_2 + D_2 \left(\frac{\left(\frac{D_1}{D_2} v_{N1} + \frac{D_1 R_a}{D_2 J_2} h_2 - \frac{D_1}{D_2 m_2} p_2 + \frac{k_1}{D_2} x_1 - \frac{k_2}{D_2} x_{2a}\right)}{\left(1 - \frac{D_1}{D_2}\right)}\right)\right) \\
&+ \left(x_{2b} k_2 + D_3 \left(\frac{\left(\frac{D_4}{D_3} v_{N2} - \frac{D_4 R_b}{D_3 J_2} h_2 - \frac{D_4}{D_3 m_2} p_2 + \frac{k_4}{D_3} x_3 - \frac{k_3}{D_3} x_{2b}\right)}{\left(1 - \frac{D_4}{D_3}\right)}\right)\right) \\
&= F_9 + \left(x_{2a} k_2 + \left(\frac{\left(D_1 v_{N1} + \frac{D_1 R_a}{J_2} h_2 - \frac{D_1}{m_2} p_2 + k_1 x_1 - k_2 x_{2a}\right)}{\left(1 - \frac{D_1}{D_2}\right)}\right)\right) \\
&+ \left(x_{2b} k_2 + \left(\frac{\left(D_4 v_{N2} - \frac{D_4 R_b}{J_2} h_2 - \frac{D_4}{m_2} p_2 + k_4 x_3 - k_3 x_{2b}\right)}{\left(1 - \frac{D_4}{D_3}\right)}\right)\right) \\
&= F_9 + x_{2a} k_2 + \frac{\left(D_1 v_{N1} + \frac{D_1 R_a}{J_2} h_2 - \frac{D_1}{m_2} p_2 + k_1 x_1 - k_2 x_{2a}\right)}{\left(1 - \frac{D_1}{D_2}\right)} + x_{2b} k_2 + \frac{\left(D_4 v_{N2} - \frac{D_4 R_b}{J_2} h_2 - \frac{D_4}{m_2} p_2 + k_4 x_3 - k_3 x_{2b}\right)}{\left(1 - \frac{D_4}{D_3}\right)}
\end{aligned}$$

$$\begin{aligned}
h'_2 = \tau_{2a} + \tau_{2b} &= \frac{F_9}{R_b} - \frac{F_8}{R_a} = (R_b)^{-1}(F_{10}) - (R_a)^{-1}(F_7) = (R_b)^{-1}(F_{11}) - (R_a)^{-1}(F_4) = (R_b)^{-1}(F_{12} + F_{13}) - (R_a)^{-1}(F_5 + F_6) \\
&= (R_b)^{-1}(x_{2b}k_2 + D_3v_7) - (R_a)^{-1}(x_{2a}k_2 + D_2v_6) = (R_b)^{-1}(x_{2b}k_2 + D_3x'_{2b}) - (R_a)^{-1}(x_{2a}k_2 + D_2x'_{2a}) \\
&= (R_b)^{-1} \left( x_{2b}k_2 + D_3 \left( \frac{\left( \frac{D_4}{D_3} v_{N2} - \frac{D_4 R_b}{D_3 J_2} h_2 - \frac{D_4}{D_3 m_2} p_2 + \frac{k_4}{D_3} x_3 - \frac{k_3}{D_3} x_{2b} \right)}{\left( 1 - \frac{D_4}{D_3} \right)} \right) \right) \\
&\quad - (R_a)^{-1} \left( x_{2a}k_2 + D_2 \left( \frac{\left( \frac{D_1}{D_2} v_{N1} + \frac{D_1 R_a}{D_2 J_2} h_2 - \frac{D_1}{D_2 m_2} p_2 + \frac{k_1}{D_2} x_1 - \frac{k_2}{D_2} x_{2a} \right)}{\left( 1 - \frac{D_1}{D_2} \right)} \right) \right) \\
&= \left( \frac{x_{2b}k_2}{R_b} + \frac{D_3}{R_b} \left( \frac{\left( \frac{D_4}{D_3} v_{N2} - \frac{D_4 R_b}{D_3 J_2} h_2 - \frac{D_4}{D_3 m_2} p_2 + \frac{k_4}{D_3} x_3 - \frac{k_3}{D_3} x_{2b} \right)}{\left( 1 - \frac{D_4}{D_3} \right)} \right) \right) \\
&\quad - \left( \frac{x_{2a}k_2}{R_a} + \frac{D_2}{R_a} \left( \frac{\left( \frac{D_1}{D_2} v_{N1} + \frac{D_1 R_a}{D_2 J_2} h_2 - \frac{D_1}{D_2 m_2} p_2 + \frac{k_1}{D_2} x_1 - \frac{k_2}{D_2} x_{2a} \right)}{\left( 1 - \frac{D_1}{D_2} \right)} \right) \right) \\
&= \left( \frac{x_{2b}k_2}{R_b} + \left( \frac{\left( D_4 v_{N2} - \frac{D_4 R_b}{J_2} h_2 - \frac{D_4}{m_2} p_2 + k_4 x_3 - k_3 x_{2b} \right)}{R_b \left( 1 - \frac{D_4}{D_3} \right)} \right) \right) \\
&\quad - \left( \frac{x_{2a}k_2}{R_a} + \left( \frac{\left( D_1 v_{N1} + \frac{D_1 R_a}{J_2} h_2 - \frac{D_1}{m_2} p_2 + k_1 x_1 - k_2 x_{2a} \right)}{R_a \left( 1 - \frac{D_1}{D_2} \right)} \right) \right) \\
&= \frac{x_{2b}k_2}{R_b} + \frac{\left( D_4 v_{N2} - \frac{D_4 R_b}{J_2} h_2 - \frac{D_4}{m_2} p_2 + k_4 x_3 - k_3 x_{2b} \right)}{R_b \left( 1 - \frac{D_4}{D_3} \right)} - \frac{x_{2a}k_2}{R_a} - \frac{\left( D_1 v_{N1} + \frac{D_1 R_a}{J_2} h_2 - \frac{D_1}{m_2} p_2 + k_1 x_1 - k_2 x_{2a} \right)}{R_a \left( 1 - \frac{D_1}{D_2} \right)}
\end{aligned}$$



## Appendix D: Arduino script: Multiple DHT22 Temperature Sensors

```

3TempSensor_KM
// MQP Temperature System
// - Krishna Madhurkar
// - 12/2/2018

//Libraries Used
#include <DHT.h>;

//Constants
#define DHT1PIN 7 // Temperature sensor 1
#define DHT2PIN 6 // Temperature sensor 2
#define DHT3PIN 5 // Temperature sensor 3

#define DHTTYPE DHT22 // DHT 22 (AM2302)
DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);

//Variables

float hum1; //Stores humidity value
float temp1; //Stores temperature value
float hum2;
float temp2;
float hum3;
float temp3;
float avgTemp; //Stores average temperature value

void setup()
{
  Serial.begin(9600);
  Serial.println("Temperature detection");
  dht1.begin();
  dht2.begin();
  dht3.begin();
}

void loop()
{
  delay(2000);
  //Read and Store data
  hum1 = dht1.readHumidity();
  temp1 = dht1.readTemperature();
  hum2 = dht2.readHumidity();
  temp2 = dht2.readTemperature();
  hum3 = dht3.readHumidity();
  temp3 = dht3.readTemperature();
  avgTemp = ( temp1 + temp2 + temp3 )/3 ;

  //Print temp and humidity values to serial monitor
  //Check if nan -> not a number
  if (isnan(temp1) || isnan(hum1)) {
    Serial.println("Failed to read from DHT #1");
  } else {
    Serial.print("Humidity: ");
    Serial.print(hum1);
    Serial.print(" %, Temp: ");
    Serial.print(temp1);
    Serial.println(" Celsius");
  }
  if (isnan(temp2) || isnan(hum2)) {
    Serial.println("Failed to read from DHT #2");
  } else {
    Serial.print("Humidity: ");
    Serial.print(hum2);
    Serial.print(" %, Temp: ");
    Serial.print(temp2);
    Serial.println(" Celsius");
  }
  if (isnan(temp3) || isnan(hum3)) {
    Serial.println("Failed to read from DHT #3");
  } else {
    Serial.print("Humidity: ");
    Serial.print(hum3);
    Serial.print(" %, Temp: ");
    Serial.print(temp3);
    Serial.println(" Celsius");
  }
  if (isnan(avgTemp) || isnan(temp1) || isnan(temp2) || isnan(temp3) ) {
    Serial.println("Average invalid: Failed to read from 1/3 DHT");
  } else {
    Serial.print(" Average Temp: ");
    Serial.print(avgTemp);
    Serial.println(" Celsius");
    delay(2000); //Delay 2 sec.
  }
}
//End

```

Figure D-1: Temperature Sensor Arduino Code

## Appendix E: Arduino script: IMU

```
Heading §
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
}

float m_min = (LSM303::vector<int16_t>){-32767, -32767, -32767};
float m_max = (LSM303::vector<int16_t>){+32767, +32767, +32767};

void loop() {
  compass.read();

  float heading = compass.heading();

  Serial.println(heading);
  delay(100);
}

gyro_minimu §
#include <Wire.h>
#include <L3G.h>

L3G gyro;

float G_Dt=0.005; // Integration time (DCM algorithm)
// We will run the integration loop at 50Hz if possible

long timer=0; //general purpose timer
long timer1=0;

float G_gain=.00875; // gyros gain factor for 250deg/s
float gyro_x; //gyro x val
float gyro_y; //gyro y val
float gyro_z; //gyro z val
float gyro_xold; //gyro cumulative x value
float gyro_yold; //gyro cumulative y value
float gyro_zold; //gyro cumulative z value
float gerrx; // Gyro x error
float gerry; // Gyro y error
float gerrz; // Gyro z error

void setup() {
  Serial.begin(9600);
  Wire.begin(); // i2c begin

  if (!gyro.init()) // gyro init
  {
    Serial.println("Failed to autodetect gyro type!");
    while (1);
  }
  timer=millis(); // init timer for first reading

  gyro_x = gyro_x*G_Dt; // Multiply the angular rate by the time interval
  gyro_y = gyro_y*G_Dt;
  gyro_z = gyro_z*G_Dt;

  gyro_x +=gyro_xold; // add the displacement(rotation) to the cumulative displacement
  gyro_y += gyro_yold;
  gyro_z += gyro_zold;

  gyro_xold=gyro_x; // Set the old gyro angle to the current gyro angle
  gyro_yold=gyro_y;
  gyro_zold=gyro_z;

  if((millis()-timer1)>=1000) // prints the gyro value once per second
  {
    timer1=millis();
    Serial.print("G ");
    Serial.print("X: ");
    Serial.print(gyro_x);
    Serial.print("Y: ");
    Serial.print(gyro_y);
    Serial.print("Z: ");
    Serial.println(gyro_z);
  }
}
```

```

gyro.enableDefault(); // gyro init. default 250/deg/s
delay(1000); // allow time for gyro to settle
for(int i =0;i<100;i++){ // takes 100 samples of the gyro
gyro.read();
gerrx+=gyro.g.x;
gerry+=gyro.g.y;
gerrz+=gyro.g.z;
  delay(25);
}

gerrx = gerrx/100; // average reading to obtain an error/offset
gerry = gerry/100;
gerrz = gerrz/100;

Serial.println(gerrx); // print error vals
Serial.println(gerry);
Serial.println(gerrz);
}

void loop() {
  if((millis()-timer)>=5) // reads imu every 5ms
  {

gyro.read(); // read gyro
timer=millis(); //reset timer
gyro_x=(float)(gyro.g.x-gerrx)*G_gain; // offset by error then m
gyro_y=(float)(gyro.g.y-gerry)*G_gain;
gyro_z=(float)(gyro.g.z-gerrz)*G_gain;

gyro_x = gyro_x*G_Dt; // Multiply the angular rate by the time i
gyro_y = gyro_y*G_Dt;
gyro_z = gyro_z*G_Dt;

```

#### Calibrate §

```

#include <Wire.h>
#include <LSM303.h>

LSM303 compass;
LSM303::vector<int16_t> running_min = {32767, 32767, 32767};
running_max = {-32768, -32768, -32768};

char report[80];

void setup() {
  Serial.begin(9600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
}

void loop() {
  compass.read();

  running_min.x = min(running_min.x, compass.m.x);
  running_min.y = min(running_min.y, compass.m.y);
  running_min.z = min(running_min.z, compass.m.z);

  running_max.x = max(running_max.x, compass.m.x);
  running_max.y = max(running_max.y, compass.m.y);
  running_max.z = max(running_max.z, compass.m.z);

  sprintf(report, sizeof(report), "min: %+6d, %+6d, %+6d, %+6d, %+6d, %+6d",
    running_min.x, running_min.y, running_min.z,
    running_max.x, running_max.y, running_max.z);
  Serial.println(report);

  delay(100);
}

```

```

Complimentfilter
#include <Wire.h>
#include <L3G.h>
#include <LSM303.h>
3G gyro;
SM303 accel;

loat G_Dt=0.020; // Integration time (DCM algorithm) We will

ong timer=0; //general purpose timer
ong timer1=0;
ong timer2=0;

loat G_gain=.00875; // gyros gain factor for 250deg/sec
loat gyro_x; //gyro x val
loat gyro_y; //gyro x val
loat gyro_z; //gyro x val
loat gyro_xold; //gyro cummulative x value
loat gyro_yold; //gyro cummulative y value
loat gyro_zold; //gyro cummulative z value
loat gerrx; // Gyro x error
loat gerry; // Gyro y error
loat gerrz; // Gyro 7 error

loat A_gain=.00875; // gyros gain factor for 250deg/sec
loat accel_x; //gyro x val
loat accel_y; //gyro x val
loat accel_z; //gyro x val
loat accel_xold; //gyro cummulative x value
loat accel_yold; //gyro cummulative y value
loat accel_zold; //gyro cummulative z value
loat aerrx; // Accel x error
loat aerry; // Accel y error
loat aerrz; // Accel 7 error

void gyroZero(){
// takes 200 samples of the gyro
for(int i =0;i<200;i++){
gyro.read();
gerrx+=gyro.g.x;
gerry+=gyro.g.y;
gerrz+=gyro.g.z;
delay(20);
}
gerrx = gerrx/200; // average reading to obtain an error/offset
gerry = gerry/200;
gerrz = gerrz/200;

Serial.println(gerrx); // print error vals
Serial.println(gerry);
Serial.println(gerrz);
}

void readGyro(){
gyro.read(); // read gyro
timer=millis(); //reset timer
gyro_x=(float)(gyro.g.x-gerrx)*G_gain; // offset by error then multiply by gyro
gyro_y=(float)(gyro.g.y-gerry)*G_gain;
gyro_z=(float)(gyro.g.z-gerrz)*G_gain;

gyro_x = gyro_x*G_Dt; // Multiply the angular rate by the time interval
gyro_y = gyro_y*G_Dt;
gyro_z = gyro_z*G_Dt;

gyro_x +=gyro_xold; // add the displacement(rotation) to the cumulative displom
gyro_y += gyro_yold;
gyro_z += gyro_zold;

gyro_xold=gyro_x ; // Set the old gyro angle to the current gyro angle

```

```

    gyro_yold=gyro_y ;
    gyro_zold=gyro_z ;
}

void printGyro(){
    timer2=millis();

    // The gyro_axis variable keeps track of roll, pitch,yaw based on the com
    Serial.print(" GX: ");
    Serial.print(gyro_x);
    Serial.print(" GY: ");
    Serial.print(gyro_y);
    Serial.print(" GZ: ");
    Serial.print(gyro_z);

    Serial.print(" Ax = ");
    Serial.print(accel_x);
    Serial.print(" Ay = ");
    Serial.print(accel_y);
    Serial.print(" Az = ");
    Serial.println(accel_z);
}

void Accel_Init()
{
    accel.init();
    accel.enableDefault();
    Serial.print("Accel Device ID");
    Serial.println(accel.getDeviceType());
    switch (accel.getDeviceType())
    {
        case LSM303::device_D:
            accel.writeReg(LSM303::CTRL2, 0x18); // 8 g full scale: AFS = 011
            break;
        case LSM303::device_DLHC:
            accel.writeReg(LSM303::CTRL_REG4_A, 0x28); // 8 g full scale:
            break;
        default: // DLM, DLH
            accel.writeReg(LSM303::CTRL_REG4_A, 0x30); // 8 g full scale:
    }
}

void accelZero(){
    //I found this to be more problematic than it was worth.
    //not implemented
    // takes 100 samples of the accel
    for(int i =0;i<100;i++){
        gyro.read();
        aerrx+=accel.a.x >> 4;
        aerry+=accel.a.y >> 4;
        aerrz+=accel.a.z >> 4;
        delay(10);
    }
    aerrx = gerrx/100; // average reading to obtain an error/offset
    aerry = gerry/100;
    aerrz = gerrz/100;
    Serial.println("accel starting values");
    Serial.println(aerrx); // print error vals
    Serial.println(aerry);
    Serial.println(aerrz);

    // Reads x,y and z accelerometer registers
    void readAccel()

    accel.readAcc();
}

```

```

accel_x = accel.a.x >> 4; // shift left 4 bits to use 12-bit repres
accel_y = accel.a.y >> 4;
accel_z = accel.a.z >> 4;

// accelerations in G
accel_x = (accel_x/256);
accel_y = (accel_y/256);
accel_z = (accel_z/256);
}

void complimentaryFilter(){
  readGyro();
  readAccel();
  float x_Acc,y_Acc;
  float magnitudeofAccel= (abs(accel_x)+abs(accel_y)+abs(accel_z));
  if (magnitudeofAccel > 6 && magnitudeofAccel < 1.2)

  x_Acc = atan2(accel_y,accel_z)*180/ PI;
  gyro_x = gyro_x * 0.98 + x_Acc * 0.02;

  y_Acc = atan2(accel_x,accel_z)* 180/PI;
  gyro_y = gyro_y * 0.98 + y_Acc * 0.02;

}

void setup() {
  Serial.begin(9600);
  Wire.begin(); // i2c begin

  if (!gyro.init()){ // gyro init
    Serial.println("Failed to autodetect gyro type!");
    while (1);
  }
  timer=millis(); // init timer for first reading
  gyro.enableDefault(); // gyro init. default 250/deg/
  delay(1000); // allow time for gyro to settle
  Serial.println("starting calibration");
  gyroZero();
  Accel_Init();

}

void loop() {
  // reads imu every 20ms
  if((millis()-timer)>=20)
  {
  //complimentaryFilter();
  readGyro();
  }
  // prints the gyro value once per second
  if((millis()-timer2)>=1000)
  {
  printGyro();
  }
}

```

Figure D-2: IMU Arduino Code

## Appendix F: Bill of Materials

Table F-1: Bill of Materials

Item	Quantity	Material	Manufacturing Process
Bumper	1	PLA	3D Printed
Wheels	4	Rubber / Plastic	Bought
Wheel Adaptor	2	Delrin	Turned
3M Screws	2	Steel	Bought
FS Top Link	2	Aluminum	Milled
MSHXNUT0.13832	2	Steel	Bought
FS Wheel Mount Bracket	2	PLA	3D Printed
FS Bottom Link	2	PLA	3D Printed
FS Mounting Plate	1	Aluminum	Milled
Steering Frame Mount	1	Aluminum	Milled
Shock Spacer	2	PLA	3D Printed
Nylon Bearing	2	Nylon	Bought
Wheel Mount	2	PLA	3D Printed
Camera One Steering Mount	1	PLA	3D Printed
Steering Bar	1	PLA	3D Printed
Pivot Post with Arm Mount	1	PLA	3D Printed
Pivot Post without Arm Mount	1	PLA	3D Printed
Link	1	PLA	3D Printed
Pivot Post Support	1	PLA	3D Printed
Steering Arm	1	PLA	3D Printed
Servo Control Horns	1	Plastic	Bought
Servo	1	N.A.	Bought
Servo Housing	1	PLA	3D Printed
Fan	1	N.A.	3D Printed
Receiver	1	N.A.	Bought

ESC	1	N.A.	Bought
ESC and Receiver Mount	1	PLA	3D Printed
Battery	1	N.A.	Bought
Battery and IMU Mount	1	PLA	3D Printed
IMU	1	N.A.	Bought
Motor Temperature Mount	1	PLA	3D Printed
Temperature Sensor	1	N.A.	Bought
Circuit Box 1	1	PLA	3D Printed
Circuit Box 2	1	PLA	3D Printed
Motor	1	N.A.	Bought
Motor Mounting Plate	1	Delrin	Milled
Gearbox Spacer	2	PLA	3D Printed
Center Plate	1	Delrin	Milled
Outer Plate`	1	Delrin	Milled
1-8_in_low_friction_brush_bearing	3	Nylon	Bought
1-4_in_low_friction_thrust_bearing	4	Nylon	Bought
1-4in_External Self -Locking Retaining Ring	2	Steel	Bought
1-4 in teflon sleeve brushing	2	teflon	Bought
Compound Gear Axle	1	Aluminum	Turned
Outer Shaft	1	Aluminum	Turned
48P30Tgear	2	Steel	Bought
7880K170_12T	2	Steel	Bought
Pin cross head ai	4	Steel	Bought
B18.3.4M - 3 x 0.5 x 5 SBHCS --N	3	Steel	Bought
RS Mount To Base Plate	2	PLA	3D Printed
Rear SuspensionTower	2	PLA	3D Printed
LinkNoCurves	4	PLA	3D Printed
MounToRearAxle_Right	1	PLA	3D Printed



MounToRearAxle_Left	1	PLA	3D Printed
Rear Diff Housing_Bottom	1	PLA	3D Printed
Rear Diff Housing_Top	1	PLA	3D Printed
Rear Diff Aluminum Plate	4	Aluminum	Milled
Differential_Large Gear	1	Steel	Bought
Differential_Small Gear	1	Steel	Bought
Universal Joints	2	Plastic	Bought
10-32 Bolts	2	Steel	Bought
8-32 Bolts	32	Steel	Bought
10-32 Nuts	2	Steel	Bought
8-32 Nuts	32	Steel	Bought
Pins	6	Steel	Bought
Arduino Mega	2	N.A.	Bought
Raspberry Pi 3 B+	2	N.A.	Bought
Circuit Board	2	N.A.	Bought
Shock Absorbers	4	N.A.	Bought
Base Plate	1	PLA	3D Printed
Rear Axle	2	PLA	3D Printed
Tachometer	1	N.A.	Bought
LEDs	4	N.A.	Bought
Resistors	8	N.A.	Bought

## Appendix G: Arduino Script: Sensor System

```
//Temperature Sensor Includes
#include <DHT.h>

//Constants
const int DHT1PIN = 7; // Temperature sensor 1
const int DHT2PIN = 6; // Temperature sensor 2
const int DHT3PIN = 5; // Temperature sensor 3
const int FAN_CONTROL_PIN = 9; // Fan PWM (4th wire)

#define DHTTYPE DHT22 // DHT 22 (AM2302)
DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);
//Variables
float hum1; //Stores humidity value
float temp1; //Stores temperature value
float hum2;
float temp2;
float hum3;
float temp3;
float avgTemp; //Stores average temperature value

//Fan Controller Includes

int FAN_RPM = A3;
int analogPin = 3; // Temp data connected to analog pin 3
int fanSpeed = 0; // variable to store the read value
int fanRpm = 0;
int i = 0;

//IMU Includes
//Written by Joe St. Germain 10/1/16
// Simply keeps track of the gyro reading from an minimu9
// I recommend going one step further with a
// complimentary filter with the accelerometer for greater accuracy over longer periods of time.
// http://www.pieter-jan.com/node/11 Is the work that I based this project off and its a good reference.
// This filter should be tested and adjusted to work for your purpose.
//Gyros and Accelerometers are MEMS (Miniature elorto mechanical systems) devices. They are actual physical
//mechanical systems with electrical sensors attached to them inside the IC.
//For this reason each module is slightly diffrent. Also you may find increasing the time constant and or
//the gyro rate and accelerometer sensitivity may increase performance.
//If you would like to suggest changes to this code based on your results notify the TA.
//Thank you and have a great term. -Joe

#include <Wire.h>
#include <L3G.h>
#include <LSM303.h>
L3G gyro;
LSM303 accel;

float G_Dt = 0.020; // Integration time (DCM algorithm) We will run the integration loop at 50Hz if possible

long timer = 0; //general purpose timer
long timer1 = 0;
long timer2 = 0;

float G_gain = .00875; // gyros gain factor for 250deg/sec
float gyro_x; //gyro x val
float gyro_y; //gyro x val
float gyro_z; //gyro x val
float gyro_xold; //gyro cummulative x value
float gyro_yold; //gyro cummulative y value
float gyro_zold; //gyro cummulative z value
float gerrx; // Gyro x error
float gerry; // Gyro y error
float gerrz; // Gyro 7 error

float A_gain = .00875; // gyros gain factor for 250deg/sec
```

```

float accel_x; //gyro x val
float accel_y; //gyro x val
float accel_z; //gyro x val
float accel_xold; //gyro cumulative x value
float accel_yold; //gyro cumulative y value
float accel_zold; //gyro cumulative z value
float aerrx; // Accel x error
float aerry; // Accel y error
float aerrz; // Accel 7 error

volatile byte revolutions;
float rpm;
unsigned long timeon;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  dht1.begin(); //Temperature Sensor 1
  dht2.begin(); //Temperature Sensor 2
  dht3.begin(); //Temperature Sensor 3
  pinMode(FAN_CONTROL_PIN, OUTPUT); // Fan Output
  pinMode(FAN_RPM, INPUT); // Fan Input

  Wire.begin(); // i2c begin

  if (!gyro.init()) { // gyro init
    //Serial.println("Failed to autodetect gyro type!");
    while (1);
  }

  timer = millis(); // init timer for first reading
  gyro.enableDefault(); // gyro init. default 250/deg/s
  delay(1000); // allow time for gyro to settle
  //Serial.println("starting calibration");
  gyroZero();
  Accel_Init();

  // AttachInterrupt automatically updates values
  // DigitalPinToInterrupt selects pin to external peripheral (Hall Effect Sensor)
  // Is triggered per each rising edge case (logic goes from low to high)
  attachInterrupt(digitalPinToInterrupt(2), revolution_counter, RISING); // for Arduino Mega we can use pins 2, 3, 18, 19, 20, 21

  // Initialize Values
  revolutions = 0;
  rpm = 0;
  timeon = 0;
}

void loop() {
  delay(500);

  if (revolutions > 2) {
    // Calculate RPM
    rpm = revolutions * 60000 / (millis() - timeon);
    // Reset Parameters To Get Next Accurate Reading
    timeon = millis();
    revolutions = 0;
  }
  // Serial.println(rpm);
}

if ((millis() - timer) >= 20)
{
  //complimentaryFilter();
  readGyro();
  getHumidities();
  setFanSpeed();
}
// prints the gyro value once per second
if ((millis() - timer2) >= 1000)
{
  String tToP = printTemperatures();
  // printAllSensors();
  String gyroToP = printGyroData();
  Serial.println(gyroToP + "," + tToP + "," + String(fanSpeed) + "," + String(rpm));
}
}

```

```

String printGyroData() {
  timer2 = millis();

  String gyroToPrint = String(gyro_x) + "," + String(gyro_y) + "," + String(gyro_z);
  return gyroToPrint;
}

String printTemperatures() {
  // Serial.print(String(temp1) + "," + String(temp2) + "," + String(temp3));
  String tempToPrint = String(temp1) + "," + String(temp2) + "," + String(temp3);
  return tempToPrint;
}

void setFanSpeed() {
  if (avgTemp < 15) {
    fanSpeed = 85; // one third the PWM val (255)
  } else if (avgTemp < 20) {
    fanSpeed = 127; // half the PWM 255
  } else if (avgTemp < 25) {
    fanSpeed = 170;
  } else {
    fanSpeed = 255; // 100% PWM duty
  }
  analogWrite(FAN_CONTROL_PIN, fanSpeed);
}

void getHumidities() {
  hum1 = dht1.readHumidity();
  temp1 = dht1.readTemperature();
  hum2 = dht2.readHumidity();
  temp2 = dht2.readTemperature();
  hum3 = dht3.readHumidity();
  temp3 = dht3.readTemperature();
  temp3 = 21;
  avgTemp = ( temp1 + temp2 + temp3 ) / 3 ;
}

void gyroZero() {
  // takes 200 samples of the gyro
  for (int i = 0; i < 200; i++) {
    gyro.read();
    gerrx += gyro.g.x;
    gerry += gyro.g.y;
    gerrz += gyro.g.z;
    delay(20);
  }
  gerrx = gerrx / 200; // average reading to obtain an error/offset
  gerry = gerry / 200;
  gerrz = gerrz / 200;

  //Serial.println(gerrx); // print error vals
  //Serial.println(gerry);
  //Serial.println(gerrz);
}

void readGyro() {
  gyro.read(); // read gyro
  timer = millis(); //reset timer
  gyro_x = (float)(gyro.g.x - gerrx) * G_gain; // offset by error then multiply by gyro gain factor
  gyro_y = (float)(gyro.g.y - gerry) * G_gain;
  gyro_z = (float)(gyro.g.z - gerrz) * G_gain;

  gyro_x = gyro_x * G_Dt; // Multiply the angular rate by the time interval
  gyro_y = gyro_y * G_Dt;
  gyro_z = gyro_z * G_Dt;

  gyro_x += gyro_xold; // add the displacement(rotation) to the cumulative displacement
  gyro_y += gyro_yold;
  gyro_z += gyro_zold;

  gyro_xold = gyro_x; // Set the old gyro angle to the current gyro angle
  gyro_yold = gyro_y;
  gyro_zold = gyro_z;
}

```

```

}

void printGyro() {
  timer2 = millis();

  // The gyro_axis variable keeps track of roll, pitch,yaw based on the complimentary filter
  Serial.print(" GX: ");
  Serial.print(gyro_x);
  Serial.print(" GY: ");
  Serial.print(gyro_y);
  Serial.print(" GZ: ");
  Serial.print(gyro_z);

  Serial.print(" Ax = ");
  Serial.print(accel_x);
  Serial.print(" Ay = ");
  Serial.print(accel_y);
  Serial.print(" Az = ");
  Serial.println(accel_z);
}

void Accel_Init()
{
  accel.init();
  accel.enableDefault();
  //Serial.print("Accel Device ID");
  //Serial.println(accel.getDeviceType());
  switch (accel.getDeviceType())
  {
    case LSM303::device_D:
      accel.writeReg(LSM303::CTRL2, 0x18); // 8 g full scale: AFS = 011
      break;
    case LSM303::device_DLHC:
      accel.writeReg(LSM303::CTRL_REG4_A, 0x28); // 8 g full scale: FS = 10; high resolution output mode
      break;
    default: // DLM, DLH
      accel.writeReg(LSM303::CTRL_REG4_A, 0x30); // 8 g full scale: FS = 11
  }
}

void accelZero() {
  //! found this to be more problematic than it was worth.
  //! not implemented
  // takes 100 samples of the accel
  for (int i = 0; i < 100; i++) {
    gyro.read();
    aerrx += accel.a.x >> 4;
    aerry += accel.a.y >> 4;
    aerrz += accel.a.z >> 4;
    delay(10);
  }
  aerrx = gerrx / 100; // average reading to obtain an error/offset
  aerry = gerry / 100;
  aerrz = gerrz / 100;
  //Serial.println("accel starting values");
  //Serial.println(aerrx); // print error vals
  //Serial.println(aerry);
  //Serial.println(aerrz);
}

// Reads x,y and z accelerometer registers
void readAccel()
{
  accel.readAcc();

  accel_x = accel.a.x >> 4; // shift left 4 bits to use 12-bit representation (1 g = 256)
  accel_y = accel.a.y >> 4;
  accel_z = accel.a.z >> 4;

  // accelerations in G
  accel_x = (accel_x / 256);
  accel_y = (accel_y / 256);
  accel_z = (accel_z / 256);
}

void complimentaryFilter() {

```

```
readGyro();
readAccel();
float x_Acc, y_Acc;
float magnitudeofAccel = (abs(accel_x) + abs(accel_y) + abs(accel_z));
if (magnitudeofAccel > 6 && magnitudeofAccel < 1.2)
{
  x_Acc = atan2(accel_y, accel_z) * 180 / PI;
  gyro_x = gyro_x * 0.98 + x_Acc * 0.02;

  y_Acc = atan2(accel_x, accel_z) * 180 / PI;
  gyro_y = gyro_y * 0.98 + y_Acc * 0.02;
}
}

void revolution_counter(){
// Increment revolutions by one per every rising edge of hall effect sensor
  revolutions++;
}
```

## Appendix H: Dynamical Analysis of Steering Assembly

Dynamic analysis of Steering

Assumptions:

- Rigid Links
- All links are assumed to be 3D printed of PLA
- 2D because forces in third dimension are negligible
- Frictionless and massless joints
- Links and masses are small and can be considered point masses

Loop 1 (EGIJ):

Velocities:

$$\begin{aligned} \vec{r}_{EG} + \vec{r}_{GI} + \vec{r}_{IJ} + \vec{r}_{JE} &= 0 \\ \vec{v}_{EG} + \vec{v}_{GI} + \vec{v}_{IJ} &= 0 \\ -\vec{\omega}_{DG} \times \vec{r}_{EG} + \vec{\omega}_{GI} \times \vec{r}_{GI} + \vec{\omega}_{IJ} \times \vec{r}_{IJ} &= 0 \\ \left(-\omega_{DG}(r_{EGz}) + \omega_{GI}(r_{GIz}) + \omega_{IJ}(r_{IJz})\right)\hat{j} - \left(-\omega_{DG}(r_{EGx}) + \omega_{GI}(r_{GIx}) + \omega_{IJ}(r_{IJx})\right)\hat{i} &= 0 \end{aligned}$$

Accelerations:

$$\begin{aligned} \vec{a}_{EG} + \vec{a}_{GI} + \vec{a}_{IJ} &= 0 \\ -\vec{a}_{DG} \times \vec{r}_{EG} - \vec{\omega}_{DG} \times (-\vec{\omega}_{DG} \times \vec{r}_{EG}) + \vec{a}_{GI} \times \vec{r}_{GI} + \vec{\omega}_{GI} \times (\vec{\omega}_{GI} \times \vec{r}_{GI}) + \vec{a}_{IJ} \times \vec{r}_{IJ} + \vec{\omega}_{IJ} & \\ \times (\vec{\omega}_{IJ} \times \vec{r}_{IJ}) &= 0 \\ -\vec{a}_{DG} \times (r_{EGz}\hat{i} + r_{EGx}\hat{j}) - \vec{\omega}_{DG} \times (-\vec{\omega}_{DG} \times (r_{EGz}\hat{i} + r_{EGx}\hat{j})) + \vec{a}_{GI} \times (r_{GIz}\hat{i} + r_{GIx}\hat{j}) + \vec{\omega}_{GI} & \\ \times (\vec{\omega}_{GI} \times (r_{GIz}\hat{i} + r_{GIx}\hat{j})) + \vec{a}_{IJ} \times (r_{IJz}\hat{i} + r_{IJx}\hat{j}) + \vec{\omega}_{IJ} & \\ \times (\vec{\omega}_{IJ} \times (r_{IJz}\hat{i} + r_{IJx}\hat{j})) &= 0 \\ -(\alpha_{DG}(r_{EGz})\hat{j} - \alpha_{DG}(r_{EGx})\hat{i}) - (\omega_{DG}(\omega_{DG}r_{EGx})\hat{j} + \omega_{DG}(\omega_{DG}r_{EGz})\hat{i}) & \\ + (\alpha_{GI}(r_{GIz})\hat{j} - \alpha_{GI}(r_{GIx})\hat{i}) + (\omega_{GI}(\omega_{GI}r_{GIx})\hat{j} - \omega_{GI}(\omega_{GI}r_{GIz})\hat{i}) & \\ + (\alpha_{IJ}(r_{IJz})\hat{j} - \alpha_{IJ}(r_{IJx})\hat{i}) + (\omega_{IJ}(\omega_{IJ}r_{IJx})\hat{j} - \omega_{IJ}(\omega_{IJ}r_{IJz})\hat{i}) &= 0 \\ \hat{i}: -\alpha_{DG}(r_{EGx}) + \alpha_{GI}(r_{GIx}) + \alpha_{IJ}(r_{IJx}) = \omega_{DG}(\omega_{DG}r_{EGz}) - \omega_{GI}(\omega_{GI}r_{GIz}) - \omega_{IJ}(\omega_{IJ}r_{IJz}) & \\ \hat{j}: -\alpha_{DG}(r_{EGz}) + \alpha_{GI}(r_{GIz}) + \alpha_{IJ}(r_{IJz}) = \omega_{DG}(\omega_{DG}r_{EGx}) - \omega_{GI}(\omega_{GI}r_{GIx}) - \omega_{IJ}(\omega_{IJ}r_{IJx}) & \end{aligned}$$

Using similar expansions, the other equations were solved.

Loop 2 (FHKL):

Velocities:

$$\left(-\omega_{FH}(r_{FHx}) - \omega_{HK}(r_{HKx}) + \omega_{KL}(r_{KLx})\right)\hat{j} - \left(-\omega_{FH}(r_{FHx}) - \omega_{HK}(r_{HKx}) + \omega_{KL}(r_{KLx})\right)\hat{i} = 0$$

Accelerations:

$$\begin{aligned} \hat{i}: -\alpha_{FH}(r_{FHx}) - \alpha_{HK}(r_{HKx}) + \alpha_{KL}(r_{KLx}) &= \omega_{FH}(\omega_{FH}r_{FHx}) + \omega_{HK}(\omega_{HK}r_{HKx}) - \omega_{KL}(\omega_{KL}r_{KLx}) \\ \hat{j}: -\alpha_{FH}(r_{FHx}) - \alpha_{HK}(r_{HKx}) + \alpha_{KL}(r_{KLx}) &= \omega_{FH}(\omega_{FH}r_{FHx}) + \omega_{HK}(\omega_{HK}r_{HKx}) - \omega_{KL}(\omega_{KL}r_{KLx}) \end{aligned}$$

Loop 3 (ABDG):

Velocities:

$$\left(-\omega_{AB}(r_{ABz}) - \omega_{BD}(r_{BDz}) - \omega_{DG}(r_{DEz})\right)\hat{j} + \left(\omega_{AB}(r_{ABx}) + \omega_{BD}(r_{BDx}) + \omega_{DG}(r_{DEx})\right)\hat{i} = 0$$

Accelerations:

$$\hat{i}: -\alpha_{AB}(r_{ABx}) - \alpha_{BD}(r_{BDx}) - \alpha_{DG}(r_{DEx}) = \omega_{AB}(\omega_{AB}r_{ABz}) + \omega_{BD}(\omega_{BD}r_{BDz}) + \omega_{DG}(\omega_{DG}r_{DEz})$$

$$\hat{j}: -\alpha_{AB}(r_{ABz}) - \alpha_{BD}(r_{BDz}) - \alpha_{DG}(r_{DEz}) = \omega_{AB}(\omega_{AB}r_{ABx}) + \omega_{BD}(\omega_{BD}r_{BDx}) + \omega_{DG}(\omega_{DG}r_{DEx})$$

Loop 4 (ABCF):

Velocities:

$$(-\omega_{AB}(r_{ABz}) - \omega_{BD}(r_{BCz}) - \omega_{CF}(r_{CFz}))\hat{j} + (\omega_{AB}(r_{ABx}) + \omega_{BD}(r_{BCx}) + \omega_{CF}(r_{CFx}))\hat{i} = 0$$

Accelerations:

$$\hat{i}: -\alpha_{AB}(r_{ABx}) - \alpha_{BD}(r_{BCx}) - \alpha_{CF}(r_{CFx}) = \omega_{AB}(\omega_{AB}r_{ABz}) + \omega_{BD}(\omega_{BD}r_{BCz}) + \omega_{CF}(\omega_{CF}r_{CFz})$$

$$\hat{j}: -\alpha_{AB}(r_{ABz}) - \alpha_{BD}(r_{BCz}) - \alpha_{CF}(r_{CFz}) = \omega_{AB}(\omega_{AB}r_{ABx}) + \omega_{BD}(\omega_{BD}r_{BCx}) + \omega_{CF}(\omega_{CF}r_{CFx})$$

Loop 5 (FHGE):

Velocities:

$$(-\omega_{FH}(r_{FHx}) - \omega_{HG}(r_{HGz}) - \omega_{DG}(r_{GEz}))\hat{j} + (\omega_{FH}(r_{FHx}) + \omega_{HG}(r_{HGx}) + \omega_{DG}(r_{GEz}))\hat{i} = 0$$

Accelerations:

$$\hat{i}: -\alpha_{FH}(r_{FHx}) - \alpha_{HG}(r_{HGx}) - \alpha_{DG}(r_{GEz}) = \omega_{FH}(\omega_{FH}r_{FHx}) + \omega_{HG}(\omega_{HG}r_{HGz}) + \omega_{DG}(\omega_{DG}r_{GEz})$$

$$\hat{j}: -\alpha_{FH}(r_{FHx}) - \alpha_{HG}(r_{HGz}) - \alpha_{DG}(r_{GEz}) = \omega_{FH}(\omega_{FH}r_{FHx}) + \omega_{HG}(\omega_{HG}r_{HGx}) + \omega_{DG}(\omega_{DG}r_{GEz})$$

Published MATLAB Document:

## Static and Dynamic analysis of Steering

Dylan McKillip

Angular Velocities:.....

Angular Accel:.....

Statics w/Torque: .....

Dynamics Equations: .....

Angular Velocities:

```
%Solves velocities of links
loop1i = [0, 0, 0, 0, -3.13, 0, 0, 0, 0, 0];
loop1j = [0, 0, 0, .75, 0, -1.5, 0, 0, 0, 0];
loop2i = [0, 0, 0, 0, 0, 0, 0, 0, -3.13, 0];
loop2j = [0, 0, 0, 0, 0, 0, .75, 0, 0, -1.5];
loop3i = [0, 2.57, 0, 0, 0, 0, 0, 0, 0, 0];
loop3j = [-.621, .811, 0, .56, 0, 0, 0, 0, 0, 0];
loop4i = [0, 1.57, 0, 0, 0, 0, 0, 0, 0, 0];
loop4j = [-.621, .741, .63, 0, 0, 0, 0, 0, 0, 0];
loop5i = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0];
loop5j = [0, 0, 0, .75, 0, 0, -.75, 0, 0, 0];
rel1 = [0, 0, 0, 1, 0, 0, -1, 0, 0, 0];
```



```

re12 = [0,0,0,0,1,0,0,0,1,0];
re13 = [0,0,0,0,0,1,0,0,0,-1];
input = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0];

A = [loop1i;loop1j;loop2i;loop2j;loop3i;loop3j;...
     loop4i;loop4j;loop5i;loop5j;re11;re12;re13;...
     input];
in = 19.04; %servo speed in rads/s
B = [0;0;0;0;0;0;0;0;0;0;0;in];
sol = linsolve(A,B)

```

```

sol =

    19.0400
    -0.0000
    18.7680
    21.1140
         0
    10.5570
    21.1140
         0
         0
    10.5570

```

## Angular Accel:

solves angular accel of loops

```

b = [-167.18;0;-167.18;0;-24.523;0;3.21068;0;0;0;0;0];
a = [-loop1i;loop1j;-loop2i;loop2j;-loop3i;loop3j;...
     -loop4i;loop4j;-loop5i;loop5j;re11;re12;re13];

asol = linsolve(a,b)

```

Warning: Rank deficient, rank = 9, tol = 9.484908e-15.

```

asol =

    8.3490
    6.3930
    0.7103
   -0.0000
  -44.3569
    0.0000
         0
         0
  -44.3569
    0.0000

```



```
13.2986
-2.7966
0.0000
-7.8021
-0.0000
4.9933
0.0000
7.8021
-0.0000
-7.8021
0.0000
-4.9933
14.5365
```

### Dynamics Equations:

```
%Using accels at COM and angular accels and mass solves dynamics matrix
dynin = [0;0;0.009087469;0;0;0;0.000795698;0;0;0;...
        -1.226521477;0;0;0;-1.226521477;0;0;0.029267212;...
        0.097253824;0;0;0;-0.783719793;-0.783719793;19.04];

dynamics = linsolve(sa,dynin)
```

```
Warning: Rank deficient, rank = 22, tol = 2.306160e-14.
```

```
dynamics =
```

```
0.0000
16.1406
-0.0000
-16.1315
-13.3754
2.7923
13.3754
10.5468
-13.1658
0
13.3754
-2.7916
-0.2096
-7.7901
0.2290
4.9857
0.2096
6.5636
-0.2096
-6.5636
-0.2484
-6.2122
14.5435
```

Excel Tables:

Joint	Z(in)	X(in)			
A	0	0			
B	-0.621	0			
C	0.12	-1.57			
D	0.19	-2.57			
E	0.75	-2.57			
F	0.75	-1.57			
G	1.5	-2.57			
H	1.5	-1.57			
I	1.615	-5.7			
J	2.4	-5.7			
K	1.615	-1.52			
L	2.4	1.52		PLA (lb/in <sup>3</sup> )	
				0.045159	
Segment	z	x	length	Mass	
AB	0.621	0	0.621	0.003505	
BC	0.741	1.57	1.736082	0.0098	
BD	0.811	2.57	2.694925	0.015213	
DE	0.56	0	0.56	0.003161	
DG	1.69	0	1.69	0.00954	
EG	0.75	0	0.75	0.004234	
GI	3.13	0	3.13	0.017668	
IJ	1.5	0	1.5	0.008467	
FH	0.75	0	0.75	0.004234	
HK	3.13	0	3.13	0.017668	
KL	1.5	0	1.5	0.008467	
CF	0.63	0	0.63	0.003556	
GH	0	1	1	0.005645	

Dynamics Matrix		A																			B	solution				
F/M	Az	Ax	Bz	Bx	Cz	Cx	Dz	Dx	Ez	Ex	Fz	Fx	Gz	Gx	Hx	Hx	Iz	Ix	Jz	Jx	Kz	Kx	T	ans		
Az	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Az	0
Abz	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Ax	16.1406
Abx	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Bz	0
BCDz	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Bx	-16.1315
BCDx	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Cz	-13.3754
CFz	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Cx	2.7923
CFx	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Dz	13.3754
DEGz	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Dx	10.5468
DEGx	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	Ez	-13.1658
Giz	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	Ex	-1.22652
Gix	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	Fz	13.3754
Ijz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	Fx	-2.7916
Ijx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	Gz	-0.2096
HKz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	Gx	-7.7901
HKx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	Hx	0.229
GHx	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	Hx	4.9857
GHz	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	Iz	0.2096
M_AB	0	0	0	0.621	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Ix	6.5636
M_BD	0	0	0	0.811	-1	-0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Iz	-0.2096
M_DG	0	0	0	0	0	0	0	0	0.56	0	0	0	0	0	0.75	0	0	0	0	0	0	0	0	0	Jx	-6.5636
M_GH	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Kz	-0.2484
M_FH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.75	0	0	0	0	0	0	0	0	0	Kx	-6.2122
M_HK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3.13	0	0	T	-0.78372
M_GI	0	0	0	0	0	0	0	0	0	0	0	0	3.13	0	0	0	0	0	0	0	0	0	0	0	T	14.5435
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	T	19.04

Statics Matrix		A																			B	solution				
F/M	Az	Ax	Bz	Bx	Cz	Cx	Dz	Dx	Ez	Ex	Fz	Fx	Gz	Gx	Hx	Hx	Iz	Ix	Jz	Jx	Kz	Kx	T	ans		
Az	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Az	0
Abz	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Ax	16.1563
Abx	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Bz	0
BCDz	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Bx	-16.1563
BCDx	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Cz	-13.2986
CFz	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Cx	2.7966
CFx	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Dz	13.2986
DEGz	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Dx	10.5631
DEGx	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	Ez	-13.2986
Giz	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	Ex	0
Gix	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	Fz	13.2986
Ijz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	Fx	-2.7966
Ijx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	Gz	0
HKz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	Gx	-7.8021
HKx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	Hx	0
GHx	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	Hx	4.9933
GHz	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	Iz	0
M_AB	0	0	0	-0.621	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Ix	7.8021
M_BD	0	0	0	0.811	-1	-0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Iz	0
M_DG	0	0	0	0	0	0	0	0	0.56	0	0	0	0	0	0.75	0	0	0	0	0	0	0	0	0	Ix	-7.8021
M_GH	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Kz	0
M_FH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.75	0	0	0	0	0	0	0	0	Kx	-4.9933
M_HK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3.13	0	0	T	-0.78372
M_GI	0	0	0	0	0	0	0	0	0	0	0	0	3.13	0	0	0	0	0	0	0	0	0	0	0	T	14.5365
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	T	19.04

Angular Accelerations:		A										B	input		x =:		sol: rad/s^2	
	a_AB	a_BD	a_CF	a_DG	a_GI	a_IJ	a_FH	a_HG	a_HK	w_KL								
Loop1 i	0	0	0	-0.75	3.13	1.5	0	0	0	0	-167.18	a_AB	8.349					
Loop1 j	0	0	0	0	0	0	0	0	0	0	0	a_BD	6.393					
Loop2 i	0	0	0	0	0	0	-0.75	0	-3.13	1.5	-167.18	a_CF	0.7103					
Loop2 j	0	0	0	0	0	0	0	0	0	0	0	a_DG	0					
Loop3 i	-0.621	-0.811	0	-0.56	0	0	0	0	0	0	-24.523	a_GI	-44.3569					
Loop3 j	0	2.57	0	0	0	0	0	0	0	0	0	a_IJ	0					
Loop4 i	-0.621	-0.741	-0.63	0	0	0	0	0	0	0	3.21068	a_FH	0					
Loop4 j	0	1.57	0	0	0	0	0	0	0	0	0	a_HG	0					
Loop5 i	0	0	0	-0.75	0	0	0.75	0	0	0	0	a_HK	-44.3569					
Loop5 j	0	0	0	0	0	0	0	1	0	0	0	w_KL	0					
rel wdg_f	0	0	0	1	0	0	-1	0	0	0	0							
rel wgi_h	0	0	0	0	1	0	0	0	1	0	0							
rel wkl_ij	0	0	0	0	0	1	0	0	0	-1	0							

Angular Velocity		A										B	input		x =:		sol rad/s^	
	w_AB	w_BD	w_CF	w_DG	w_GI	w_IJ	w_FH	w_HG	w_HK	w_KL								
Loop1 i	0	0	0	0	0	0	0	0	0	0	0	w_AB	19.04					
Loop1 j	0	0	0	-0.75	3.13	1.5	0	0	0	0	0	w_BD	0					
Loop2 i	0	0	0	0	0	0	0	0	0	0	0	w_CF	18.768					
Loop2 j	0	0	0	0	0	0	-0.75	0	-3.13	1.5	0	w_DG	21.114					
Loop3 i	0	2.57	0	0	0	0	0	0	0	0	0	w_GI	0					
Loop3 j	-0.621	-0.811	0	-0.56	0	0	0	0	0	0	0	w_IJ	10.557					
Loop4 i	0	1.57	0	0	0	0	0	0	0	0	0	w_FH	21.114					
Loop4 j	-0.621	-0.741	-0.63	0	0	0	0	0	0	0	0	w_HG	0					
Loop5 i	0	0	0	0	0	0	0	1	0	0	0	w_HK	0					
Loop5 j	0	0	0	-0.75	0	0	0.75	0	0	0	0	w_KL	10.557					
rel wdg_f	0	0	0	1	0	0	-1	0	0	0	0							
rel wgi_h	0	0	0	0	1	0	0	0	1	0	0							
rel wkl_ij	0	0	0	0	0	1	0	0	0	-1	0							
input	1	0	0	0	0	0	0	0	0	0	0	19.04						

	I		in/s <sup>2</sup>		
Moment of Inertia:		Accel COM	z	x	total
AB	0.000338	a_AB	0	2.592365	2.592365
BD	0	a_BD	0	0	0
CF	0.000353	a_CF	0	0.223745	0.223745
DG	0.000775	a_DG	0	0	0
GI	0	a_GI	-69.4185	0	69.41855
HK	0	a_IJ	0	0	0
FH	0.010369	a_FH	0	0	0
IJ	0.004763	a_HG	0	0	0
KL	0.004763	a_HK	-69.4185	0	69.41855
GH	0	a_KL	0	0	0