June, 2019

COMMERCIAL VEHICLE RESEARCH BUGGY FOR ACTIVE DRIVER ASSISTANCE SYSTEMS

FINAL DESIGN REPORT

Team Daimtronics

Nathaniel McCutcheon Furbeyre – nmfurbearr@outlook.com

Fernando Mondragon-Cardenas – femondra238@gmail.com

Ricardo Steven Lickiss Tan IV – rickytan4@hotmail.com

Team – Daimtronics@gmail.com

Sponsors

Daimler Trucks North America

Dr. Charles Birdsong, Cal Poly ME Department

CALIFORNIA POLYTECHNIC STATE UNIVERSITY – SAN LUIS OBISPO

MECHANICAL ENGINEERING – COLLEGE OF ENGINEERING

Abstract

This is the Final Design Report for Daimtronics, a senior project team sponsored by Professor Charles Birdsong of Cal Poly and by Daimler Trucks North America. This team integrated mechatronic systems into a scale semi-truck chassis using existing mechanical and software systems from three separate Cal Poly senior projects over the recent years: Daimscale, MicroLaren, and ProgreSSIV. The goal was to have a user-friendly platform capable of executing autonomous driving algorithms that are programmable at a high level in Simulink and Robotic Operating System (ROS). Advanced driver assistance and autonomous vehicle algorithms were not within the scope of this project, but the capability to upload the platform with such software was. Through research on existing products and technologies in the field today, as well as through communication with the sponsors, Daimtronics has compiled a list of customer needs and resulting engineering specifications that will verify whether the needs are met or not. Included is both the preliminary design direction, encompassing the selection of a motor, a computing platform and a middleware framework, as well as the final design direction as the project evolved. The sensor suite for object detection and detailed plans for the integration of the electronic, computing, and mechanical components are described. The proposed and final design of the motherboard integrating the electronic and computing platforms of the system is detailed. A description of the current state of the project is included, as well as suggested next steps for future teams who will be working on this platform. A timeline of key deliverables and their due dates throughout the 2018-2019 academic school year is included.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Autonomous driving for commercial vehicles is a major opportunity for companies and organizations that manage to implement the technology. A platform to expedite research in the trucking space and to enable university students to partake in autonomous vehicle and driver assistance research is therefore a large boon to both the companies that fund these projects and to society as a whole.

This project is the successor to the Daimscale senior project, the team working to create a small-scale tractor-trailer chassis with RC (radio control) capabilities. The purpose of our project is to outfit the model developed by Daimscale with the mechatronic components necessary to execute autonomous vehicle software for research and testing purposes. Target capabilities for the end of this year's project include automated driving in pre-programmed patterns, automated torque-controlled braking, and basic collision avoidance while developing detailed documentation for future users of the platform. These capabilities will demonstrate the satisfaction of the actual goal of the project, having a flexible user-centric platform capable of facilitating intelligent vehicle research and experimentation. Various sensors, actuators, and microcontrollers will need to be selected and utilized to form a cohesive and intelligent system to carry out the necessary actions of a semi-autonomous vehicle. The sponsors of this project are Daimler Trucks North America and Dr. Birdsong of Cal Poly, San Luis Obispo. The work done on this platform may also be used for a future technical elective course on driver assistance technology in development by Dr. Birdsong.

The team consists of three members: Nate Furbeyre, Fernando Mondragon, and Ricky Tan. Each member of the team is concentrating in mechatronics and has experience implementing mechatronic solutions on the mechanical, electrical, and software levels of development. The team has additional experience with general mechanical design, manufacturing, and computational methods of analysis.

This document is organized into sections to convey the relevant information regarding the project thus far. Section 2 will outline the Background, including custom needs, alternative preexisting solutions, and relevant technical research. Section 3 goes over the Objectives of the project, spelling out the problem statement, the bounds of the project, and engineering specifications. Section 4 will outline the Concept Design, with decisions made in both hardware and software design. Section 5 details the Final Design with descriptions and diagrams. Section 6 outlines our Manufacturing, detailing the specifics how we produced the platform. Section 7 will outline the Design Verification describing how the design passes or fails each of the design specifications. Section 8 contains an overview of the Project Management system that we used during development. Finally, Section 9 describes the Conclusions and future recommendations that we suggest after having finished the project.

Our code for the project is located in a GitHub repository at https://github.com/nfurbeyr/daimtronics.

2. Background

In order to develop a fully defined and contextualized problem definition, we communicated with our Sponsors, Daimler and Dr. Birdsong, and performed background research to understand preexisting solutions to intelligent vehicle research and to generally build an understanding of the current state of the intelligent vehicle field.

2.1 Customer Needs

Daimler is the primary customer and therefore is the main source of customer needs, but Dr. Birdsong's expertise was utilized in developing a consistent and meaningful set of needs and wants. The needs of each of these customers was identified in the initial conference call with Daimler and subsequent meeting with Dr. Birdsong conducted October 15, 2018.

2.1.1 Daimler's Needs - Students

Daimler aims to use this platform as a base for autonomous software competitions and as a platform for intern projects in automation. For these purposes the research buggy needs to be representative of a semi-truck, capable of operations typically associated with autonomous driving such as object avoidance, compatible with MATLAB, quick to set up for tight development cycles, and able to provide feedback with regards to the performance of the vehicle. In this respect, the value of the platform to Daimler is its capability to maximize the productivity of university and early-career talent interested in the trucking/automotive industry with regards to furthering the development of safe and effective automated trucking solutions.

2.1.2 Daimler's Needs - Researchers

More generally, this platform should be usable for Daimler researchers interested in furthering this technology. For this audience, the motion analysis data will be particularly important, along with the vehicle similitude and autonomous driving capabilities available for utilization. This aspect of Daimler's requirements is secondary to the needs listed in 2.1.1.

2.1.3 Dr. Birdsong's Needs

Dr. Birdsong is interested in developing a course dedicated to vehicle controls and dynamics and would like a platform that will be easy for mechanical engineering students who are not as well versed in low-level programming applications and who want to learn vehicle dynamics and control systems. His priorities in this case rest more on the ease-of-use, autonomous capabilities, and the data analysis capabilities for educational/lab use. The platform must be easy to integrate with Simulink, as this is a tool most mechanical engineers at Cal Poly are familiar with.

2.2 Existing Products

While searching for existing products through various research databases, news articles, and academic papers, no truly competing products or solutions were found that completely satisfied the needs of our sponsors. Therefore, it was necessary for us to widen the scope of what was determined to be a similar product and look at products that satisfied some aspects of our sponsor's needs.

Highway Pilot is Daimler's first commercial autonomous system that has reached level 2 automation as defined by SAE international [1]. Level 2 Partial Automation allows the vehicle to manage both steering and acceleration under certain driving conditions, such as in highway driving. The Freightliner Inspiration Truck is fitted with the Highway Pilot and was the first licensed truck for autonomous operation. Current capabilities include: a lane keeping assist accomplished with the use of a windscreen camera, an active brake assist used as an emergency braking assistant, a proximity control assist, a sideguard assist, a crosswind assist to counteract lane drift, and a predictive powertrain control which reacts to the

topography. The Freightliner Inspiration Truck is obviously not a scaled-down research vehicle but it is still necessary to analyze the full-scale product to determine the needs and wants of Daimler. A display on the Freightliner is shown in Figure 1.

Figure 1: Highway Pilot in action on the Freightliner Inspiration Truck [1]

Another relevant existing product is the Scale Vehicle Autonomous Test Bed created by Masters student Andrew Liburdi from the University of Windsor in 2010 as the focus of his thesis [2]. Liburdi cites a major need for test data to verify models in vehicle dynamic research and how scale testing is useful for validating control methods due to its low cost, low hazard, and high repeatability. The test bed itself was not made to analyze autonomous algorithms and safety protocols, but rather to just collect data on vehicle dynamics across various situations not easily replicated under full scale conditions.

Small scale vehicle platforms were also created at both Berkeley and MIT. The BARC (Berkeley Autonomous Race Car) is a $1/10^{th}$ scale development platform for autonomous driving [3]. It runs on open source software, utilizing Robotic Operating System (ROS), Julia, Python, and C++. It has a Odroid XU4 microcontroller that handles the high level computation, and an Arduino that communicates with the microcontroller and with the suite of sensor and actuators on the platform. The MIT RACECAR is a similar small scale vehicle used at MIT for robotics research and education [4]. Like the BARC, the MIT RACECAR utilizes open source software such as ROS. The RACECAR, however, only uses a Nvidia Jetson TX 1 embedded supercomputer that controls the platform and handles all the necessary computations. All of the vehicles created at these universities are not scaled down semi-truck research platforms; however, they are excellent platforms to study the electronic control units in a scaled vehicle.

The high cost associated with proper testing and research for autonomous systems has resulted in many forms of research being conducted through pure simulation. The issue, as previously cited by Liburdi, is that those simulations themselves need to be proven in order to be accepted. It is completely possible to create a model that uses the technical specs of a full-scale semi-truck and conducts algorithms for autonomous testing; however, without an actual real-world testing procedure, those simulations would not be completely valid.

In addition to the scaled test bed that was primarily aimed at research and analysis purposes, there are also a few different commercially available models of scaled down semi-trucks. Tamiya, a company

specialized in producing small scaled models of various real-world systems, has a 1/14th scale semi-truck that they offer [5]. A separate company called Integy also has a 1/14th scale model semi-truck [6]. While these products look similar to the full-scaled semi-trucks, the dynamics and similitude are not held to the same scaled standards. These are RC-only vehicles and exhibit no autonomous capabilities.

2.2.1 Relevant Patents

Table 1 contains a list of patents that are relevant to the autonomous vehicle engineering challenge at hand. The patent name and number are shown, along with a short description of what the patent covers.

Table 1: List of Relevant Patents

Patent Name	Patent Number	Description
Electronic control device for controlling autonomously controllable assemblies	6512970	This outlines an electronic control unit for individual, autonomously controlled assemblies used in motor vehicles. [7]
Truck with monitored and resettable electronic control units	5890080	This is a method for determining installed electronic devices on a truck by having a control unit identify the installed device from the transmitted data. There is an instrumentation control unit that provides a centralized reset to clear faults in the installed electronics. [8]
Calibration of multi-sensor system	20100235129	This is a method for preprocessing sensor data from one or more mounted sensors on a vehicle. The method is meant to overcome common obstacles between vision-based sensors and lidar sensors, using data gathered from the lidar sensor to run an algorithm and determine best sensor tilt for the vision sensor to reduce range errors [9]
Laser distance measuring equipment	105527619	The invention aims to provide laser distance measuring equipment, which comprises a laser emitting and receiving device, a reduction gear driving device and a rotating device, wherein the laser emitting and receiving device is used for relevant distance measuring information [10]
Control and systems for autonomously driven vehicles	8126642	The system outlined includes operation control mechanisms having inputs and producing outputs to control operation of a vehicle. It also contains a self-contained autonomous controller configured to operate with a variety of different sensors and different operation control mechanisms. [11]

Of the five described, each patent either fits within the category of sensor technology or electronic control units (ECUs) for vehicles. These topics will be important to keep in mind when developing a design and will assist in ideation in the upcoming project phases.

2.3 Relevant Technical Papers

The most relevant technical papers for our design challenge come from previous Cal Poly senior projects that focused on individual components of the system we are trying to design. A small-scale intelligent vehicle (SSIV) design platform was completed in 2017 by three mechanical engineering students under the supervision of Dr. Charles Birdsong [12]. The next year, Dr. Birdsong headed another group that designed a Simulink interface to follow up with motion control and analysis for the intelligent vehicle platform [13]. Finally, an ongoing senior project by the Daimscale team finishing up at the end of Fall Quarter 2018 has been tasked with replicating a Daimler Trucks NA semi-truck at a 1/14th reduced scale [14]. The research and findings of these three project teams will be referenced extensively during our design process.

2.3.1 Actuators

The driving force of the vehicle is provided by actuators integrated in the truck's chassis. In general, an actuator is responsible for controlling some physical mechanism behind a system. Two major actuators in small scale vehicles are motors for providing power transmission into linear motion and servos for providing control over steering. The Daimscale team working on the chassis for the truck has selected a Tekin RX8 ESC (electronic speed control module) and 1550kv 1/8 motor, as shown in Figure 2 [15]. The motor, coupled with an SMC 14.4V lithium-polymer battery, is responsible for the motion of the vehicle. In addition, two different types of servos have been specified – The Savox 1270TG for 360 oz-in torque to control and maintain steering angles, and the Savox 0220MG for shifting gears and actuating a "5th wheel" that locks and unlocks the trailer hitch. The previous team has already bought and integrated the previous components into the chassis, but our team has some discretion in switching out these parts for a more complete and optimized design. Actuator control is directed by the computing platform, which is informed by sensor data and processed using whatever software architecture we develop. Each of these are further explained in the following sections along with papers describing the current state of the intelligent vehicle field.

Figure 2: Motor and motor controller specified by the Daimscale team [15].

The motor and ESC are both already integrated into the vehicle chassis and are capable of receiver commands from the RC controller.

2.3.2 Sensors

To meet the autonomous capabilities that our customers require, we will need to incorporate a suite of sensors that gives the truck awareness of its physical surroundings. The Daimscale team has already specified a wheel-speed sensor and an inertial measurement unit (IMU) to give information about the speed and direction of the truck. For object detection, however, some sort of proximity sensor(s) will need to be added into our design. The ProgreSSIV team incorporated cameras for computer vision, and ultrasonic range finders for autonomous purposes [12]. These are possible options for our design when moving forward, as they will help the truck understand more about its surroundings. There are also other sensor options that should be considered before a final design is chosen, including LiDAR and RADAR.

2.3.3 Computing and Software

In addition to the aforementioned sensors and actuators, a Raspberry Pi 3 B+ microcomputer and a Teensy 3.6 microcontroller are available to handle the computing platform of the vehicle, as shown in Figure 3. These components are not finalized and are merely what previous senior project teams have used in their design. In depth analysis for specifying the components of our design will be described in Section 4.

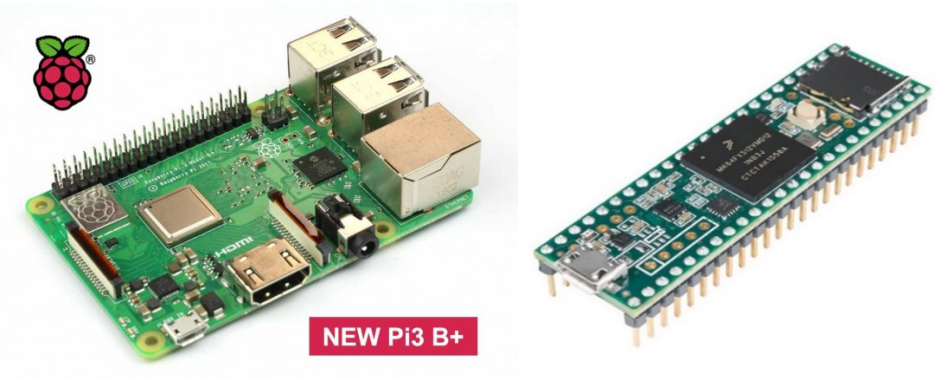


Figure 3: Raspberry Pi 3 B+ (left) [16] and Teensy 3.6 (right) [17] embedded controllers

Some background into previous senior project designs, however, is enlightening in understanding more of what we will be working with. The primary reason for specifying both the Raspberry Pi and the Teensy boils down to the fact that one microcontroller does not run fast enough to both implement effective controls over the system and communicate with all of the sensors and actuators simultaneously [14]. The Pi has its own operating system, which prevents it from being utilized as a true real-time system that is required for satisfactory motion of the vehicle. By offloading communication with the sensors and actuators to the Teensy microcontroller, the Pi is able to have enough resources to effectively run the control loop algorithm since it only needs to communicate with the Teensy to acquire relevant data. Information is transmitted between the two computation modules through an SPI protocol. The Raspberry Pi's Wi-Fi capabilities were utilized for ease-of-use when sending new control software. The Pi's capability for image processing is an additional beneficial aspect that may be utilized in the future.

2.3.4 Current State of Autonomous Driving

Information from previous Cal Poly senior projects is crucial to the design throughout this project, but there are plenty of other technical papers outside the university that give more generalized information on the contemporary state of autonomous driving, and the platforms that underpin vehicles' autonomy. We will be using these papers to inform us of the general methods and systems in use today.

The Society of Automobile Engineers (SAE) defines five levels of autonomous driving. An article by Victoria Transport Policy Institute breaks down the different autonomous capabilities into six different categories ranging from Level 0 (no automation) to Level 5 (full automation) [18].

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 4: Levels of autonomous driving ranging from 0 (no automation) to 5 (full automation) [18]

Each level incrementally relieves the human driver of certain actions, and instead places that responsibility on the vehicle's systems. Naturally, as vehicles progress towards higher levels, the systems become more complicated, relying on robust software and hardware components for accurate and reliable control of the vehicle.

A research paper by Loughborough University's School of Civil and Building Engineering, and their School of Aeronautical and Automotive Engineering breaks down the fundamentals behind one common outlook of what is actually happening to make a car autonomous [19]. The paper's basic description of the motion planning behind on-road driving consists of "(1) finding a path, (2) searching for the safest maneuver, and (3) determining the most feasible trajectory." The path that the autonomous system identifies is based

on incoming sensor data and represents a geometric passageway that the vehicle must follow to reach its destination while avoiding potential collision hazards along the way. Once sensor data is taken in, maneuver-planning calculations are computed to generate a "high-level characterization of the motion of the vehicle." From here, trajectory planning occurs in a real time setting, where additional incoming sensor data is considered to optimize specific parameters such as navigation comfort, avoiding obstacles and obeying any traffic laws.

Since the main objective of Daimtronics is to create a platform for future implementation of intelligent vehicles, the software that builds the platform is of great importance. Generally speaking, there has been much success in incorporating open source software into autonomous vehicle design. One of BMW's primary affiliates, BMW Car IT GmbH, specializes in researching and advancing the state of software development for self-driving vehicles [20]. The group suggests a combination of two different open source software platforms: AUTOSAR (Automotive Open System Architecture) and ROS (Robotic Operating System).

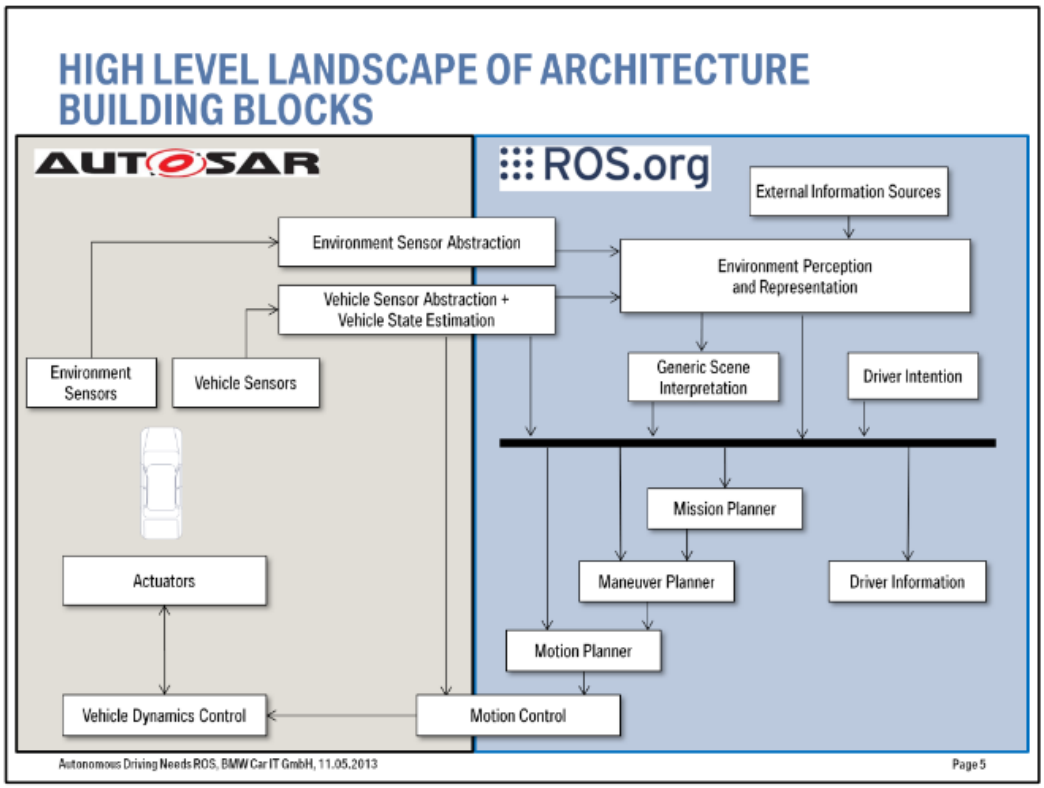


Figure 5: System Diagram for an Autonomous Driving Platform from BMW Car IT GmbH [20]

Software from each open source platform preside over different areas of the autonomous vehicle, with ROS taking over most of the computing responsibilities for decision making and path finding, and AUTOSAR taking over a more direct interface to the sensor and actuator hardware installed on the vehicle. It is important to note that the presentation of this material is predominantly for investigative research and design ideas and does not go into detail about the specific implementation of these platforms.

Nonetheless, the above diagram gives good idea of how open source software may be able to be incorporated into the proposed project.

In the development our scale platform we will have to emulate the drive characteristics of semi-trucks. In a report titled *Cyber-Physical System Based Optimization Framework for Intelligent Powertrain Control* published in the SAE International Journal of Commercial Vehicles by Chen et al. [21], an overall method is explained for controlling electric vehicles based off different performance characteristics. The paper describes how to optimize different drive variables based off whether the human driver selects sport, eco, or normal mode, shown diagrammatically in Figure 6.

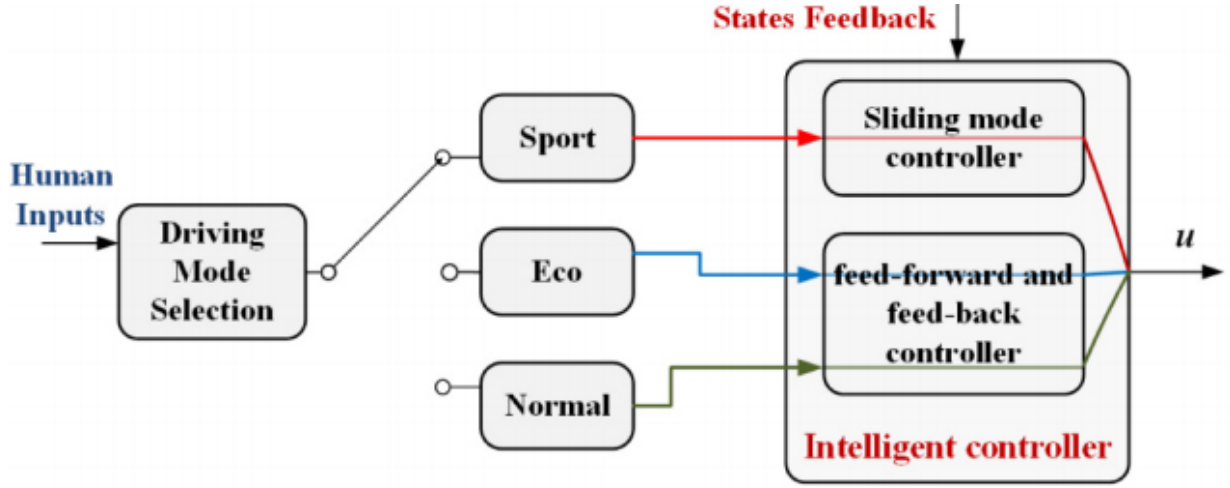


Figure 6: Controller selection diagram [21]

We will likely not be switching between different drive modes, but the mathematical framework developed may be useful in defining the appropriate cost functions to tune our controller's parameters and architecture to emulate truck dynamics. The paper is particularly appropriate since it deals with electric vehicles, which can be viewed as scaled-up versions of our RC-based platform. While not directly related to the intelligent vehicle field, this report provides additional foundation for executing vehicle control.

There exist many models and frameworks to describe the dynamics of vehicles and for how digital platforms should make driving decisions. A use case of these models is outlined in *Personalized Driver/Vehicle Lane Change Models for ADAS*, a paper published in the IEEE Transactions on Vehicular Technology journal by V.A Butakov and P. Ioannou [22]. Their paper mentions two kinematic models for executing lane change maneuvers, polynomial and sinusoidal, and goes into detail about the sinusoidal model, their choice for the system they developed. This kinematic model was used in conjunction with a stochastic model, the Gaussian Mixture Model, in order to create an Advanced Driver Assistance System (ADAS) lane-change framework that would tune itself to the preferences of the driver. Other stochastic methods mentioned were hidden Markov models, neural networks, fuzzy logic, Bayesian networks, and support vector machines. While our team is not intimately familiar with any of these models, we will be

keeping their existence in mind when deciding what types and what quality of data is necessary for the end-user to have when implementing these kinds of decision-making algorithms.

3. Objectives

The objectives stated in this section will be the driving force for our team and represents the overall philosophy going into this project, guiding the direction that our efforts will be focused.

3.1 Problem Statement

In order to increase safety and efficiency in the transportation industry, Daimler Trucks North America needs a way to enable interns, university-level competitors in autonomous trucking competitions, and general autonomous vehicle researchers to safely and accurately test semi-trucks outfitted with autonomous and driver-assistance capabilities on a scaled-down platform. To achieve this, we need to integrate electronics into an existing RC-scale semi-truck tractor/trailer chassis to allow new users to operate and learn from its capabilities such as vehicle dynamics, signal and data processing, feedback control and more.

Specific aspects of solving our problem include integrating the computing platform, controls, and sensors with the chassis while writing the drivers, middleware, and user interface for the platform.

3.2 Boundary Diagram

The boundary sketch in Figure 7 and the boundary diagram in Figure 8 outline the scope of this project from a more visual perspective.

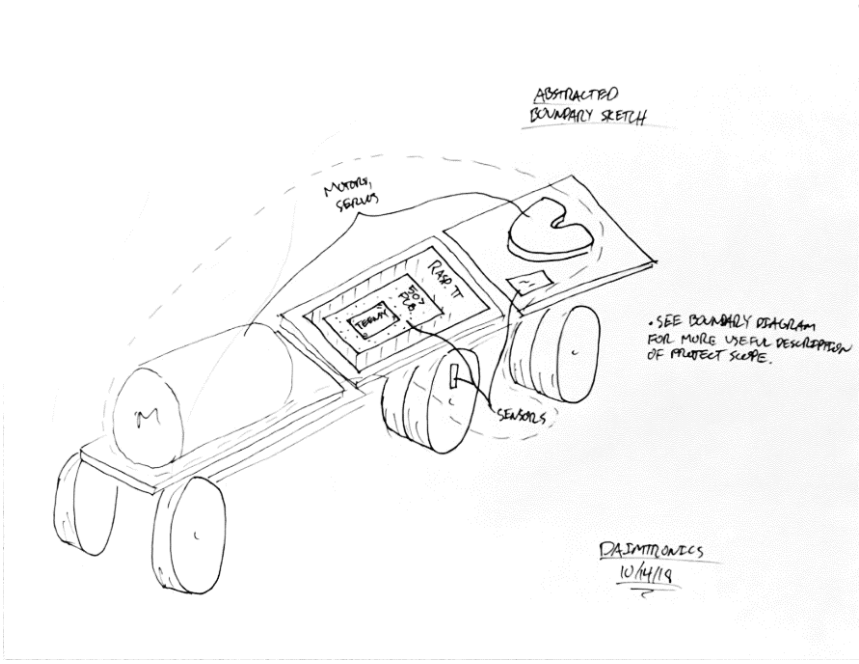


Figure 7: Daimscale -> Daimtronics boundary sketch

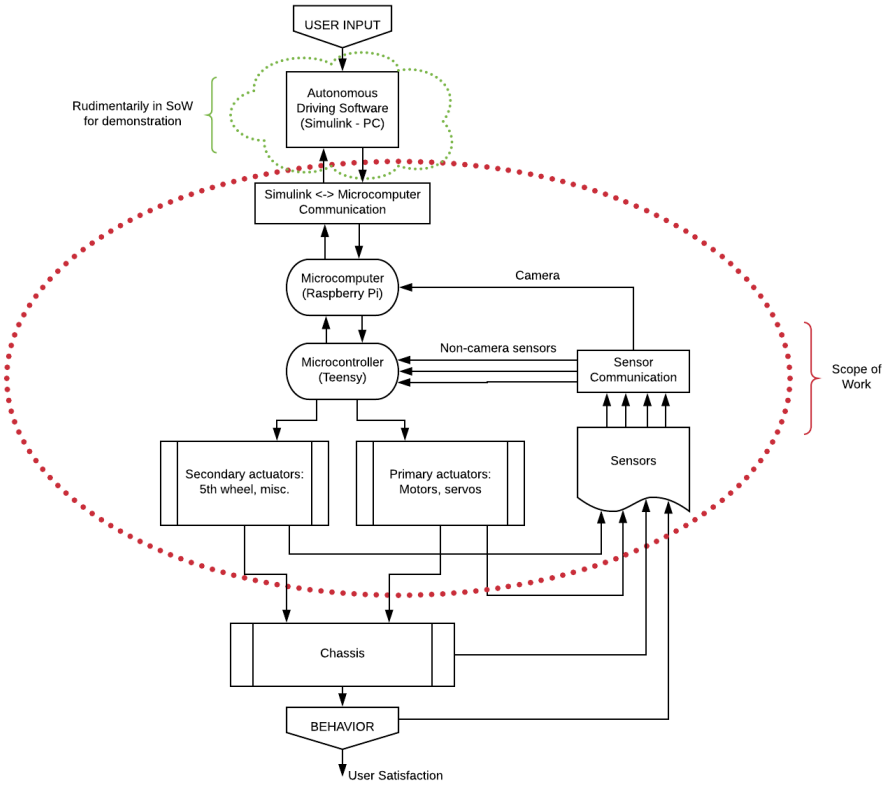


Figure 8: Daimtronics boundary diagram

All of the mechanical components should be completed in December 2018 by the Daimscale team, encapsulated in the 'Chassis' block in Figure 8. The autonomous driving software is going to be implemented by the users of this product and so is not included in our scope. However, rudimentary software will be developed by our team for demonstration and testing purposes. The primary scope of our work exists in integrating a computing platform, sensors, and actuators onto the chassis. As mentioned earlier, The Teensy operates on a custom motherboard to handle sensor wiring that will likely have to be redesigned during our project.

Evaluating the Pi, Teensy, sensors, and the RC motors from the previous senior projects for whether they are the optimal solution for our project is also included in our scope. If there are options better suited to the end goal of the project this will be evaluated versus the additional time required to implement alternate solutions and whether that additional time will decrease the quality of the final product.

3.3 Customer Needs List

Performing background research and communicating with our sponsors resulted in a set of customer needs that will need to be kept in mind during the design process. These needs are based on the three different customer audiences listed in Section 2.1: Daimler at the university level, Daimler researchers, and Dr. Birdsong of Cal Poly. A Quality Function Deployment (QFD) House of Quality contains the list of needs and wants on the left-hand side of Figure 8. The list covers requirements in vehicle dynamics analysis, autonomous capability and practicality for being involved in future projects

3.4 Quality Function Deployment Process

The QFD process was utilized to develop an appropriate set of engineering specifications and targets for this project. The full QFD House of Quality can be found in Appendix A. A portion of the QFD chart, the customer needs/wants versus our engineering specifications, is outlined in Figure 9.

Column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Direction of Improvement	▲	▼	◇	◇	▼	◇	◇	◇	▲	▲	▲	▲	◇	▲	▲	▼
WHAT: Customer Requirements (Needs/Wants) / HOW: Engineering Specifications (Tests)	Cost	Control Loop Time	Acceleration (0-top speed)	Deceleration (top speed - 0)	Setup time	Steering Curve and Max Angle	Sensors read in Simulink	Top speed	Max wifi range	Object detection range	Object detection angle	Battery Life	Software Application Layer	Documentation Rating	Voltage Safety Rating	Replacement Test Time
Semi-truck braking capability	▽	○		●				○								
Semi-truck acceleration	▽	○	●			○		○								
Semi-truck steering capability	▽	○	○			●										
Motion analysis of truck		○	○	○			●									
Practical in competitions	○				●			▽	○	○	○	○	●	○		○
Manual remote controlled driving	▽		▽	▽				▽	●							
5th Wheel Control		▽											●	▽		
Longitudinal Control		▽											●	▽		
Lateral Control		▽											●	▽		
Transmission Control		○											●	▽		
Wheel Speed Control		○	●	●			○						●	▽		
Vehicle Similitude		○					●									
Crash Resilience	▽		●	●		●		●							○	
Serviceability & Wiring Simplicity															●	●
Well-documented code													●	●		
Overvoltage protection	●														●	
Emergency & Out-of-Range Stops		○		●								▽				
Code optimized for fast execution	○	●								▽						
Reasonable time between recharge	●											●				

Figure 9: Customer Requirements and Engineering Specifications

The left column of Figure 9 lists the customer wants and the top row lists the engineering specifications. The specifications are outlined further in section 3.5. In Figure 9, the intersection of each row and column represents the degree to which the engineering specification addresses a customer want – either strongly (filled circle), moderately (unfilled circle), weakly (triangle), or not at all. The setup time will be critical for the platform being usable in competitions and in a course's lab. The capability for object avoidance and general automated driving abilities are important to every customer.

3.5 Specifications Table

An output of the QFD process is the specification table, shown in Table 2 below. The table lists the specification number (for referencing), the parameter description, requirement, tolerance, risk, and method of verifying compliance. Each parameter has a short description beneath the table that gives more relevant information.

Table 2: QFD Engineering Specifications

Spec #	Parameter	Requirements/Target	Tolerance	Risk	Compliance
1	Cost	$3,000	Max	M	I
2	Control Loop Frequency	20 Hz	Min	L	T
3	Acceleration Time (0-top speed)	4 sec	Max	L	T, A
4	Deceleration Time (top speed - 0)	4 sec	Max	L	T, A
5	Setup time	5 min	Max	L	T
6	Steering Curve and Max Angle	10% similitude deviation	Max	M	T, A
7	Sensors read in Simulink	Yes	Nominal	L	T, I
8	Top speed	20 mph	Min	L	T
9	Max Wi-Fi range	10 m	Min	H	T
10	Object detection range	5 m	Min	H	T
11	Object detection angle	180 deg	Min	M	T
12	Battery Life	30 min	Min	L	T
13	Software Application Layer	All devices	Nominal	M	T, I
14	Documentation Rating	8/10 rating	Min	M	T, I
15	Voltage Safety Rating	SF > 2	Min	L	I, A
16	Replacement Test Time	TBD	Max	H	T, I

Under compliance, ‘A’, ‘T’, ‘S’, and ‘I’ refer to the methods intended for determining compliance with the specification – Analysis, Testing, Similarity to Existing Designs, and Inspections, respectively. The tolerances are given as percentages of the nominal value, as a minimum or maximum constraint, or as a nominal constraint. Nominal constraints are either go/no-go design considerations with no specific tolerance. Under risk, ‘L’, ‘M’, and ‘H’ designate whether the specification is low, medium, or high-risk scaling with the difficulty of achieving the requirement.

3.5.1 Cost

Cost constitutes the total amount of money spent during our portion of the project. This will likely include the cost of sensors, may include the cost of a new computing platform, may include the cost of a custom PCB (printed circuit board) for electronic integration, and may include the cost of new motors. The budget is a maximum of $3,000, with a goal of minimizing costs and staying under this limit. The cost of sensors can become expensive depending on the performance required, hence the medium-risk designation. Cost compliance will be verified by inspection through the tracking of project expenditures.

3.5.2 Control Loop Frequency

The control loop frequency is the frequency at which it can update the vehicle's actuation signals, the inverse of the time it takes for the control algorithm to run once all the way through (the loop time). It impacts many aspects of the design as a loop time that is too long will interfere with how often sensors and actuators are updated, directly impacting the performance. Loop time can be minimized by writing efficient code and by using a computation platform with high enough performance that the loop time is sufficiently insignificant. The risk is low due to the high performance over cost ratio of modern electronics and our team's comfortability with fast languages such as C++. Compliance will be verified through testing; execution of the control algorithms will be measured by whatever task scheduling architecture is eventually implemented. The execution time of all supporting tasks, such as those to read sensors, are absorbed by this specification in that their data needs to be available for the control loop to run appropriately. A minimum frequency of 20 Hz has been chosen as the absolute lowest speed at which our controllers can run. It is likely we will be achieve higher frequencies.

3.5.3 Average Acceleration Time

Acceleration time as a specification is defined as the time to go from zero to its top speed. This is specified to be a maximum of four seconds, verified through testing and inspection of the selected motor's capabilities on the platform. Meeting this specification means we will have enough power on the platform to scale down to the appropriate acceleration curve that trucks exhibit without running into the physical limits of the system. Meeting this specification presents low risk due to the observed capabilities of the Daimscale chassis' motor. This will be verified through testing and by analyzing the vehicle's performance curves.

3.5.4 Average Deceleration Time

Deceleration time is defined as the time to go from top speed to zero. This is also specified to be a maximum of four seconds to ensure sufficient braking capability to match actual semi-truck deceleration curves. Meeting this specification presents low risk due to the observed capabilities of the Daimscale chassis' motor, which can be used to output decelerating torque. This will be verified through testing and by analyzing the vehicle's performance curves.

3.5.5 Setup time

The setup time of the platform is the time it takes to go from having a new high-level control algorithm developed to testing it on the platform where the platform was previously in nearby storage. This includes the time to attach batteries, connect the computing platform to the Wi-Fi, compile and download the user's algorithms, and then have the car performing the algorithm. We aim for this to be less than five minutes total to ensure that the platform is conducive to quick iteration. Verification will be conducted by timing the procedure ourselves.

3.5.6 Steering Curve and Max Angle

The steering curve and max angle of the platform are important aspects of maintaining similitude. The Daimscale team was responsible for selecting the proper wheel and steering mechanism, as well as adjusting them for proper similitude. Our team's focus will be maintaining the current steering curve and max angle established by the Daimscale team through proper servo control of the steering mechanism. A

maximum deviation of 10% from actual truck performance was determined to be sufficient to maintain similitude. This is a medium risk specification with compliance being determined by testing of the servo and steering mechanism, as well as analysis of the deviation.

3.5.7 Sensors Read in Simulink

This specification only requires that sensor data can be read both by the vehicle's algorithms and by the user within Simulink or whatever high-level environment is used for the vehicle intelligence. This risk is low due to previous senior projects sponsored by Dr. Birdsong, such as the ProgreSSIV team's, being able to accomplish this and our team's overall experience with Simulink. Simple testing and inspection are sufficient to determine if the specification has been met.

3.5.8 Top Speed

The top speed of the current platform is the $1/14^{th}$ scaled speed of a semi-truck driving at 65mph. The top speed was calculated by the Daimscale Team to be about 20mph [12]. The project should only be making improvements on the current platform and not limiting any work done on the platform; therefore, the goal is to meet this 20mph at a minimum. The risk is low for this specification due to the current platform already being capable of meeting and exceeding this value and thus limiting the top speed would be simple to do.

3.5.9 Max Wi-Fi Range

While the platform is in motion, it must still be able to communicate with Simulink through Wi-Fi. A max Wi-Fi range of 10m at a minimum is required to accomplish this. The risk is low due to the current state of Wi-Fi technology having vastly greater average ranges. Compliance with this specification will be verified through testing; preferably with the platform in motion to ensure it remains in communication with Simulink running on a PC.

3.5.10 Object Detection Range

The object detection range is the effective range of our object detection sensors. The object detection range can be maximized through careful selection of sensors that meet the minimum range required while still being fast enough for the microcontroller to process the data. A minimum of 5m allows for slightly over half a second between detection and collision when travelling at the vehicle's top speed. The time to collision is increased proportional to the degree at which the vehicle is driver under its top speed. This specification is high risk due to the performance quality 5m represents for sensors at our scale.

3.5.11 Object Detection Angle

The object detection angle is the counterpart to the object detection range. It specifies how wide of an angle (typically centered on the forward-facing centerline of the vehicle) that the sensors are able to scan. A minimum detection angle of 180° was determined to be sufficient for the purposes of this project. This is a medium risk specification, with its success determined by both placement of the sensors on the vehicle, and the types of sensors that are included in the design. Testing the range of object detection will be used to verify this specification.

3.5.12 Battery Life

The battery life determines how long the model semi-truck will be able to be continuously run without recharging the battery pack. A minimum target of 30 minutes was set for this specification, and this will primarily be measured through testing duration of the battery in the completed system. There is a low risk to this metric, with the only obstacle being cost of a battery.

3.5.13 Software Application Layer

Implementing a software application layer is one of the primary objectives of this project. This layer provides an abstraction between how a user will control the model semi-truck, and what actually goes on behind the scenes in the low-level software. To be complete, the software application layer must deliver an interface to read the data for each sensor that has data to be read, and to write different output levels to any actuators that will control the motion of the vehicle. This specification will be measured by how many of the devices are available to the application layer. The target is to have any device that is within the mechatronic system able to be read from or written to by future teams that need to make the vehicle more intelligent.

3.5.14 Documentation Rating

The documentation rating is a measure of how comprehensive and easy to understand our intelligent vehicle platform is. To be well documented our project must have consistent comments throughout the source code and thorough descriptions of application layer function calls. This metric will be tested towards the end of the project by subjecting students to use our platform, surveying the usability of our project from an applications standpoint.

3.5.15 Voltage Safety Rating

To have a robust system that is resilient to the electronics breaking from accidentally hooking up leads where they shouldn't be hooked up and from any spikes in system draw, the system needs to have some voltage protection. The electronic design needs to incorporate elements that prevent user error, such as voltage regulators, diodes, etc. The primary way of verifying that this specification is met, is analyzing the electrical circuit once all of the necessary protection has been put in place. This is low risk, because simply adding the electrical components will achieve a high voltage safety rating.

3.5.16 Replacement Test Time

The replacement test time specification measures how easy it is to replace a component of the system, should a component break down during use. The determining factors into this metric are how complicated the electronic wiring is, and how easy it is to fasten/unfasten different components from the overall system. Currently there is no target duration for this specification, but once more is known about the different components in the system, a maximum reasonable part replacement time will be specified. If the most complicated to replace parts can be replaced in under this time limit, this specification will be considered successfully met.

4. Concept Design

The aspects of our project's design can be separated into two categories: hardware and software. Throughout our design process we will be leveraging preexisting, third-party resources to expedite the development of our platform and to maximize the ease with which the platform can be repaired and updated in the future. In particular, there exist many open-source software frameworks that can be incorporated into our design to aid in creating a reliable architecture.

The primary hardware aspects of our design are the computing platform, motor, and sensors. The key aspects of our software are the middleware and the application layer. The interchangeability of each of these aspects means that we will be deciding on our solution for each aspect individually with our design being the sum of each optimized choice. This process is shown in the following subsections.

4.1 Hardware Component Design Concepts

The hardware components include: the computing platform, the actual computer or computers used to handle the vehicle's intelligence; the main motor driving the entire vehicle, the single-motor configuration having been chosen by the Daimscale team in order to replicate a semi truck's drivetrain; and the sensors on the vehicle to send environmental data to the computing platform. Other physical aspects of the vehicle that do not need in-depth decision analyses is the servo for steering, the servo for 5th-wheel actuation (see Section 2.3.1), the mounting solution for each sensor, and the mounting solution for the computing platform. Servos need only be integrated appropriately electrically and mounting will be developed as the final device models are selected and the Daimscale chassis is completed.

4.1.1 Computing Platform

The computing platform directs information between sensors, controllers, and actuators. It is the central component for the mechatronics behind this system, and each peripheral component needs to be compatible with how the computing platform behaves. Every device has particular communication protocols that it is compatible with, such as I2C, UART, and USB. It is imperative that the computing platform is able to handle the combined needs of all of the sensors and actuators in the design. For the purposes of our analysis we will be assuming that a mix of communication protocols will be necessary to satisfy the needs of each peripheral device so that the overall flexibility of the computing platform is all that we need to evaluate.

Appendix C contains a decision matrix regarding the selection of the computing platform for our system. There is a total of six different design options shown in the columns of the table, along with seven different criteria used to compare the design options. Included in the options are the Raspberry Pi 3B+ and Teensy 3.6 combination selected by the previous senior projects [12] [13]; an Nvidia Jetson TX2, used in the MIT RACECAR [4]; an Odroid and Arduino Uno combination; and a custom-built platform. Some of these options are shown below in Figure 10.

Figure 10: Arduino Uno [23], Nvidia Jetson TX2 [24] and Odroid C2 [25] (from left to right)

The Arduino Uno is a common embedded device that is used in many different hobby applications. It is relatively easy to set up, and there is plenty of online community help for debugging the device. Unfortunately, it doesn't have the most powerful computing resources. At a clock speed of 16MHz and only 2KB of RAM, there are other options that give us higher performance for prototyping this platform. The Nvidia Jetson TX2 lies on the other end of the spectrum; it is a powerful microcomputer that is currently being used for state-of-the-art AI programming applications. Unfortunately, the Jetson TX2 is too expensive for our purposes at a retail price of $599. Another competitive computing platform is the Odroid C2. This option is a more powerful version of the popular Raspberry Pi, at only a slightly higher price level ($50). There are two main drawbacks with this model: no built-in WiFi functionality, and fewer online resources for programming and debugging during software development.

Taking into account the weighting applied to each criterion, the selected computing platform was chosen to be the Raspberry Pi 3B+ in tandem with a Teensy 3.6 (shown previously in Figure 3). This setup gave high marks in processing power, time to implement, and the number of ports and pins - the most weighted aspects of the computation platform decision matrix.

The other aspect of the design for the computing platform is the method with which the computing units, sensors, and actuators physically interface with each other. We will be developing a custom printed circuit board (PCB) to act as the interfacing motherboard connecting the computation platform to the rest of the vehicle. Concerns regarding the motherboard discovered through experimentation with the SSIV vehicle are explained in section 4.4.1.

4.1.2 Motor

Our vehicle runs off of a single motor with the power transmitted through a three-speed transmission and a differential to the wheels. Since the lone motor is responsible for the locomotion of the entire vehicle and for maintaining semi-truck similitude, it is important for the motor to have enough torque and speed output capacity to satisfy dynamic demands requested by the control system the user develops. In addition to having sufficient spare output capacity, the motor has the potential to be a large portion of the overall cost of the platform – subsequently, minimizing excess motor cost is a high priority, both for our own project's budget and for the replicability of our platform. Other factors include available documentation of motor behavior and characteristics, the motor's efficiency and weight, and the time

required for us to integrate the motor into the existing platform. Appendix D contains the motor decision matrix evaluating the Tekin, Maxon, Turnigy, and Multistar motors shown in Figure 11.

Figure 11: Tekin [15], Maxon [26], Turnigy [27], and Multistar [28] motors (clockwise from top left)

The Tekin motor is the motor selected by Daimscale that is already integrated into the platform [14]. This is the top choice designated by the decision matrix; it has sufficient performance capabilities, is relatively low cost, and adds the smallest implementation burden to our team due to it already being in the chassis. The second choice was the Maxon motors, which are high performance motors used by the Microlaren and ProgreSSIV teams for their vehicle [12] [13]. The Maxon motors were much higher cost versus the Tekin motors when taking into account the motor drivers and had a higher time to implement both due to the physical integration necessary and due to requiring CAN communication and thus additional software to process its communication.

The Turnstar and Turnigy motors, while not as expensive as the Maxon motors, were not noticeably different than the Tekin motor and were more expensive. Combined with that, we would have to redo the motor integration for little benefit, so we decided against the Turnstar and Turnigy motors. The Tekin motor provides sufficient performance at a low enough price point and with minimal time burden.

4.1.3 Sensors

Sensor selection is a multistage process starting with defining the maneuvers we want the platform to be able to execute if required by the user. For modelling purposes, we determined that the platform should

be capable of performing forward object avoidance, side object avoidance, lane following, lane changing, object following, J-Turns [31], parking maneuvers, and trailer pickups. Each maneuver has a set of objects that the vehicle will have to be able to detect in order to safely perform, and each object has a range and angle requirement associated with it. Appendix F shows this process and lists the various angles and distances required to perform these maneuvers.

Concatenating the various sensor range requirements dictated by the object list results in Figure 12 below, our minimum required coverage areas for our sensor suite.

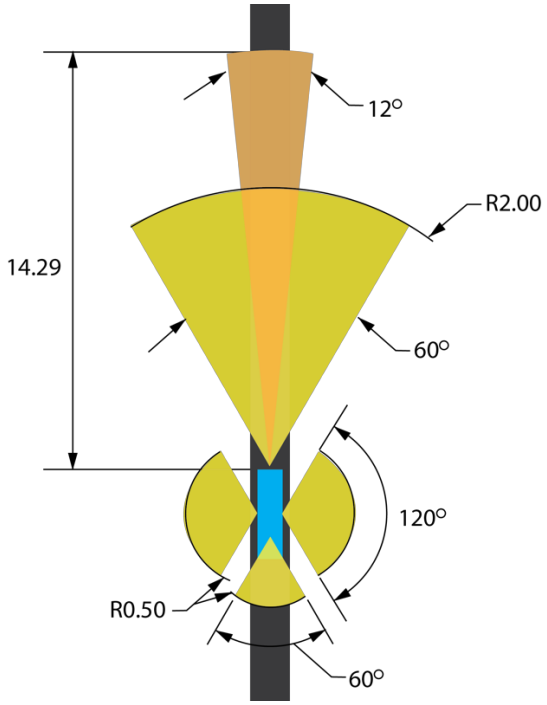


Figure 12: Sensor coverage areas with angles and the distances of the sensors shown in meters [29]

The primary coverage areas to be able to execute our required maneuvers are a narrow long-range sensor looking forwards, a wide medium-range sensor looking forwards, a wide short-range sensor on either side, and a wide short-range sensor on the rear. While the wide short-range side sensors, the wide medium-range sensor, and the rear short-range sensor were determined based off the geometry of the road and objects around the vehicle, it was necessary to evaluate the narrow long-range using a different approach. According to an analysis published in the journal Advanced Motion Control and Sensing for Autonomous Vehicles, a range of 200m was necessary for a forward facing sensor when a vehicle is traveling at highway speeds [33]. This range allows enough time for the vehicle to detect an object and react appropriately. Accounting for a $1/14^{th}$ scale vehicle gives us a required range of 14.29m. The 12-degree angle was determined based on analysis in the same textbook. Authors Li and Wang determined 12-degrees to be a good compromise between good obstacle coverage and avoidance of false and unnecessary measurements.

The specific sensor selection will be performed in the upcoming weeks as the particular class, model, multitude, and orientation of each sensor is developed according to our minimum coverage areas and available mounting positions on the chassis. This will depend on the final platform delivered by the Daimscale team. Other aspects that we will be minding as we develop the sensor suite specifications will be the vertical field of view for each sensor and the minimum latency for effective use as additional criteria for sensor model selection.

4.2 Software Component Design Concepts

As detailed in our introduction and problem statement, one of the primary objectives of this project is to create a platform that future students and researchers can use for a variety of vehicle algorithm applications. Part of this objective requires us to create a user-friendly application layer where end users can use our platform without being bogged down by the low-level details of reading and writing to different peripheral devices in the system. An effective design creates an abstraction between what the user wants the vehicle to do and what actually happens behind the scenes. Proper software design will satisfy the user-friendliness requirement of our customer wants and needs.

4.2.1 Middleware

In order to systematically decide between the many options available for middleware, we created a decision matrix on the most viable options, located in full in Appendix E. The different options evaluated are ROS (Robotic Operatic System), AUTOSAR (AUTomotive Open System Architecture), YARP (Yet Another Robot Platform), and custom-built middleware, where we write our own libraries to handle device abstraction and intercommunication. Each of the third-party options are open-source software platforms that have community support and are used for projects within the robotics and embedded systems realms.

Of the criteria, the three most important are time to implement, Simulink compatibility, and flexibility. The full decision matrix can be seen in Appendix E. Time to implement is important not only for getting our platform up and running, but also for future groups that may want to add additional functionality to a system based on our architecture. Being compatible with Simulink is an important contribution to the user-friendliness of our application layer (discussed in section 4.2.2). Flexibility is crucial for the possibility of end users wanting to add new hardware components on top of our final design without having to redesign the middleware itself. The middleware we use should not be a barrier to adding new peripheral devices to the system.

Our software design moving forwards will incorporate open-source code from ROS. It carries a low time-to-implement, access to a variety of sensor packages available in both C++ and Python, can smoothly interface with Simulink through the Robotics System Toolbox, has an active support community, and because ROS is so widely used as an open-source robotics platform there is plenty of community-written code for future additions and iterations of the sensor and actuator suite. Due to it being an actively maintained and utilized framework, future users of the systems will be less likely to have to deal with archaic and unsupported code if making changes to our system. An overview of the ROS architecture is given below in Figure 13.

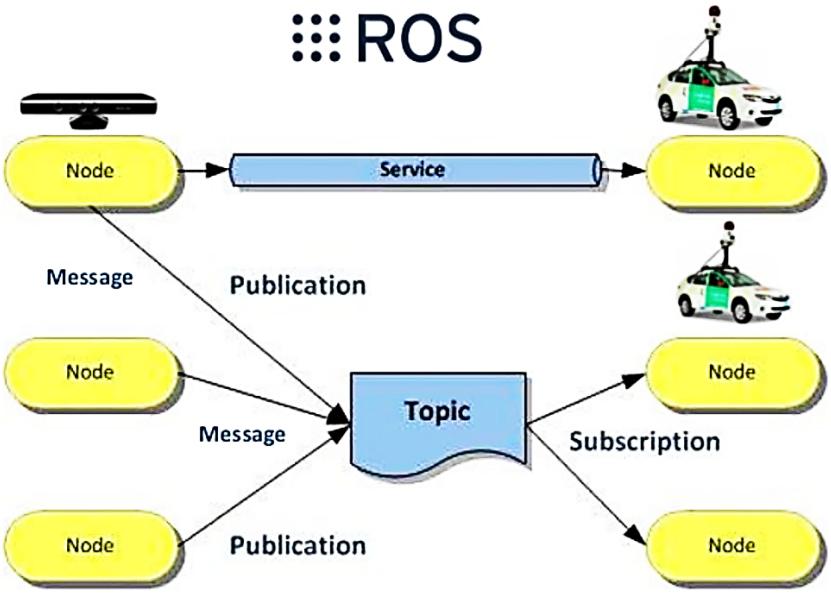


Figure 13: ROS architecture overview [30]

ROS's Wiki page gives a brief overview of the implementation of ROS [31]. The framework is based around different processes ('nodes' in ROS terminology) that each host an executable program. Groups of nodes exist within different packages that have commonality based on attributes of the system. When necessary nodes communicate with each other through ROS Topics at runtime. Nodes can "subscribe" to topics that belong to other nodes, allowing for the organized transfer of information from one part of the system to another. This is a very high-level overview of the inner workings behind ROS, and we will familiarize ourselves with more pertinent details as we begin software development and implementation.

4.2.2 Application Layer

Above middleware is the application layer of our software architecture. This is the layer that the user interacts with, the place where abstracted autonomous algorithm designs can be constructed and tested. The ProgreSSIV team developed a Simulink file that gives users space to create their own control systems by providing the outputs of every sensor on their vehicle and inputs to each of the actuators. How to wire everything in-between is up to the user. This is the overall scheme we will be implementing. Figure 14 below shows a simplified control model in the ProgreSSIV interface.

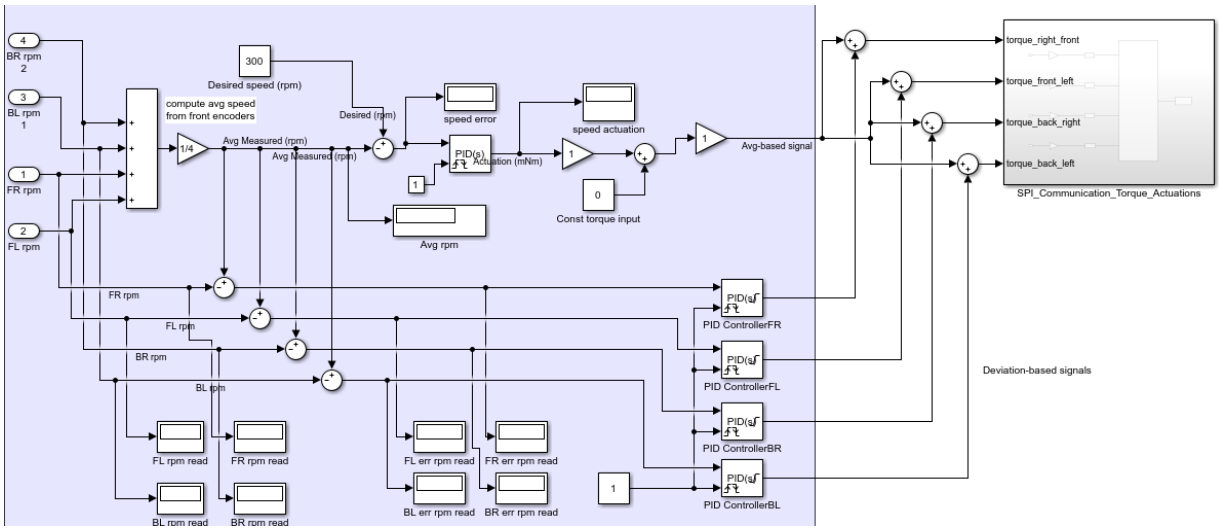


Figure 14: ProgreSSIV example algorithm

The file was provided by Dr. Birdsong with an initial PID average speed controller. We added an additional signal layer varying the actuation signal sent to each wheel individually to attempt to keep wheel speeds even. This is not an optimized or effective control example, but it showcases the ease with which a user can quickly implement and test an idea. The left side of the diagram pulls the rpm readings of each wheel and the right side is where torque signals are sent to each wheel's motor.

The design of our application layer will prioritize ease-of-use and provide real-time displays of the vehicle's sensor readings and actuator output signals.

4.3 Potential Risks

Through experimentation with the SSIV platform developed by MicroLaren and ProgreSSIV and discussion with sponsors, we have identified several key risks we need to mitigate with our design. Appendix G contains a checklist with some of the hazards that are associated with our design. In general, the major concern is that this project involves an object with a significant amount of mass that is design to be able to travel at speeds up to 20 mph.

4.3.1 Loss of Communication

In the event that the semi-truck loses connection with Wi-Fi, the computing unit will most likely maintain the motor and servo outputs that were present just before the disconnect. Unless we incorporate an automatic shutoff feature to our design, the vehicle will be out of control, posing a risk both to the vehicle itself, and also any object or people around it. The ProgreSSIV senior project team implemented a "dead man's switch" in their intelligent vehicle design that limited any actuation of the vehicle to only when a user was holding an RC controller trigger down. This ensures that active attention is required in order for the vehicle to operate and is a safety feature we will be replicating.

4.3.2 Battery Spontaneous Combustion

The class of batteries we will likely be using are LiPo (Lithium polymer) batteries due to their high-performance characteristics and being the battery type already integrated in the Daimscale chassis. The

risk associated with this class of battery is their propensity to catch on fire when mishandled. To ensure the user's and vehicle's safety we must we design robust electrical connections and keep the batteries enclosed and protected in the event of a crash. Our user manual will also reflect safe handling practices for future users.

4.4 Design Challenges and Unknowns

There are several aspects of our design that are particularly important to the robustness of our finished product that we will be focusing on moving forward, described in the sections below.

4.4.1 Electrical Power Concerns

The major issue discovered while experimenting with the ProgreSSIV vehicle was the instability of the electronics. Their platform utilizes a single power supply, a single battery supplying power to all of the motors, servos, microcontrollers and sensors. This simplified the overall power system, but insufficient filtering and isolation between the motor and computing portions of the circuit resulted in enough noise to cause the Raspberry Pi to crash and restart. The power draw from the servos and motors would occasionally starve the microcontrollers of their required current at random, often frequent intervals. This is a major detriment to the user-friendliness of the platform and is a problem we will be addressing in our design through increased isolation between the power supplies, through both our redesigned motherboard and separate power supplies.

4.4.2 Sensor Specifics

As outlined in section 4.1.3, our current focus with regards to sensors has been accurately defining the coverage areas required. The entirety of our remaining component specification work is in finding appropriate sensor models and developing a layout that will satisfy the requirements we showed in Figure 12 at high enough performance such that users have access to useful sensor data. The ease with which each sensor integrates into the rest of our platform will be taken into account in our selection in tandem with our motherboard development, described below in section 4.4.3.

4.4.3 Motherboard Design

Developing a redesigned motherboard will require a moderate amount of research into analog noise filtering techniques to ensure we have a robust design that mitigates the electrical effects discussed in 4.4.1. Sensors and electrical devices in general have several options for inter-device communication and the compatibility of each sensor with our computing platform can depend on the flexibility that we design into the motherboard. Further, developing a custom PCB requires a significant time investment in reading through manufacturer datasheets and finding components that are compatible and equally robust. The details regarding our board design will likely be completely known only after going through the design process once and testing prototype boards.

5. Final Design

Our concept design section specified major choices that are persisting into the final design, including the two microcontrollers of the computing platform, the motor and the ROS infrastructure that will control the tasks running on the Raspberry Pi. Our final design section rounds out the entire system with components and assemblies that have not been covered in the concept design. Major points of the final design are the sensor selection for object detection, the motherboard and electronic connections, physical mounting that will ground each of the components in the system to the chassis, further details behind the ROS code on the Pi, and the proposed design to the Teensy task code structure.

5.1 Sensor Selection

In the concept design portion, we laid out basic guidelines to follow for meeting our design specifications in terms of sensor functionality. Since PDR, we have decided to go with a more intuitive approach to the sensors and not rely on scaling the sensors down to the same 1/12th ratio that is seen in the semi-truck chassis. This decision was suggested by our sponsors due to the physical behavior of sensors not scaling the same way that the mechanical components can scale. For example, an ultrasonic range finder (URF) cannot accurately scale down the signals transmitted by the device to be deemed a truly scaled down for modelling purposes. Based on this, we are treating the semi-truck system as a general robot that requires the ability to sense its surroundings, without caring about any similitude constraints to the full-scale semi-truck.

5.1.1 360° Spinning Lidar

We have selected Slamtec's RPLidar A2M8 360° Laser Scanner for general midrange object detection (Figure 15). This sensor will be mounted at the front of the vehicle for maximizing its field of view. Embedded in the device is a motor that controls the frequency with which it spins, allowing for variable 360° scan rates between 5 and 15Hz. When recording, it has a sample frequency (which is the number of individual light pulses emitted) of between 2000 and 8000 Hz. Each sample can detect objects between 0.15 and 12m. While the full 360° nature of the lidar is great for the field of view, the relatively low refresh rate of the semi-trucks surroundings (max 15 Hz) is a problem. In the next subsection we incorporate a solution to this concern.

Figure 15: RPLidar A2M8 360 Scanner [34]

A point-cloud, as shown in Figure 16 is generated through recording data while the sensor is spinning. Each point represents a single sample, and the cloud is updated after each 360° scan. RPLidar models, including the A2M8 have access to Slamtec's software for processing point cloud data. Instructions overviewing the software development is given in the RPLidar SDK guide [35]. which will be instrumental down the road during development of the data processing protocol. This sensor will be hooked up to the Raspberry Pi, which is running ROS, through an RS-232 communication protocol. A benefit to using the RPLidar is that there are existing ROS libraries for the sensor that have been tried and tested.

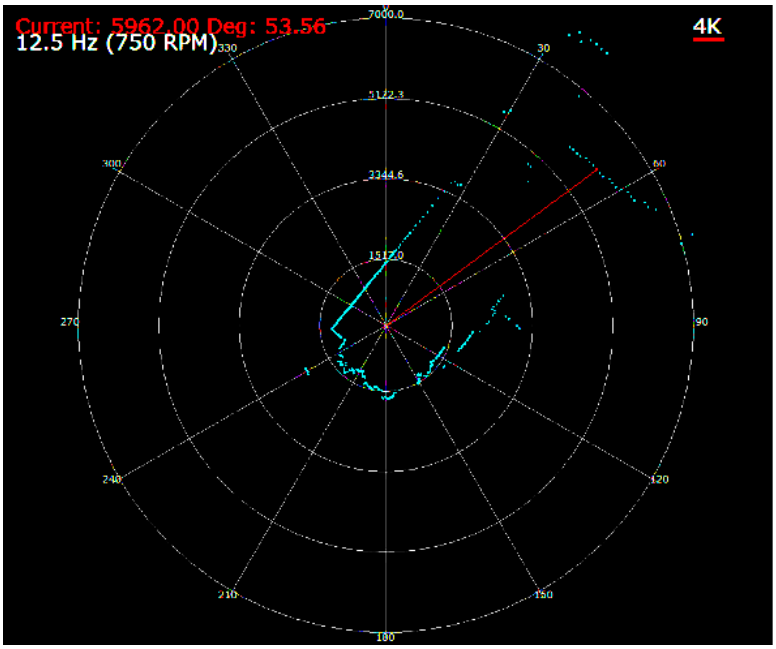


Figure 16: Point Cloud generated by the spinning LIDAR [35]

5.1.2 Long Distance 1-D LiDAR

A forward facing, one-dimensional LIDAR supplements the spinning LIDAR by allowing a much higher refresh rate to any object in front of the vehicle. There is one main concern with the low frequency of the spinning lidar: at max speeds of the semi-truck an object in a collision path with the vehicle will not be tracked very well. It will take the RPLidar a full rotation before the object's location is updated. This is not ideal when the semi-truck is moving fast and is in jeopardy of running directly into the object.

A LIDAR-Lite 3 was specified to keep track of objects directly in front of the vehicle, especially at longer ranges (up to a potential range of 40m). This device is able to read the distance of objects through either I2C (inter-integrated circuit) or PWM (pulse-width modulation) communication channels. Its configurability allows us to adjust settings that dictate accuracy, operating range and measurement time. Finally, the device is a low power sensor, only requiring 85 mA and 5V during operation. The Lidar-Lite 3 is shown in Figure 17 below.

Figure 17: LIDAR-Lite 3 High Performance [36]

The end of the cable comes as bare wires which we will be able to plug directly into pin sockets on the motherboard. The documentation for the Lidar Lite 3 specifies an additional resistor if using PWM-based communication and two additional resistors with an electrolytic capacitor if communicating over I2C. Because these components are sensor-specific, they will not be integrated into the motherboard and will instead be built on an adjacent mini breadboard. See Section 5.2.5 for more information.

5.1.3 Ultrasonic Range Finders (URF)

To round out the sensors dedicated to object detection, we have decided to equip the semi-truck with a suite of HC-SR04 Ultra01+ URFs [37]. At this stage in the design, we are planning on having five of these URFs to for illuminating any blind spots that the 1-D and spinning LiDARs do not cover. The three main areas are directly behind the tractor (where the trailer will hitch up to the fifth wheel) and the back left and right corners of the tractor-trailer combo. The spinning LiDAR is not able to see these areas due to

being located at the very front of the vehicle, where the semi-truck cab obstructs some of the field of view to the rear.

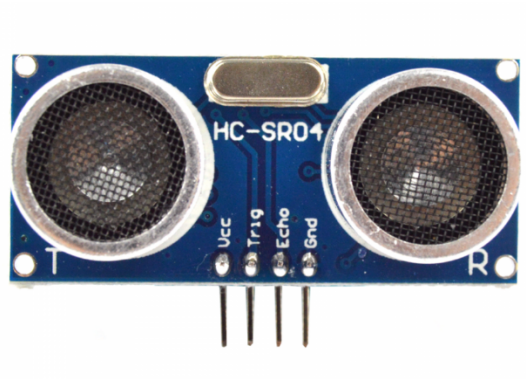


Figure 18: HC-SR04 Ultra01+ Ultrasonic Range Finder [37]

The URFs that we have selected have a range of 2 to 500cm, and an effectual angle of 15 degrees diverging from the sensor itself. A resolution of 0.3cm is achievable with this sensor. One thing that we must consider with these URFs is that they do not specify what angle an object is detected at; the only data returned from the sensor is the distance to the nearest object in the field of view. Data is transmitted through Trigger and Receive pins to calculate the distance to the nearest object as specified in the product manual [37].

5.2 Motherboard

The goal of the motherboard design is to provide a reliable common interface for the computing platform, sensors, and actuators that mitigates the issues seen on the SSIV platform (random restarting of the computing platform), provides an organized layout of the electrical components, and is easy for users in the future to use and adapt to their needs. Drawing #6 in Appendix K contains the full schematic for the motherboard.

5.2.1 Overview

The motherboard is powered by two batteries, one for the motor side and one for the computing side. Components can be powered from either power circuit depending on their noise level and to balance the current draw on each side. There are terminals for devices to be powered and pin sockets to interface with the Pi and Teensy GPIO (general-purpose input/out) pins. Additional pin headers are on the motherboard for interfacing with the RC receiver so that we can execute control from the RC controller and switch the platform between manual and automatic modes. We will be providing mini breadboards around the motherboard in the computing package to allow for miscellaneous through-hole components to be integrated into the board if required by a particular device to balance flexibility and platform simplicity. As an example, our linear Lidar requires an external capacitor and resistor which are unique to its model.

5.2.2 Power

See Appendix H for a simplified diagram of the power routing. We chose to have the batteries for both the computing and motor power circuits be identical models to remain interchangeable for ease-of-use and because both the motor and computing sides of the power circuit have high current requirements. The computing battery connects to a cable terminating directly on the motherboard, while the motor battery connects to a cable that is spliced between the motor and the motherboard. This allows the current going to the ESC, which powers the motor, the RC receiver, and the steering servo through the receiver, to not have go through the motherboard, vastly decreasing the current rating requirements for that particular portion of the board.

Both the computing and motor power circuits utilize an LM2678 switching voltage regulator [38] to step down the 14.8V coming from the batteries to 5V. The Teensy, Pi, and all of our selected sensors as well as most servos run off of 5V while only the signal lines operate at 3.3V with our current selection of sensors. Both the motor and computing sides of the power circuits have the main 5V power supply rail and smaller 3.3V supply rails, stepped down using a 3.3V Zener diode in parallel with the rail, to maintain flexibility. The 5V rails are capable of pulling any current up to the regulators' capacities while the 3.3V rails are each limited to 150mA by the diodes. The computing-side power circuits include additional filtering circuitry to create a power supply appropriate for sensitive devices and to supply analog reference voltages at 3.3V and 5V for our and any future analog to digital converts (ADCs). The filtering circuitry is constituted by a ferrite bead, bypass capacitor, dampening capacitor, and dampening resistor with the components selected to filter out high-frequency noise from the Teensy and Pi, minimize DC current losses, and to avoid resonance with the voltage regulator's switching frequency. The ferrite bead's capability to filter noise falls drastically with increasing DC-current bias, with the particular bead selected so that up to an amp of draw can be pulled between both the 5V and 3.3V rails (with the 3.3V further limited to 150mA by its diode) without a major loss of impedance.

We chose a switching regulator rather than a linear regulator due to the high voltage differential and high current capacity we are aiming for. Switching regulators regulate current by rapidly switching the voltage on and off through capacitors, while linear regulators have to dissipate all of the excess power. This improves both thermal requirements and massively improves efficiency, from around 30% to 80%. The downside of a switching regulator is that the switching mechanism introduces a small amount of noise inherent to the output, but the capabilities of this regulator setup outweighs the small noise addition.

The worst-case reasonable scenario not including start-up current requirements is that all of the devices not powered through the ESC have to be powered off of the computing side due to limitations on the motor-side battery. Table 3 below represents the loading of this scenario plus running the 5th-wheel servo (which needs to be powered off the board) at an estimated stall current of 1A.

Table 3: Maximum current loading conditions

Device	Current [mA]	Qty	Total [mA]
Teensy [39]	100	1	100
Pi [40]	1000	1	1000
Pi peripherals [40]	1200	1	1200
IMU [41]	13	1	13
Wheel speed [42]	25	4	100
URF [43]	15	8	120
Linear Lidar [44]	135	1	135
360 Lidar [45]	500	1	500
5th-Wheel Servo [46]	1000	1	1000
Total			**4168**

The LM2678 has a maximum current rating of 5A, giving us a healthy 850mA margin even in the worst-case operation scenario. In reality, we will be moving many of these components to the motor side to balance the total power draw between the two regulators and to concentrate noisy components on the motor side. The computing modules and sensors generally have startup current draws higher than their running draw; supplying this increased load will be handled by splitting the devices between the two power circuits if necessary.

We utilized the TI Webench Power Designer [47] to spec the components required to complete the regulator circuitry and to simulate regulation performance. Figure 19 below shows the simulated transient response due to a load current step change between 0.5A and 5A.

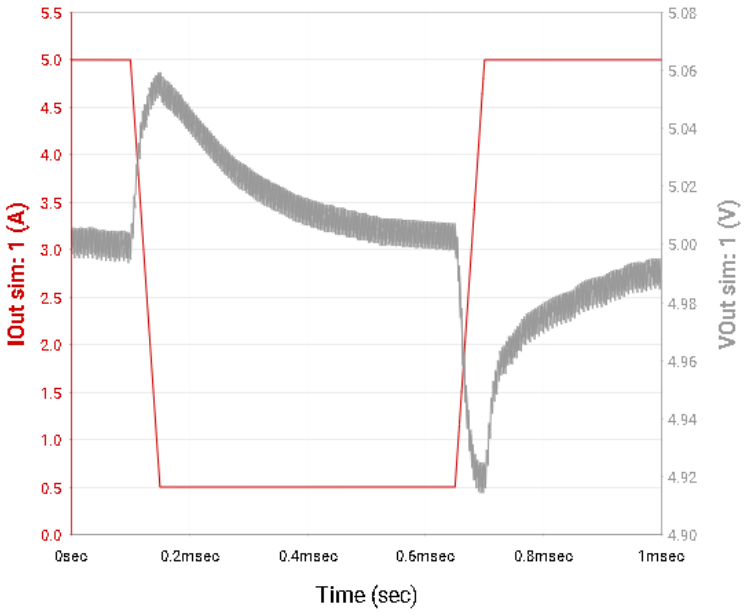


Figure 19: Webench-simulated LM2678 output voltage transient response

The output voltage increases 60mV and decreases 80mV in response to the massive current spike. The oscillations on the output voltage on top of the transient response is regulator's internal switching noise, shown in Figure 20 below.

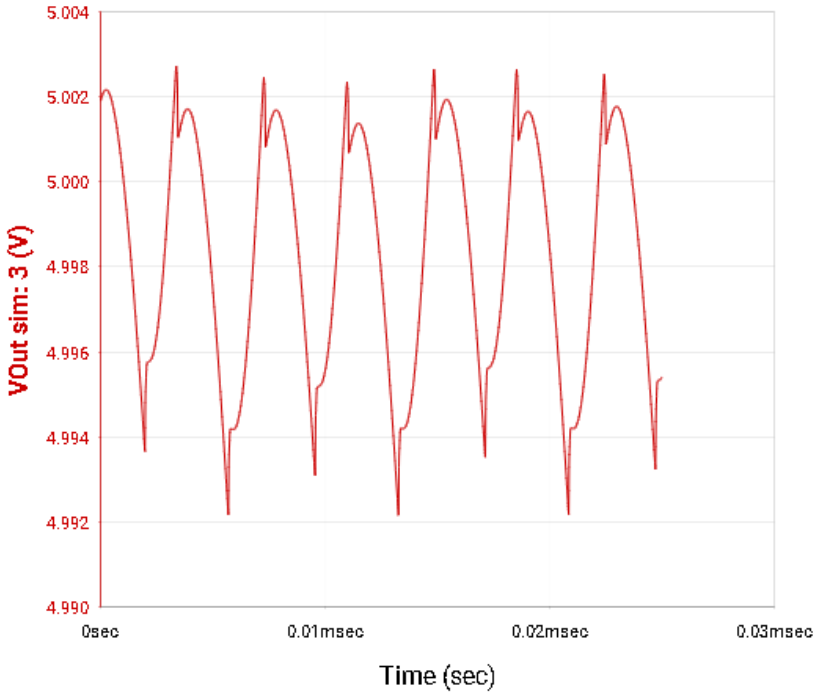


Figure 20: Webench-simulated LM2678 steady-state performance

The internal noise is responsible for +2mV and -8mV of the noise in Figure 20. It operates at roughly 260kHz, the rated frequency of the LM2678. The small amplitude is a negligible detractor with regards to the reliability of the system.

The recommended min/max voltage for the Raspberry Pi is 5.25V and 4.75V, putting the transient response well within the acceptable range [48]. The Teensy steps input voltage down to 3.3V for its own use with an LP38691 linear voltage regulator, which has a typical voltage dropout of 250mV, meaning it can take voltages down to 3.6V. The LP38691 can handle up to 10V, but the connected USB power switch is only rated for 6V (and there is not sufficient heat-sinking capability for higher voltages regardless), making this the maximum Teensy input voltage and making the Raspberry Pi the constraint in terms of electrical reliability.

The other consideration in addition to the LM2678's transient current response is how well it handles externally generated noise. Figure 21 below shows the simulated Bode plot from Webench.

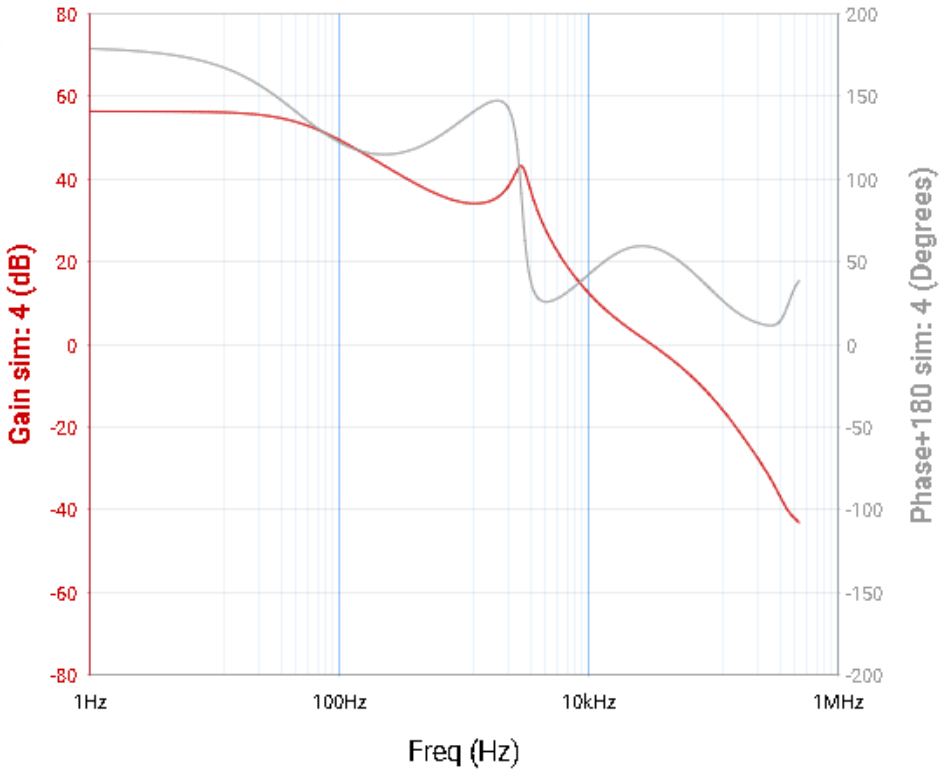


Figure 21: Webench-simulated LM2678 Bode plot

The conservative estimate of the electrical reliability will be the maximum output voltage spike (±250mV for the Pi, the constraining factor) divided by the sum of the following: the maximum output voltage decline due to a transient spike (Figure 19), the amplitude of external noise times the gain associated with the external noises' frequencies based on Figure 21, and the noise due to discontinuous operation, discussed in the next paragraph. This frequency response simulation only helps to justify our current design; it does not drive it. However, if it turns out that noise is still an issue on our motherboard, we will use these frequency response simulations along with measurements of the noise generated by our devices to design a new regulator circuit. For our current setup, the simulated crossover frequency (the frequency at which the gain equals zero, no amplification or diminishing) is approximately 32kHz; any noise with a frequency higher than this will be increasingly dampened.

The final factor affecting our electrical power reliability is the additional voltage ripple the regulator experiences when operating in discontinuous mode, which is where the operation of the regulator's internal MOSFET starts to create voltage jumps at low load. This is not simulated in Webench, but Figure 22 below from the LM2678 datasheet shows what we can expect.

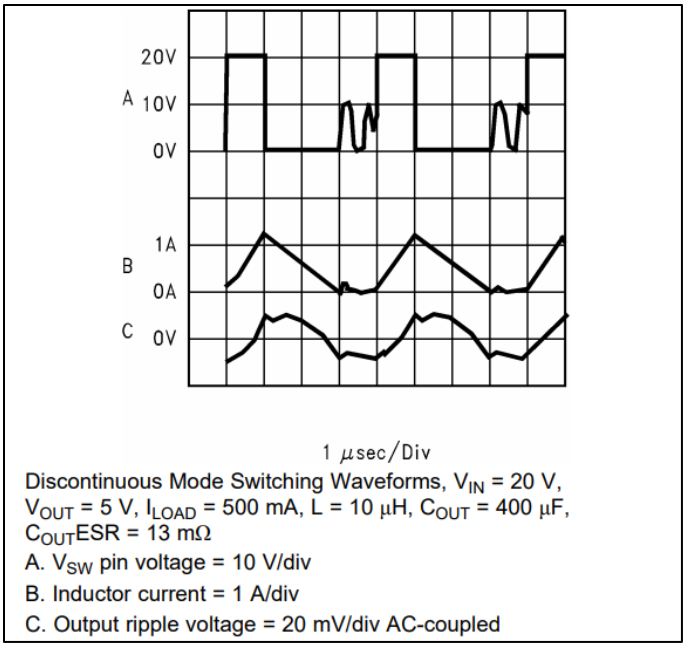


Figure 22: Expected discontinuous LM2678 operation [37]

The amplitude of the ripple is limited to around 10mV and the regulator enters this mode around 500mA. The parameters of Figure 22 above are close enough to our actual case to be applicable. While it's likely that we will enter this current range when everything is idling, 10mV is a minimal contribution towards the 250mV limit, particularly since there will be reduced noise at low current since components may not be operating. A final, conservative measure of the electrical noise safety factor will take this into account, however.

With regards to additional safety measures, the LM2678 includes an on/off pin. This pin on the motor-side regulator is hooked up to a GPIO port on the Teensy so that we can cut power to the sensors and reduce current draw if we find an appropriate scenario for this behavior to trigger. The computing-side regulator is not controlled with its on/off pin to avoid any possibility of undefined behavior when the power supply is receiving commands from its own load.

5.2.3 Pin Organization

The motherboard either directly utilizes or makes available for modular use all of the pins on both the Raspberry Pi and the Teensy, including the non-breadboard-able "pins" on the underside of the Teensy. The motherboard plugs directly into the Pi pin rack, leaving them unchanged but with traces to pins that we want hardwired to other areas of the motherboard, such as the serial communication pins going to the Teensy and 5V power supply to the Pi. The Teensy pins are broken out and lumped into sections based off their intended usage.

The Teensy's communication ports are available such that all of them can be used simultaneously if we have that many devices that require communication pins. While it is possible to rearrange which pins are used for which communication ports, it will not be possible to add more since the underlying alternate pin functions are all being utilized (there is a small, finite number of hardware-supported independent

communication lines allowed for each protocol). If alternate functions other than communications are required of a pin on a communication port, there may be a pin combination that allows an alternate pin to be used for communications on the same port, freeing the original pin for its other alternative function. However, the layout is such that this will be a remote last resort since it will require almost all of the pins being utilized. Table 4 below shows the pins that correspond to the chosen communication ports on the Teensy.

Table 4: Selected Teensy communication port pins [49]

Comm. Ports	Pin 1	Pin 2	Pin 3	Pin 4
Serial 1:	0	1		
Serial 2:	9	10		
Serial 3:	7	8		
Serial 4:	31	32		
Serial 5:	33	34		
Serial 6:	47	48		
I2C 0:	18	19		
I2C 1:	37	38		
I2C 2:	3	4		
I2C 3:	56	57		
SPI 1:	11	12	14	15
SPI 2:	51	52	53	54
CAN 0:	29	30		

The bolded ports are preferred ports that should be used before others in a given protocol class; these minimize the loss of other functionality. The number of the port corresponds to their designation on the manufacturer's Teensy pinout documentation.

Alternate functions other than communication protocols on the Teensy are PWM outputs, analog inputs, and pins capable of either. The several Teensy pins we designated as PWM-only are all used by the motors and servos and we did not make available to plug into for other uses. Analog inputs should be routed through the analog-only pins before using one of the dual-capability PWM-analog pins. Similarly, any general GPIO functionality should be handled by pins with no intended alternate functionality before being handled by one of the more useful PWM or analog pins.

A caveat on the Teensy's analog pins is that they are only 3.3V tolerant while all of our analog sensors (the ultrasonic range-finders) and many analog sensors generally output 5V. Rather than attempting to step down each analog pin individually, the motherboard has a dedicated eight-channel analog to digital converter (ADC) chip, an MCP3008, that communicates over SPI, with the SPI lines passed through a bidirectional 3.3V - 5V logic shifter. Since SPI is a bus-style communication protocol, the ADC chip and logic shifter combination allows for a higher number of analog inputs at no additional pin usage while allowing for separate dedicated 5V and 3.3V analog inputs on the motherboard.

5.2.4 Mode Switching & Actuation

To switch the truck between manual and automatic modes, we have to be able to send commands from the RC controller to the computing platform. Since the receiver only outputs PWM signals, we plan on having the Teensy read the receiver output PWM on the two auxiliary channels and program the RC controller to modulate the PWM on those channels, one for the dead-man switch and one for signaling what mode to operate in.

RC motor and servos use a modified, roughly standard style of PWM signals as an artefact of their prior signaling method, Pulse-Positioning Modulation (PPM). Instead of having a duty cycle going from 0-100%, the receiver sends a pulse between 1ms and 2ms long every 10ms, limiting the range of duty cycles to 10-20% and with a set frequency of 100Hz. We hooked up the outputs of our receiver to an oscilloscope to empirically measure the signals. Figure 23 below shows the steering servo signal.

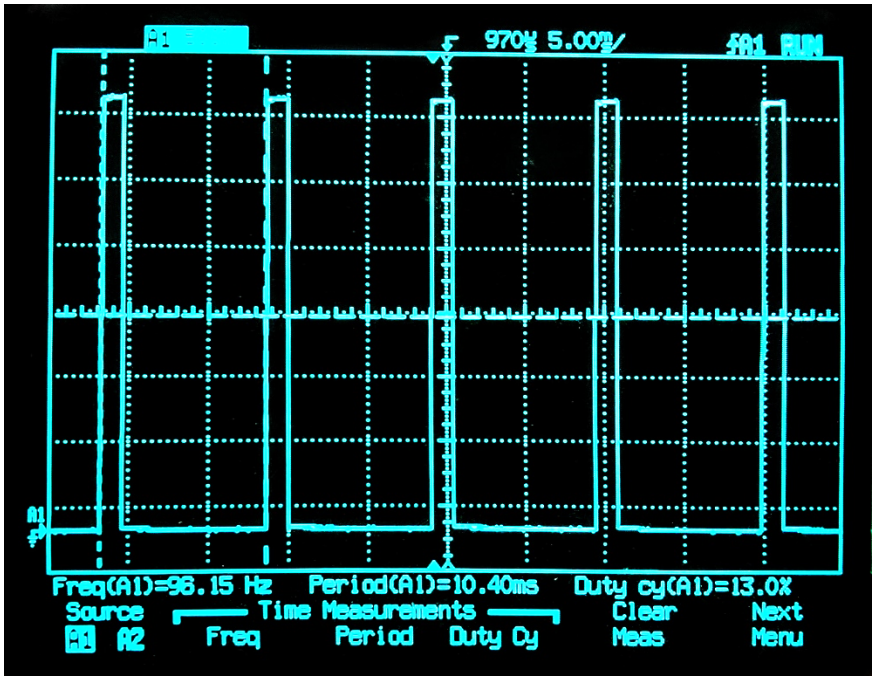


Figure 23: Oscilloscope capture of RC receiver steering signal

The outputs we observed at maximum input on the RC controller never reached the 1ms and 2ms limits of the standard. We believe this is due to the controller limiting output itself rather than the components using a different standard. We found that the Daimscale team further limited forward motor output to 90% of its “max”, while steering can be controller-commanded up to 120%, which was still beneath the 2ms limit of the standard. We will impose whatever limits are appropriate to the output in software for automatic mode, keeping in mind that it’s likely we can extract slightly more range out of the motor and servos than the RC controller does.

To hijack the PWM signals going to the motor and servo for autonomous mode, we are using single-pole double-throw (SPDT) solid-state relays (SSRs) triggered by a GPIO pin on the Pi and fed PWM signals from

the Teensy. The signal is sent from the Pi rather than the Teensy due to the Pi's higher current draw limit on its pins.

SPDT relays switch a common connection (the pole) between two other connections (the throws), in this case the input to the motor/servo and the PWM outputs of the receiver and Teensy, respectively. We selected solid-state relays due to the high reliability, smaller footprint, and simpler circuit design. For SPDT relays, SSRs also happened to be cheaper. Figure 24 below shows the circuit diagram of our chosen relay, the LCC110.

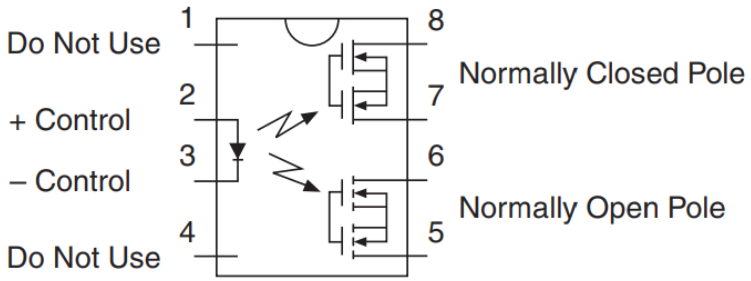


Figure 24: LCC110 SPDT SSR circuit diagram

The other option versus using relays to switch between signals was to route the receiver PWM output directly to the Teensy, have the Teensy read it and then recreate it with all of the motor and servo inputs being directly connected to the Teensy. We chose not to do this in case the resolution of receiver outputs is poor, which is likely due to the modified-PWM style of RC components mentioned above and possible performance constraints on the Teensy. Using an intermediary relay preserves maximum capability in both manual and automatic modes since there are no issues generating PWM outputs on the Teensy. This significantly lowers the performance requirements of the Teensy in reading the receiver's output PWM's since we only need to be able to distinguish between a moderate number of signals.

To ensure safety, the default (normally-closed) position will be manual mode. If the RC receiver and controller lose contact the system will no longer read the dead-man switch and revert to manual mode, which has no signals being sent to it.

5.2.5 Modularity

The motherboard is meant to be generic with respect to the sensors that are attached to it to allow for flexibility both for our team and for anyone who works with the platform in the future. In the event that small components such as resistors or capacitors need to be added to allow a particular sensor or device to be utilized, we will be integrating mini breadboards into the computing enclosure wherever it turns out to be appropriate. Figure 25 below shows the model we will be purchasing.

Figure 25: Sparkfun mini modular breadboard

This system will also allow for general expandability without needing a revised motherboard design. For example, if additional 5V analog input pins are required, an additional ADC circuit can be built on one of the breadboards and wired into the communication ports of the computing platform.

5.3 Mounting

In order to integrate all of the electrical and computing components onto the existing chassis, mechanical mounting components will be designed to effectively use the footprint provided by the Daimscale team. A majority of the mechanical mounting will consist of 3D printed components. We will be purchasing a spool of TPU (thermoplastic polyurethane, a flexible printing material) filament to be used on components that require damping, and a spool of PETG (polyethylene terephthalate glycol, a common 3D printing material) filament for any other components not critical for damping.

Although having the complete truck body shell is not a requirement, we will take careful consideration in designing the mounting components in order to preserve as much of the shell as possible. We will not be modifying the truck body shell until after testing the components for optimal placement/angles and finalizing designs.

5.3.1 Raspberry Pi + Teensy Mounting

The Raspberry Pi will sit on the main computing enclosure on the scaled down vehicle. The computing enclosure is still being designed to account for the Raspberry Pi, Teensy, Motherboard and the mini breadboards. We will account for the limits of the platform due to the truck body shell and will stay within it. Sitting between the computing enclosure and the Raspberry Pi will be a Damping Mount shown in Figure 26 below. We will use this mount to reduce any vibrational effects on the electronics.

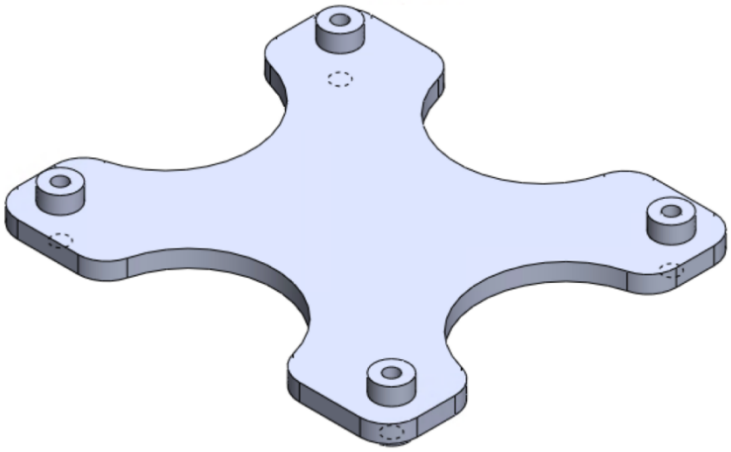


Figure 26: Raspberry Pi Damping Mount CAD Rendering

5.3.2 Lidar Mounting

The 1-D Lidar mount is a redesign of the Daimscale team's front bumper mount shown in Figure 27 below. This redesign ensures that the current front bumper can be used in combination with the 1-D Lidar.

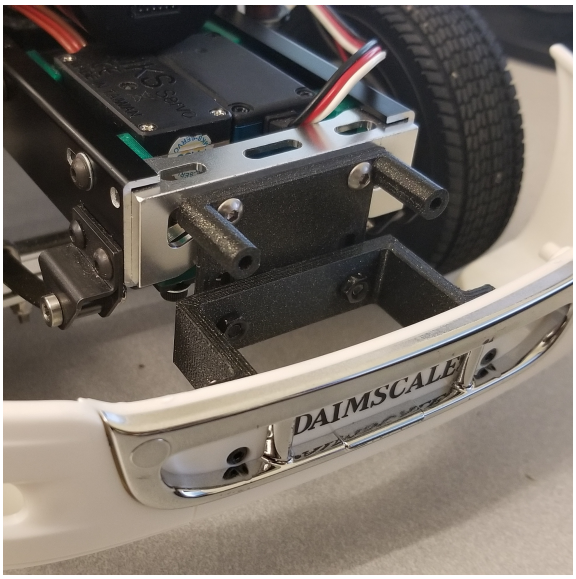


Figure 27: Daimscale's current front bumper design

The mount is meant to sit at the front end of the truck to allow the 1-D Lidar a full forward facing field of view. We will fasten the 1-D Lidar on to the 1-D Lidar Mount with M3 cap screws, as with the rest of the assembly.

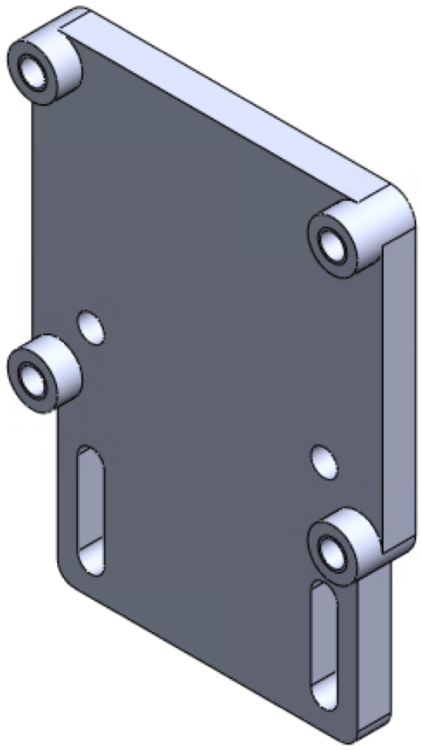


Figure 28: 1-D Lidar Mount CAD Rendering

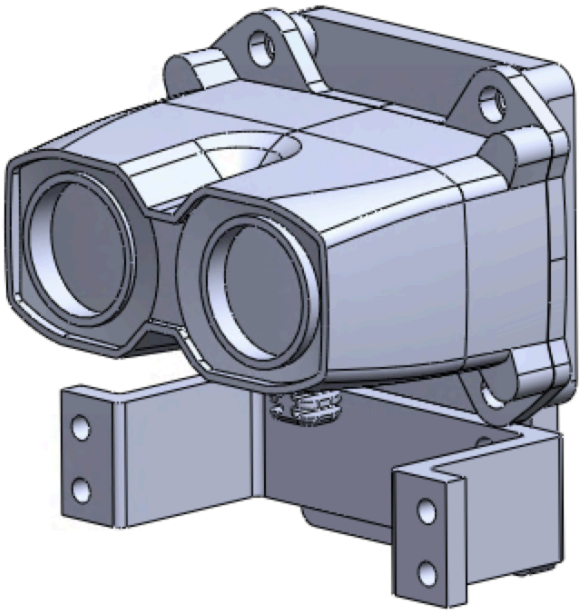


Figure 29: Assembly of 1-D Lidar and front bumper mount on the 1-D Lidar Mount CAD Rendering

We determined that the ideal position of the 360˚ spinning RPLidar is on the hood of the truck body shell. Our current design has the RPlidar sitting over the 1-D Lidar on the front end of the truck as shown in Section 6.1.2. We will be removing the hood from the truck body shell due to the size of the RPLidar,

limited space of the truck shell hood, and to maximize visibility of the environment for the sensor. The main platform is shown in Figure 30 below.

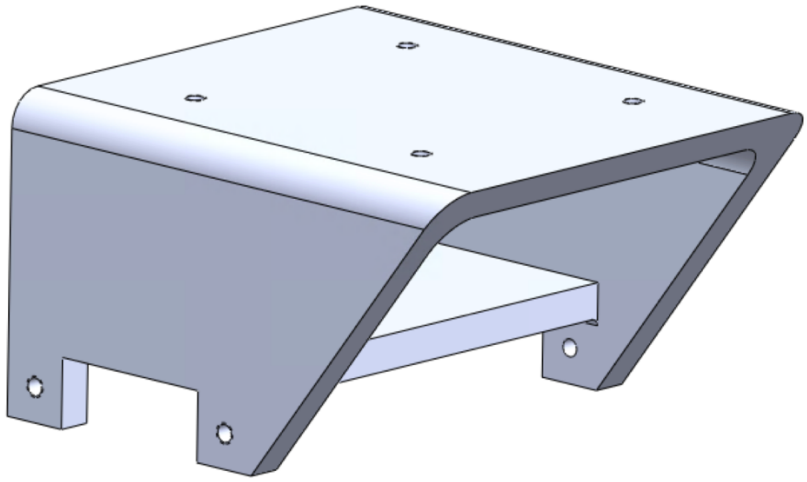


Figure 30: RPLidar Mount CAD Rendering

5.3.3 Ultrasonic Mounting

Since we will use the Ultrasonic Range Finder (URFs) to illuminate any areas the RPLidar does not detect, we will need to have a modular design of the URF mount for ease of testing as we finalize our design. Our design of the URF mount is therefore able to shift longitudinally along the side of the truck and is designed with a pivot that can be locked into place with a setscrew. This design will allow us to freely test the data we receive from the URFs from various angles and positions, until we settle on the best angle and position of the mounts. The URF mount will consist of 3 main parts: the URF Bracket, the URF Mount front, and the URF mount back, seen in Figures 31 and 32 below.

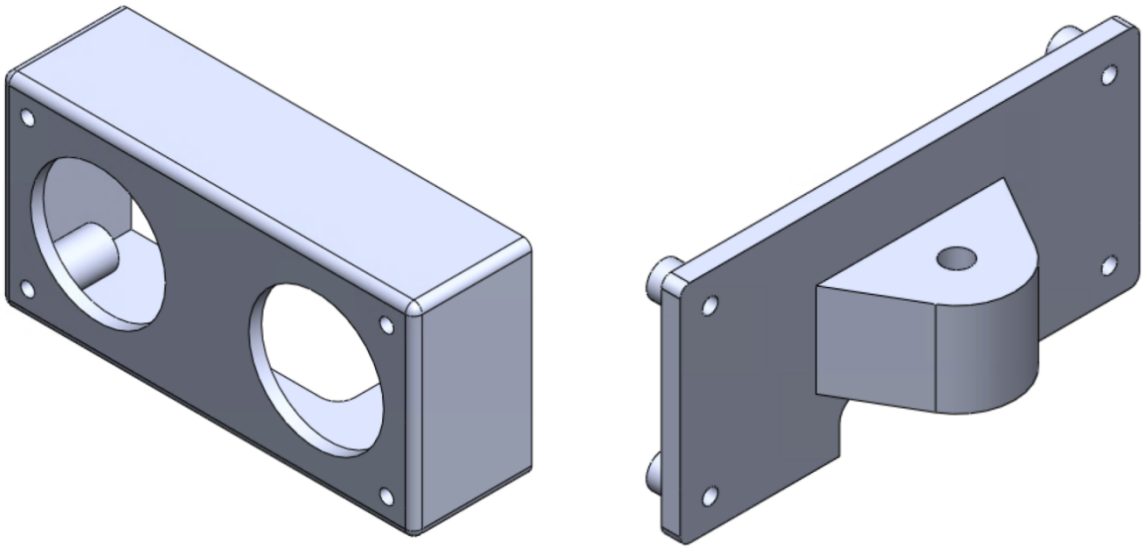


Figure 31: URF Mount Front (left) and URF Mount Back (right) CAD Renderings

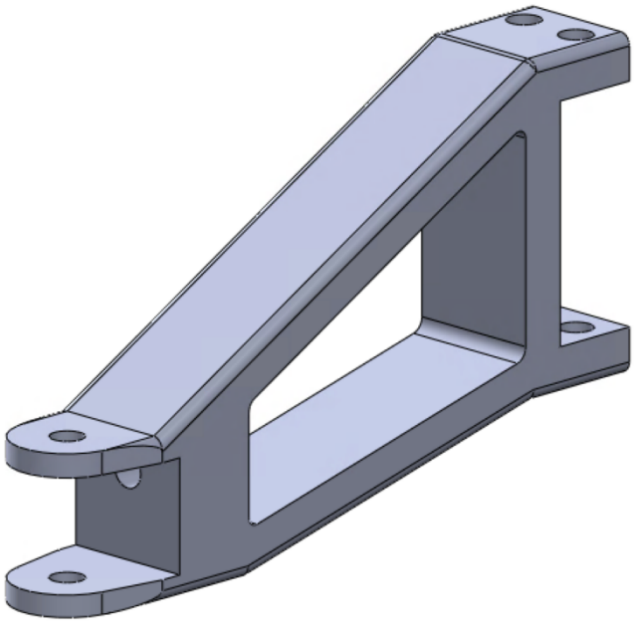


Figure 32: URF Bracket CAD Rendering

5.4 Software

The software design for our project is broken into two sections. The first section highlights what code will be run on the Raspberry Pi. We have made initial steps in writing the code, to a point where a group of ROS executables can run and communicate with each other. No actual semi-truck autonomous functionality has been completed at this time. The second section regards the development of the real-time code that runs on the Teensy. Initial concepts have been supplied, but none of the code has been written. We will likely first get the Teensy up and powered with the motherboard board before we start uploading code to the device and see if it is behaving as we expect.

5.4.1 ROS Design for the Raspberry Pi

As laid out earlier in the report, the middleware to run on the Raspberry Pi is ROS. This architecture consists of many "nodes" (otherwise known as executables) that can run independently of one another, and topics that relay the system information between the different nodes. Our initial design in ROS includes four nodes: rplidar_node, lite_lidar_node, pi_comm_node, and controller_node. The first two nodes listed have been found online through existing repositories, while the latter two will need to be completed by us. Authors robopeak [50] and cocasema [51] host the GitHub repositories with ROS-based, C++ source code for the RPLIDAR and Lite Lidar, respectively. Any of these nodes can be launched without the other nodes running, but this does not necessarily mean that they will function as intended. To play its part in the system, a given node will need to communicate to other nodes via ROS topics. A node that outputs data can "publish" its data to a topic. A node that needs to read data can "subscribe" to the topic that it wants data from.

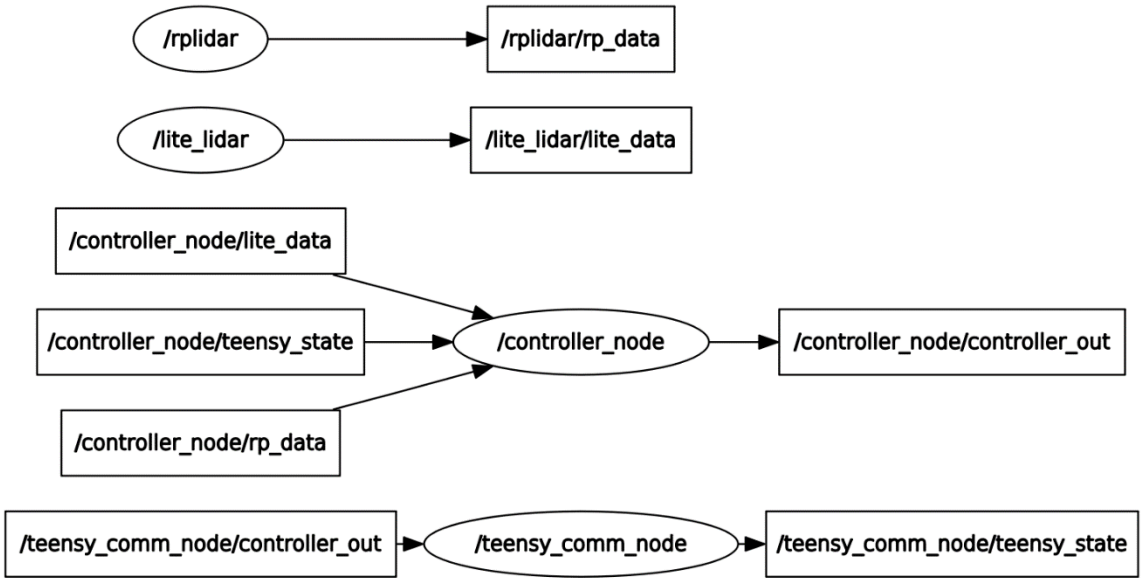


Figure 33: Initial ROS design with nodes and topics

Figure 33 shows the general layout of the system. Nodes are shown as circles and topics are shown as boxes. The above graph was generated through a ROS utility package called rqt_graph. Rqt_graph is run in parallel with a system of ROS nodes, and generates a diagram showing the relationship and dependencies within the system.

5.4.1.1 rplidar

The rplidar node is shown at the top of the diagram. This node has a simple purpose of reading and processing data from the spinning LIDAR and publishing the output to a the rp_data. Again, the implementation of this node is credited to robopeak and the author's GitHub repository.

5.4.1.2 lite_lidar

The lite_lidar node behaves similarly to the rplidar. It processes data from the forward facing, 1-D LIDAR and publishes its data to the lite_data topic. Author cocasema has written most of the code for this node.

5.4.1.3 controller_node

The controller node is the third node down in Figure 33. It subscribes to the three topics shown of lite_data, teensy_state and rp_data. Using a combination of this data, the controller node will carry out control systems calculations, and publish any results to the controller_our topic. This code will be written by us in the months to come.

5.4.1.4 pi_comm_node

Another node written by us is the pi_comm_node. This node subscribes to the controller_out topic to read and relay information across a serial port to the Teensy. In addition, the pi_comm_node reads data from the Teensy through the serial port. This data is published to the ROS system as teensy_data, and is needed for control loop functionality.

```
/* Controller must publish the controller output */
ros::Publisher ctrl_output = n.advertise<std_msgs::String>("controller_out", 1000);

/* Controller must subscribe to the following  */
ros::Subscriber lite_sub = n.subscribe("lite_data", 1000, communication_cb);
ros::Subscriber rp_sub = n.subscribe("rp_data", 1000, communication_cb);
ros::Subscriber teensy_state = n.subscribe("teensy_state", 1000, communication_cb);
```

Figure 34: Sample code for publishing and subscribing to ROS Topics

A snippet of C++ code that is written for the controller node is shown above in Figure 34. The first line depicts the syntax used for publishing data to a topic. A ROS Publisher object is created called ctrl_output, that is used to send messages (controller output data) to the topic. As written, the message is a String, with 1000 bytes of buffer space to avoid losing data that is not read immediately by a subscriber. Later in our development, we will need to change the message type to be some data that can be processed and sent to the actuators of the system, rather than a simple String placeholder. The actual topic that the message is posted to is denoted by the first parameter: "controller_out." The latter three lines show the syntax for subscribing to a topic. In this case the controller node needs to know information about the state of the Teensy and what its peripheral sensors are reading, along with the data from both the Lite LIDAR and the RPLIDAR. Instantiating these Subscriber objects is similar to how the Publisher object was created. The primary difference is that a third parameter is needed as a callback function defining how to actually process the incoming messages.

5.4.1.5 ROS and Simulink

One specification of ours is to interface between ROS and Simulink. Figure 33 does not show any node corresponding to communicating with a Simulink model, simply because we have not gotten to that stage in the development. We are still planning to cover this feature in the months to come. Fortunately, there is clear documentation regarding this aspect of our system [52]. Quick research reports that there is a Robotics System Toolbox feature within Simulink to create Publishers and Subscribers that can interact with Topics within the ROS system. The article mentions that Simulink can be run on the same device as the ROS system for easy communication. One concern we have with this is that running Simulink on the Raspberry Pi will take up significant amounts of processing power that may inhibit the performance of the ROS nodes controlling the semi-truck. It is possible that we will have to run the Simulink model remotely and communicate via Wi-Fi to the Raspberry Pi. A ROS node would then pick up the signal and relay the controller information to the rest of the system.

5.4.2 Teensy ChibiOS Architecture

The Teensy will be running a third-party real-time operating system called ChibiOS. We had initially planned to run the MIT open-sourced software known as FreeRTOS. The main benefit of FreeRTOS was that members of the team have experience with this technology from previous mechatronics courses. Unfortunately, the Teensy's microprocessor is a Freescale MK66FX1M-0VMD18 which does not support any version of FreeRTOS. As such, we have decided to go with the alternate plan for running ChibiOS. ChibiOS is licensed under GNU as a free software for running high performance embedded applications. The operating system can support a high volume of threads that are able to be interrupted, or preempted, by each other depending on different priority levels and time constraints [53]. The preemptive multitasking nature of this software is greatly important to the success and performance of our project.

We will be hosting several tasks that read sensors, control actuators and compute values that need to run "simultaneously." Preemption of threads running tasks is necessary for the smooth operation of the vehicle. Figure 35 shows the task diagram for the Teensy. It will be implemented in ChibiOS C/C++ code. Tasks with a low priority number can interrupt other tasks with a higher priority number in order to satisfy their timing requirements.

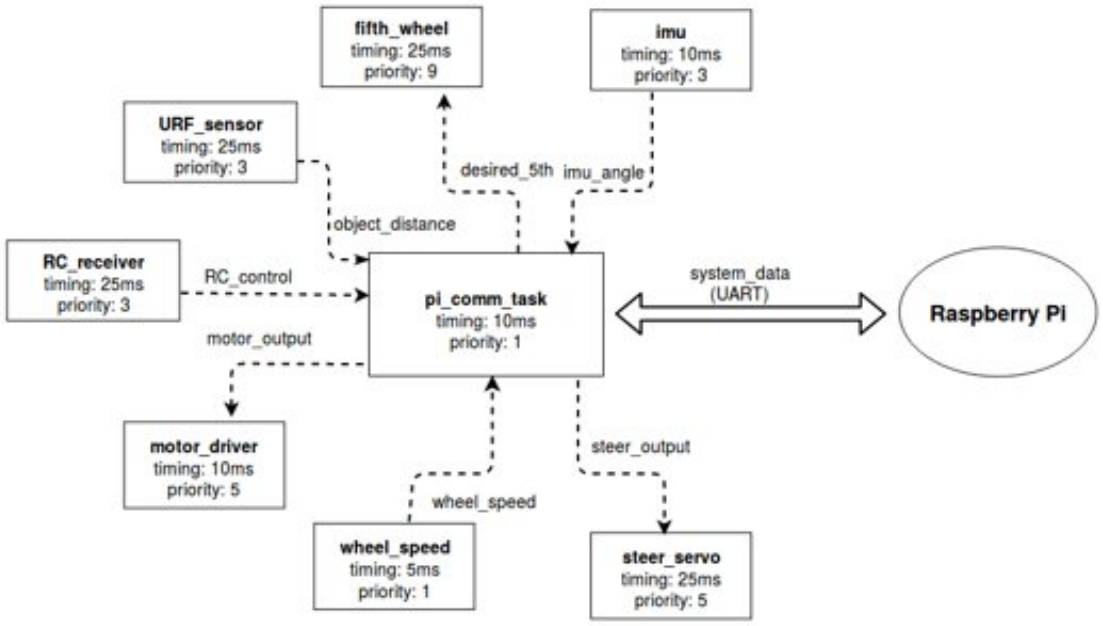


Figure 35: Teensy Task Diagram

5.4.2.1 system_data

Although this is not a task in itself, it is an important aspect of the design. The system_data will hold a variety of data that other tasks will have access to. A variable of this type can be initialized and passed as a reference to each of the other tasks in the system. Figure 36 shows our initial thoughts of what it might include. This is subject to change as we progress through implementing the design.

```
struct system_data_t {
    int16_t motor_output;     // output sent to the motor driver
    int16_t speed_setpoint;   // desired speed as a setpoint to the controller
    int16_t steer_output;     // output sent to the steering servo
    int16_t wheel_speed;      // speed that the wheel speed sensor is recording
    uint16_t imu_angle;       // euler angle read by the BNO055 IMU (degrees)
    uint16_t URF_distance;    // distance to an object seen by a URF
    uint8_t drive_mode;       // signifies either manual mode or an autonomous mode
    bool    desired_5th;      // desired state of the 5th wheel (locked or unlocked)
    bool    actual_5th;       // actual state of the 5th wheel
};
```

Figure 36: A system_data variable containing information for other tasks to use.

5.4.2.2 teensy_serial

The teensy_serial provides information transmission between the Raspberry Pi and the Teensy about the state of the vehicle. This task is designed to have a single state where it continually reads data from the system_data shown in Figure 36 and sends it over the serial port (UART communication) connected to the Raspberry Pi. In the same state, the task reads incoming data (if any) from the Raspberry Pi's end of the serial port and writes that data to the system_data variable. This is what allows for each device to updated with the most current information of the system. This task has a high priority of 1 because it is basically a bottleneck between the different devices. It needs to be able to interrupt other tasks to run. If this task were to run very slowly and have a low priority, then there would be significant lag between the computing devices and the system would not behave optimally. Consequently, the timing for this task is relatively quick at 10ms. The only task to have as high of a priority as this task is the wheel speed sensor, where missing data results in catastrophic changes in control loop behavior.

5.4.2.3 motor_driver

The motor_driver task is what sends signals to the physical motor driver mounted on the chassis. This device, which regulates the voltage supplied to the motor, outputs a PWM based on what was calculated from the control loop in the Raspberry Pi. Without this task, the primary functionality of the semi-truck would be rendered useless. A reference to the system_data is passed to this task, and it will specifically read value of the motor_output variable in Figure 36 to actuate the motor.

5.4.2.4 steer_servo

The steer_servo task is similar to the motor_driver task in that it continually reads steer_output from the system_data variable writes this as a PWM level to the servo. Since there are multiple servos within the semi-truck system, and servos are relatively common in mechatronics projects like this, we will likely be able to find existing code online to help control these tasks.

5.4.2.5 fifth_wheel

The fifth_wheel is another task that controls a servo. This task is responsible for locking and unlocking the "fifth wheel," which acts as the coupling between the tractor and trailer. It will have two states, locked and unlocked, corresponding to different PWM outputs that determine angle of rotation of the servo. This task has low priority and timing requirements due to the non-critical nature of toggling the fifth wheel.

5.4.2.6 wheel_speed

The wheel speed sensor is used for tracking the speed of the vehicle. The analog device used is SparkFun's Line Sensor Breakout – QRE1113. It is an IR reflectance sensor that can detect changes in reflective surfaces, such as the alternating black and white tape on the inner rim of the wheels. This sensor needs to have the tightest priority and timing restrictions due to potentially missing data as the wheels rotate. If it does miss a section of the tape, then the sensor will report that the speed of the truck is going drastically slower than it is, which will be relayed to the control loop and cause an unwanted spike in output motor level. We have determined the priority of this task to be 1 and the initial timing to be 5ms, although we can optimize this this after testing.

5.4.2.7 imu

This task controls Adafruit's BNO055 9-axis orientation device used for tracking orientation of the semitruck. Fortunately, Adafruit has available drivers for the BNO055 sensor on GitHub [54] that we can incorporate into our task. Euler angle data regarding the semi-truck orientation is read over the I2C connection. Ultimately this angle data ends up on the Raspberry Pi's control loop after being sent over serial communication in the teensy_serial. The timing for this task is set at 10ms, which is the sample rate as specified in the datasheet [55]. The priority is 3, leaving the only the wheel_speed and pi_comm tasks able to interrupt the IMU.

5.4.2.8 URF_sensor

This task controls the URF sensors that are oriented on the sides of the vehicle. If an object is detected by the sensor, a value signifying its distance will be read from the sensor and written to the system_data. There is an associated library for the device given by ElecFreaks [36]. This will supply us with some code to help read data from the sensor based on which pins it is hooked up to on the Teensy. The task has a specified timing of 25ms and a priority of 3. It is possible that we will need to adjust this to accurately read the signals that are transmitted and received by the URF in relation to nearby objects.

5.4.2.9 RC_receiver

The RC_receiver task reads in different PWMs from the RC receiver that is already mounted on the semi-truck for manual control. There are two auxiliary channels not used for steering or acceleration that are wired up to the Teensy. The user can select different modes on the RC controller to send different PWM signals to the receiver. This task includes watching the dead-man switch on the RC controller as well as being responsible for triggering the switch between manual mode and a variety of different autonomous modes that may be saved on the Raspberry Pi. Like the URF sensor, this has a timing of 25ms and a priority of 3. We may need to alter these values to ensure functionality. In particular, some form of interrupt or polling may be required to accurately measure the output of the receiver if a static frequency is not sufficient.

5.5 Cost Breakdown

The total budget for our project is $3,000. Using this as a baseline we separate our project costs into five categories: sensors, motherboard, mechanical, electrical, and computing. Table 5 below shows our roughly budgeted and actual expenditures for the project as a whole.

Table 5: Project budget

Category	Budgeted	Spent
Sensors	$ 700.00	$ 617.72
Motherboard	$ 400.00	$ 363.73
Mechanical	$ 200.00	$ 123.86
Electrical	$ 150.00	$ 156.61
Computing	$ 100.00	$ 68.98
Total:	$ 1,550.00	$ 1,330.90

Sensor spending reflects the purchase price of all of our sensors, including those bought as part of testing and evaluation. Motherboard spending includes the requisite set of components plus extras and PCB orders for both versions of the motherboard. Mechanical costs include 3D printing filament and fasteners from McMaster. Electrical spending includes two LiPo batteries and a multitude of wires and connectors. Computing expenditures include one Teensy 3.6 and one Raspberry Pi 3B+, both bought to replace the units handed down to us.

Based off our bill of materials in Appendix I, Table 6 below shows how much we would spend in each category if we were to make another vehicle, repeating the current design.

Table 6: Vehicle budget

Category	Spending
Sensors	$ 529.70
Motherboard	$ 142.68
Mechanical	$ 88.34
Electrical	$ 123.60
Computing	$ 64.25
Total:	**$ 948.57**

The total comes out to $950 with the current version of the vehicle, assuming the only costs associated with printed parts is the bulk filament orders. As expected, the bulk of the spending is on sensors. Within our total project budget of $3,000 another group could fit the Daimtronics-related components of a second vehicle into the current pool, where the total of our current expenses and of one more vehicle comes out to about $2,300.

5.6 Updated Design since CDR

When implementing the design that was laid out during CDR, we made several changes to the different subsystems in the platform. Some sensors were re-specified, the motherboard was redesigned for improvement, additional mounting was created, and software was redesigned. The details for each subsystem are described in detail.

5.6.1 Updated Sensors

The first main change that we made regarding the sensors was to replace the URFs with TOF (time of flight) sensors. Once the URF's were hooked up to the Teensy and the code to control the devices was written, we found that the accuracy of these devices was not up to the standards that we wanted. The main issue was that when an object that was not directly square to the line-of-action for the URF signal, the signal would bounce off in another direction and the sensor would not receive feedback that an object was in the vicinity. We researched alternative sensors to solve this problem and decided on the Adafruit VL530X TOF sensors. These communicate via I2C, which needed to be considered in the design of the motherboard. An updated sensor diagram is shown in Figure 37 below.

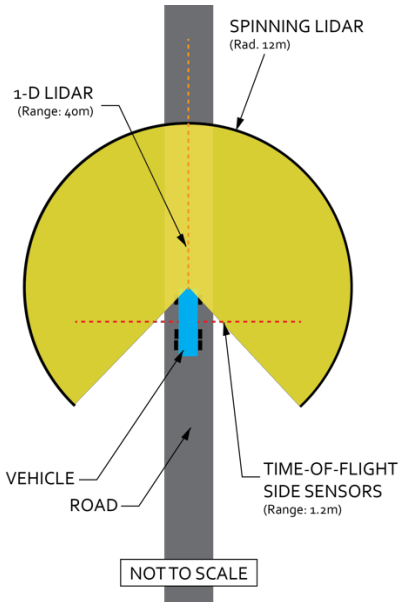


Figure 37: Final Sensor Angles and Ranges [59]

The second change pertains to the wheel speed sensors. We came across the realization that we could not determine the direction of motion of the wheels, only the magnitude of how fast the wheels were moving. Our initial plan was to change the digital version of the QRE113 back to an analog version and incorporate varying shades of reflectivity on the tape mounted inside the wheels. Using a strip of tape that had three different reflective values (as opposed to just two) would allow us to determine the direction of motion. Fortunately, during the manufacturing process we discovered that there is a direct output from the motor that can read the speed and direction of the motor in the form of a Hall Effect sensor. We decided that this would be the simplest way to read the speed of the vehicle. With this decision, we had to account for the backlash through the transmission that would offset the actual motion of the vehicle to what is being read from the motor output.

5.6.2 Updated Motherboard

The motherboard underwent an additional revision since Section 5.2, though the overall design is the same. This subsection will describe the changes from the previous design.

5.6.2.1 Adjustable 12V Supply Rail

There is an additional power supply rail supplied by a smaller 1A switching voltage regulator than the primary 5V regulators that is set to 12V but is adjustable using feedback resistors on the board. This is intended to supply peripherals that are not 3.3V or 5V, such as a more powerful 12V fan if future users require it. This rail pulls directly from the motor battery.

5.6.2.2 Electromechanical Relay

The two solid-state SPDT relays have been replaced with a single electromechanical relay that has two sets of SPDT contacts. It is a latching relay by necessity, since non-latching relays do not come at a low enough power consumption level to be supplied by the Pi or Teensy. However, this was discovered after ordering the revised board, so the negative lead of the relay goes to ground instead of to another IO pin.

The current motherboard has manual adjustments replacing that trace with a wire hooked up to the Pi GPIO. The next version of the motherboard will need to correct this (this is further expounded in Section 9.1.1).

5.6.2.3 Ports and I2C

The pin ports were rearranged to reduce trace length and complexity, and an additional I2C port was added using an I2C multiplexer and accelerator to multiply the number of available channels. The multiplexer replaces I2C channel 3 on the Teensy. It reads commands from the main line of channel 3 to switch between its eight downstream channels, allowing sequential communication between eight different I2C devices and removing the device address restriction for its eight. Each of the multiplexer I2C channels have 10K resistor pullups and blank power supply rails. These need to be connected to the same voltage of the device being read, but allows for any signal voltage, while the multiplexer shifts everything back to the 3.3V for the Teensy. The accelerator injects extra current to the active I2C lines whenever it detects voltage changes to minimize the effects of high or varying bus capacitance on the maximum data throughput of the entire channel. Despite all of this, the I2C multiplexer and accelerator circuitry is currently inoperable for unknown reasons, and we are using only the four original I2C channels.

5.2.6.4 Current Sensing 5th Wheel

There is an INA250 integrated current sensing chip attached to the power supply traces going to the 5th wheel servo. This provides a 3.3V analog output directly to one of the Teensy analog pins that is scaled at 2V/A, allowing for the detection of excess current draw by the 5th wheel servo in the event of a lockup as it tries to transition between positions.

5.2.6.5 Motor Data Output

The vehicle's motor has a data output port outputting the motor shaft's position and speed using the ticks of a Hall Effect sensor inside the housing. The output cable connects directly to the board using a JST ZHR-6 connector. The board also has 5K resistor pullups. The Eagle device file for the connector has two pins switched, so the cable has minor modifications to remedy this. Fixing the device pin definitions is listed in Section 9.1.1.

5.2.6.6 Regulators for 3.3V Lines

Finally, the Zener diodes used to provide 3.3V supply to the board have been replaced with linear voltage regulators. The diodes only worked for extremely low current and could only step voltage down to 4.2V under any kind of load. The linear regulators are not as efficient as the primary switching regulators, but are simpler and use less board space. The number of supply/ground pin pairs for each supply port was reduced to six from eight to save board space, making room for the additional 12V line, and due to not being necessary.

5.6.3 Updated Mounting

Changes since CDR for mounting include the computing housing, second battery housing, a revised RPLidar mount, and revising the URF mounts to fit the ToF sensors. All parts are 3D printed in PETG (polyethylene terephthalate glycol-modified) or TPU (thermoplastic polyurethane). PETG is for rigid parts and TPU is for flexible parts, being a rubbery material.

5.6.3.1 Computing Housing

The motherboard, Teensy, and Pi are enclosed in a box over the ESC, held in place by columns fastened to the chassis. All rigidly connected components use M2.5 hex nuts press fit into one part and the corresponding screw through the other.

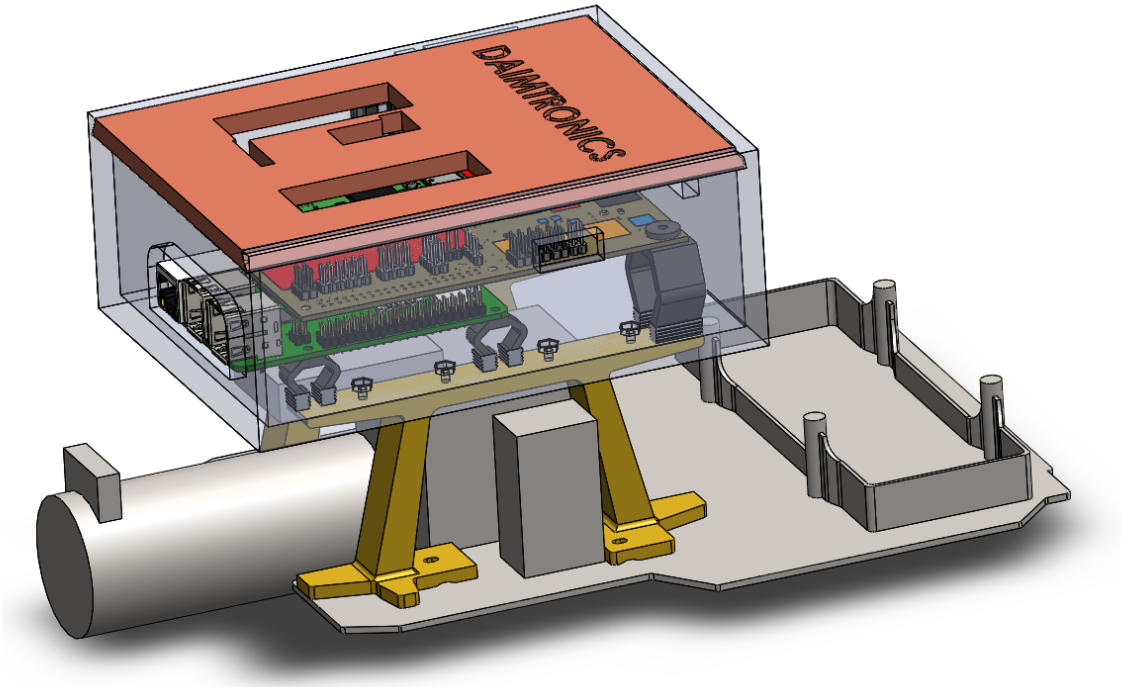


Figure 38: Computing housing

The lid provides access to the pin ports on the Teensy and has a soft snap-fit into the casing to secure it. The motherboard assembly is suspended using 3D printed TPU springs.

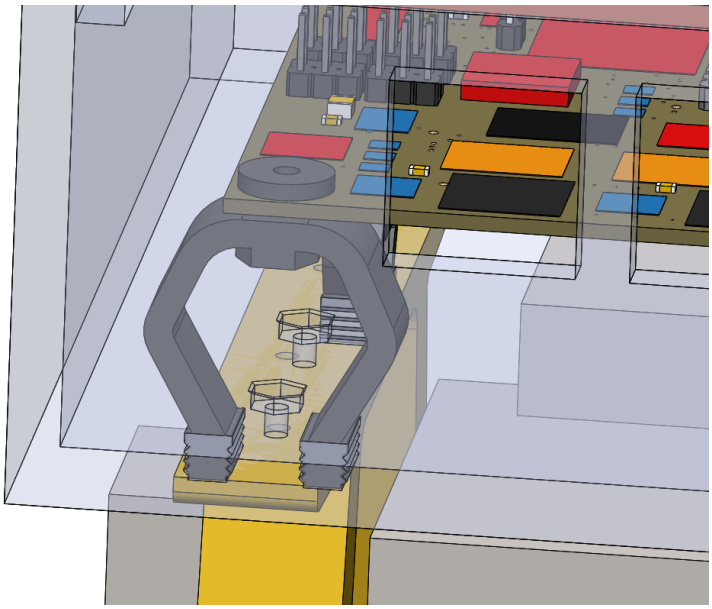


Figure 39: Motherboard springs

The springs have ridges that press-fit into matching ridges in the casing, which makes it very easy to assemble but has a high removal force due to having to compress each ridge and having to do so multiple times to remove the spring.

The case features a 5V centrifugal fan fastened to the side of the housing with a printed TPU duct press fit into the case.

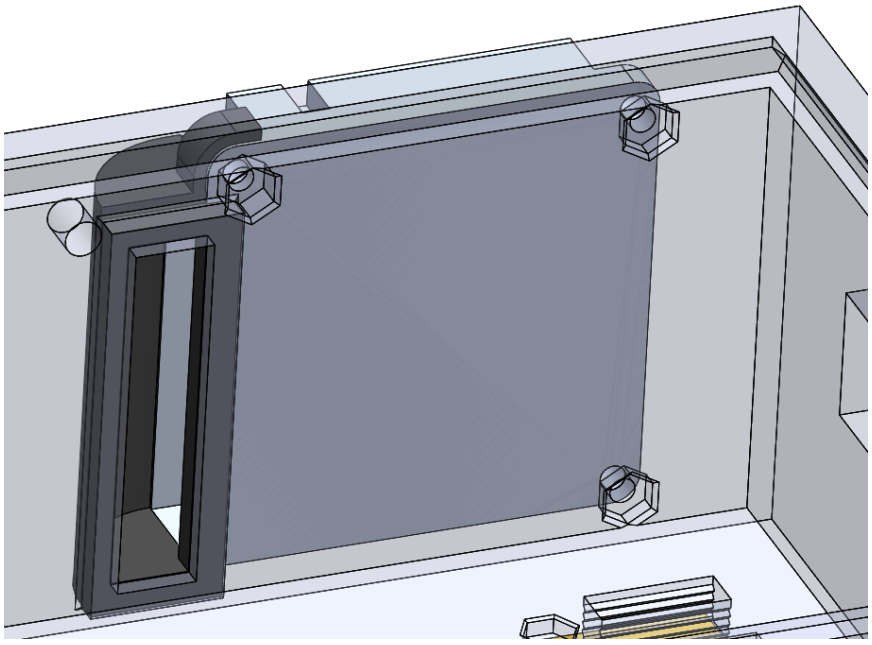


Figure 40: Case fan and duct

A centrifugal fan was chosen due to its high-pressure output versus axial fans, which is necessary to keep air moving quickly through the enclosure, which has relatively little volume. The fan is supplied off the motor-side 5V rail.

The main casing itself is three parts that slide together – the floor, the walls, and the gate. This was done so that the motherboard and spring assembly can be mounting onto the floor without the walls getting in the way.

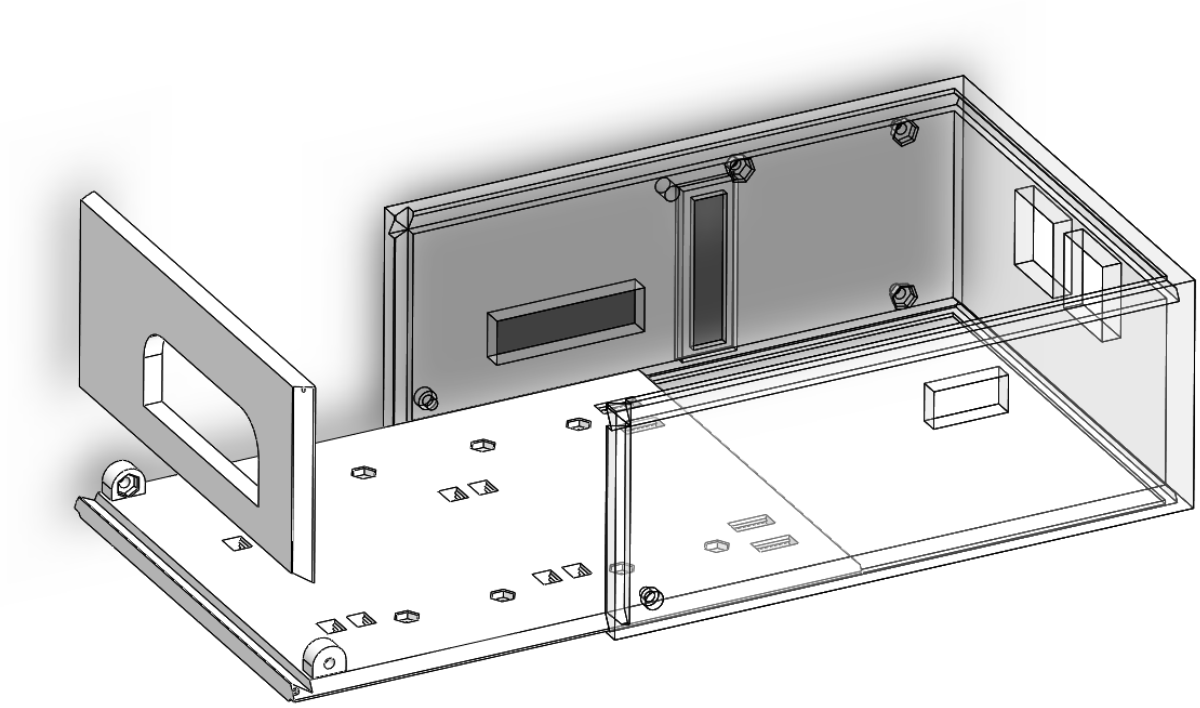


Figure 41: Computing casing components

After the floor is fully assembled, the walls slide on and are screwed to the floor to improve rigidity. The gate slides in, and then the lid after that.

5.6.3.2 Second Battery Housing

The second battery sits over the original one in the chassis, suspended by several columns fastened to the main plate of the chassis.

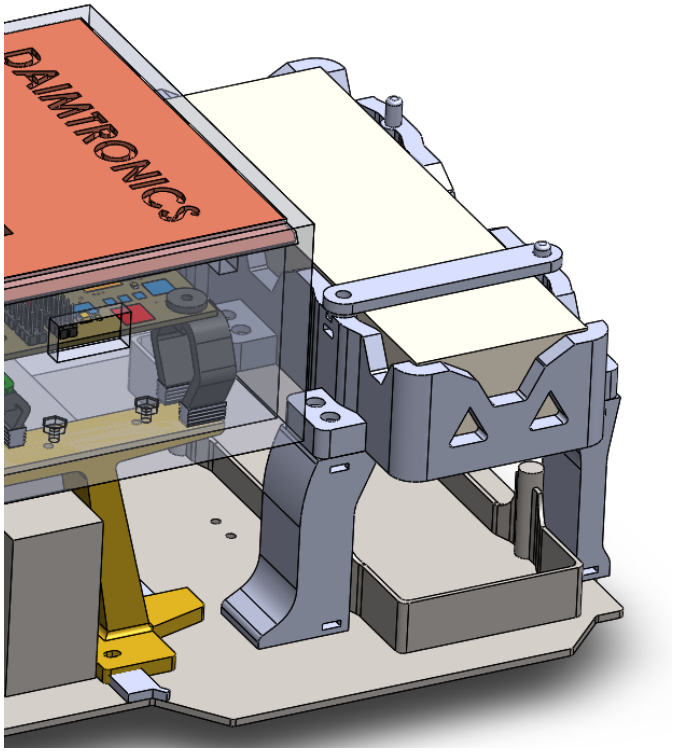


Figure 42: Second battery housing

The battery is held in with flexible rotating fingers that are functionally identical to the ones Daimscale made. Our version can be used to replace the Daimscale parts if they break.

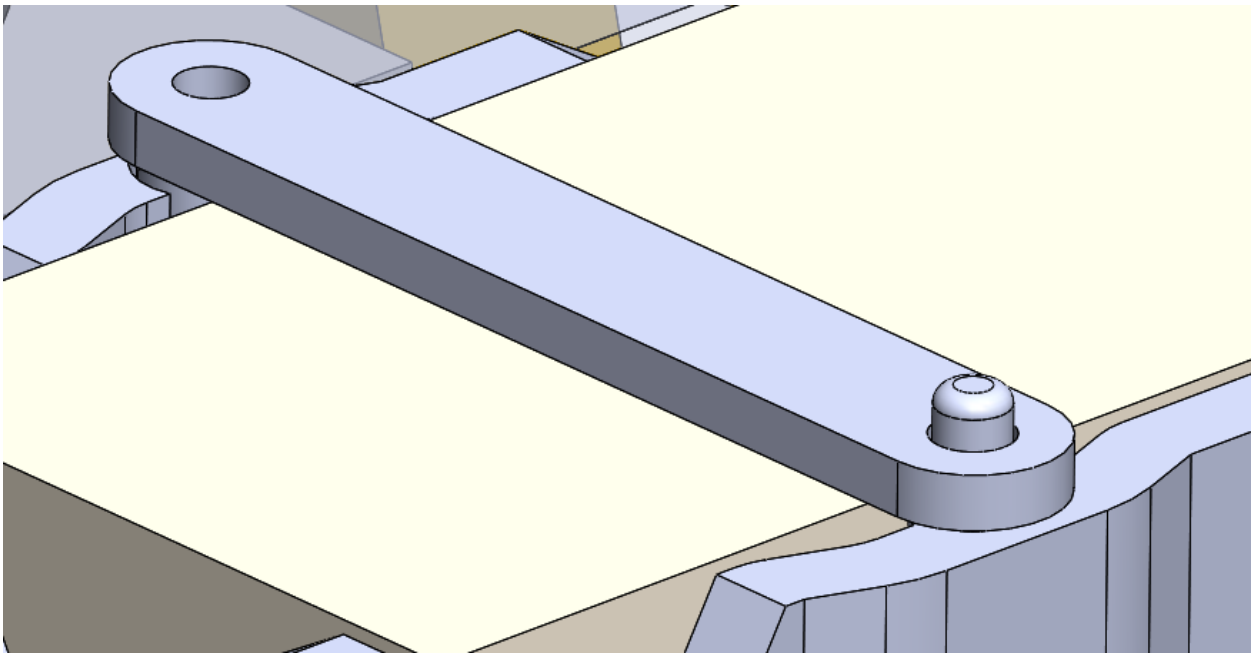


Figure 43: Battery fingers

The columns supporting the back of the housing need to be adjusted, as they currently run into the battery fingers of the original housing. This is mentioned in Section 9.1.2.

5.6.3.3 RPLidar Mounting

The updated RPLidar mount replaced the rear through-holes for screws with snap joints with levers due to there not being matching screw holes on the chassis.

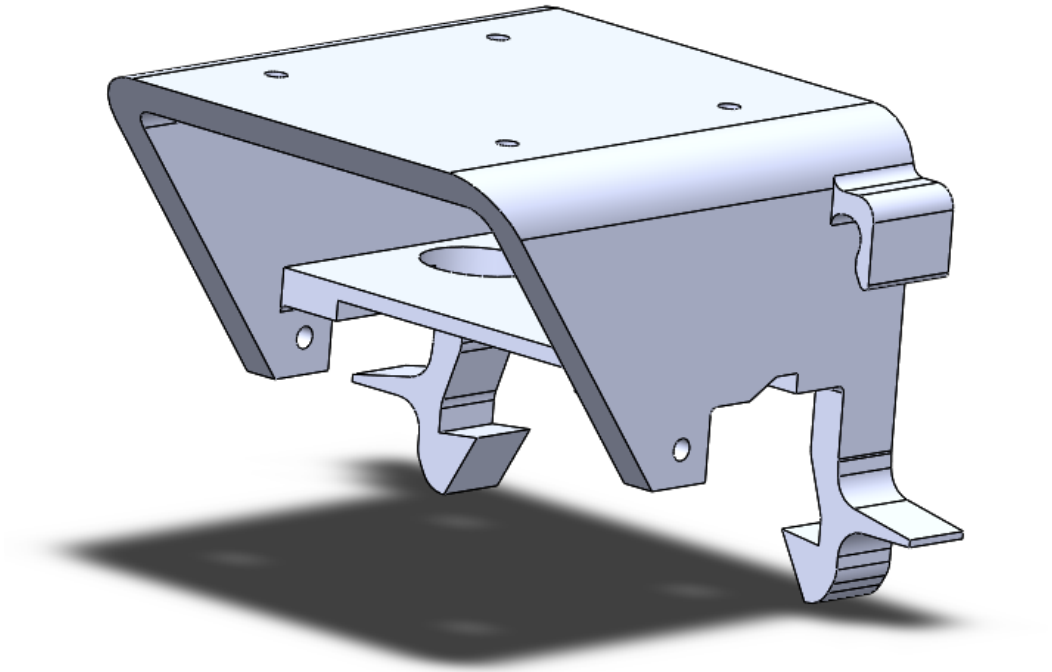


Figure 44: Updated RPLidar mount

A channel was added to the bottom of the mount to make room for the steering servo's cable. There are also hooks for the RPLidar's cable to hold it in place and extra holes to make it easier to reach all four mounting holes on the bottom of the lidar.

5.6.3.4 Time-of-Flight Mounting

The design of the side sensor mounts that were original for the Ultrasonic Rangefinders is fundamentally unchanged, but the head of the mounts have been changed to fit the much-smaller Time-of-Flight sensors instead.

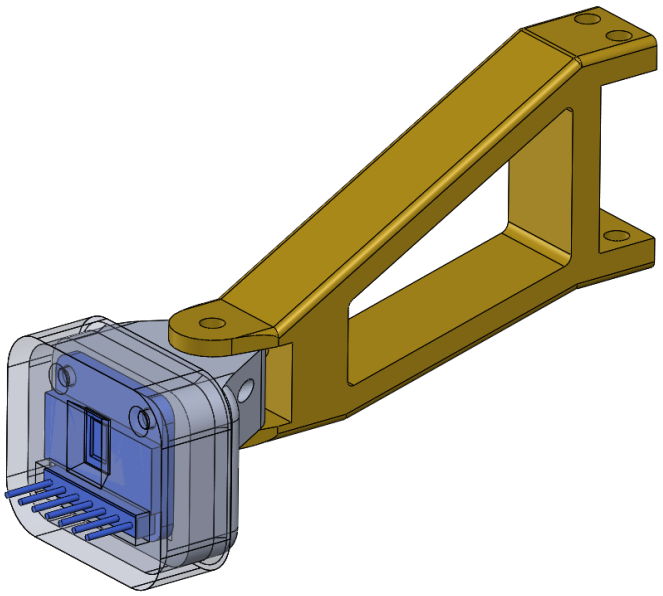


Figure 45: ToF sensor mount

The head can swivel when adjusting the angle of the sensors and then tightened down with the bolt at its swivel joint.

5.6.3.5 Servo 5th Wheel Mounting

The 5th wheel servo is attached to the rear of the chassis, several inches before the 5th wheel link. There are two clasps that fit onto the servo that are tightened down using M3 screws and preexisting tapped holes on the chassis frame.

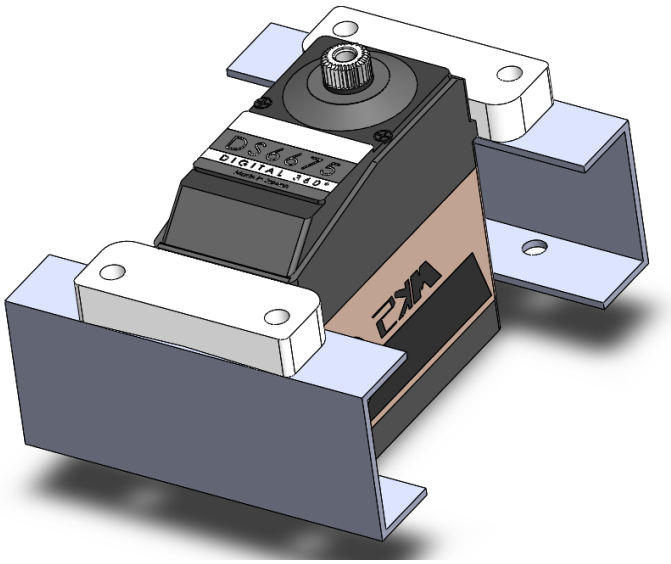


Figure 46: 5th wheel mount

The linkage connecting the servo to the 5th wheel lever has not yet been designed. A wire to pull on it should be a quick way to get it up and running and may be a near-optimal solution regardless.

5.6.4 Updated Software

Through conducting models in Simulink External Mode between a laptop and the Raspberry Pi, we discovered that the ROS network that was originally expected to be exclusively run on the Pi actually runs both on the Pi and the laptop. When Simulink is involved in External Mode, the entire control loop model is run on the PC and communicates through ROS's built in TCP/IP communication protocol to relay information to the Pi. It effectively becomes its own ROS node that is running on the PC, but still is connected to the ROS network on the Raspberry Pi. Figure 47 shows more clearly the relationship between the three devices in the system. It should be noted that a user can still "deploy" a Simulink model to the Raspberry Pi, and keep everything running on the system attached to the semi-truck. In this case, the Simulink model is converted to ROS node, and is run directly on the Pi's hardware. In this mode, however, here is no capability for viewing sensor and actuator data through Simulink scopes.

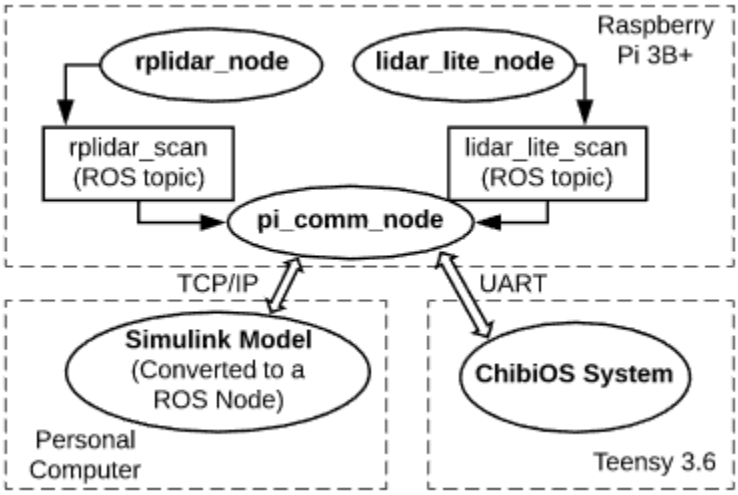


Figure 47: Updated Software System Model for Simulink, ROS and ChibiOS.

An example Simulink model has been included in Figure 48. This model shows three different subsystems. The left-most is a ROS subscriber subsystem that subscribes to all of the ROS topics running on the Pi, including the RPLIDAR's topic, Lidar-Lite's topic, and Pi serial communication topic. The middle subsystem is the control loop system that we expect future users to work most closely with. This is where all of the control system theory can be carried out and tested. The right-most subsystem takes care of publishing to ROS topics that hold all of the actuator data that eventually needs to be sent over to the Teensy for control over the motors and servos. Finally, the Simulation Rate Control block in the upper left corner controls how often the loop runs and sends messages to the actuator data topic on the Raspberry Pi.

Users can view the system data as they traditionally would using Simulink. There are scopes and simout blocks that can be used to view the data in real time, and save the data for later analysis. Users can also alter block values within the control loop subsystem to see how the semi-truck responds to changes in gain values.

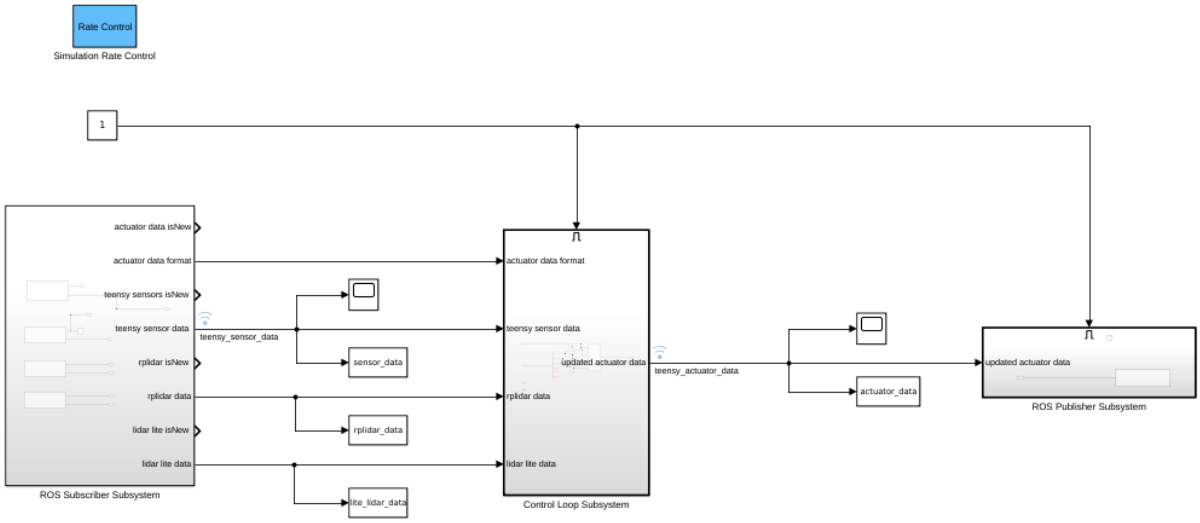


Figure 48: Example Simulink Model.

For people using our system without any desire to work with Simulink, they must work more directly with the tools that ROS provides. ROSbags are the standard approach to saving simulation data throughout a session. We have created launch files to automatically output all of the sensor and actuator data that is within a ROS topic to a ROSbag. These are stored within the “semi_catkin_ws/rosbags” directory, and a script that we wrote can be used to export the ROSbag data to csv’s for each of the topic running during the Simulation. More specific information on how to achieve this is explained in our User Manual.

Figure 49 show the updated Task Diagram for the tasks running on the Teensy. The general structure is the same as before (Figure 35), but tasks designed for the original set of sensors has been updated to display new tasks. The system_data that is shown in the diagram is not its own task, but since this holds the central state information of the semi-truck, we have included it in the diagram to show how the tasks relate to the system data. We have used a “shared-memory” approach to this statically declared variable. Any task that attempts to write data to the system_data must first acquire a mutex and lock the resource so that there are no race conditions within the code.

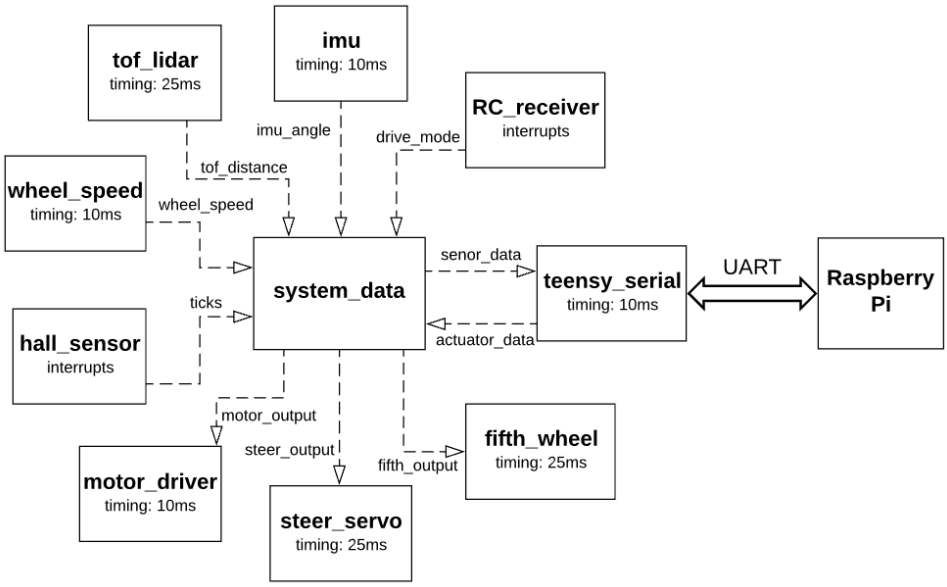


Figure 49: Updated Task Diagram for the Teensy.

More specific discussion on the implementation of individual tasks can be found in section 6.3.

The overall format of the main.ino file has also been updated for easier understanding. The top of the file contains a description of the overall organization to the Teensy code and the include statements for the header files containing the implementation for each specific sensor or actuator. Underneath that is several #define statements for each pin and its associated sensor or actuator. As we worked on all of the sensors and actuators on the Teensy, we found that having the pins declared in the specific .cpp file associated with the sensor or actuator was cumbersome for debugging and did not make for easily readable code. Instead, by having all of the pins declared at the top of main and passing them into each .cpp file as parameters, we were able to change pins on the fly in one location and avoid having to keep track of pins across multiple .cpp files.

6. Manufacturing Plan

The manufacturing for our project is split into three categories: mechanical, electrical, and software. Mechanical mounting encompasses mounting solutions, electrical encompasses wiring and motherboard assembly, and software encompasses the development of the drivers, middleware, and application layer for the computing platform.

6.1 Mechanical Manufacturing

We will be 3-D printing all of our Mechanical Mounting components. All of the sensor mounts are to be printed with PETG filament. This includes: the 1-D Lidar Mount, the RPLidar Mount, the URF Mount Front, the URF Mount Back, and the URF Bracket. PETG filament is significantly strong for the purposes of these components. The damping specific components will be printed with TPU filament. For now, the only component being printed in TPU is the Raspberry Pi Damping Mount.

6.1.1 Ultrasonic Assembly

The Ultrasonic Range Finder will be placed inside the URF Mount Front and URF Mount back and will be fastened together with M3 cap screws. This subassembly will then be placed into the URF Bracket piece as shown in Figure 50. An M5 screw will be placed in the swivel joint of this assembly to allow the URF Mount to rotate on the URF bracket. An M3 setscrew can be inserted from behind to lock the mount to a specific angle if needed.

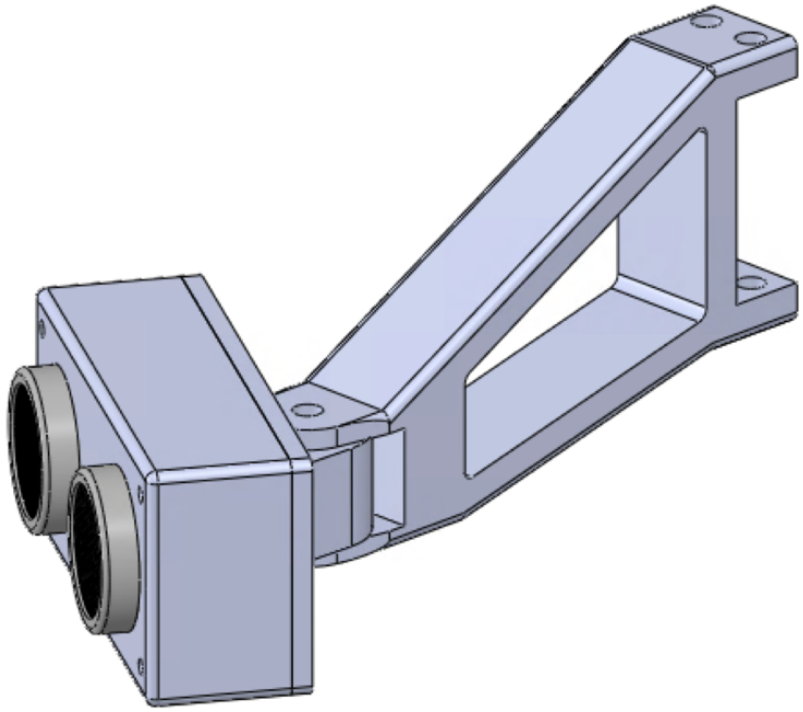


Figure 50: Full URF Side Mount assembly

6.1.2 Lidar Assemblies

The new 1-D Lidar Mount will be screwed in place of the previous front bumper mount with the same two M3 cap screws. When the mount is secured to the front of the truck, the 1-D Lidar can then be secured on the mount with 4 M3 hex cap screws.

For easier assembly, we recommend attaching the RPLidar to the RPLidar Mount before attaching it to the truck. Four M3x8 hex cap screws will secure the RPlidar to the mount from below. An M3 hex Allen key will be used to secure the screws. The entire subassembly will then be placed over the 1-D Lidar and secured with two M3 cap screws to the truck on either side of the mount. The full lidar assembly would look like the CAD Rendering in Figure 51 below.

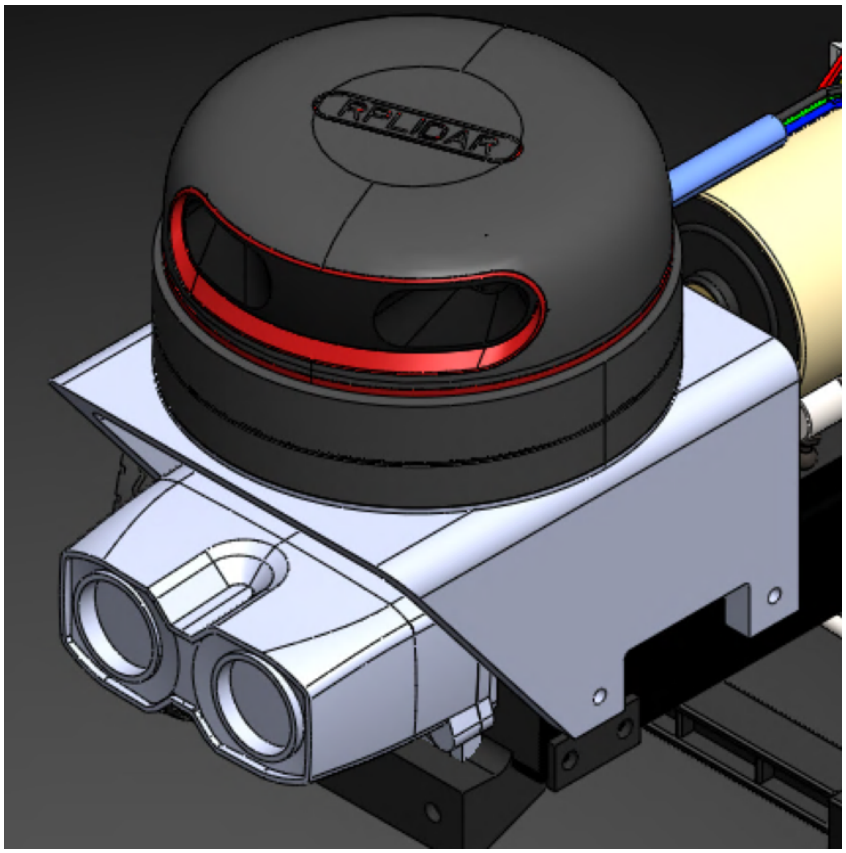


Figure 51: CAD Rendering of the RPLidar and 1-D Lidar Mount assemblies

The front bumper of the truck shell then attaches to the bottom of the 1-D Lidar Mount with two M3 hex cap screws to secure it for added protection and aesthetics. The bumper of the shell does not need any modifications.

6.1.3 Full Sensor Assembly

With the front assembly of both lidar sensors secured, the array of Ultrasonic Range Finders that supplement the lidar sensors will be attached on the sides of the truck. The sides of the truck currently have multiple tapped holes that can be used to secure the URF brackets to the truck with M3 hex cap screws as shown in Figure 52 below of the Full Sensor Assembly. The full RPLidar assembly is not shown in this figure as the STEP file provided by Slamtec contains a build error with the lens covering of the sensor and therefore it is hidden in this view.

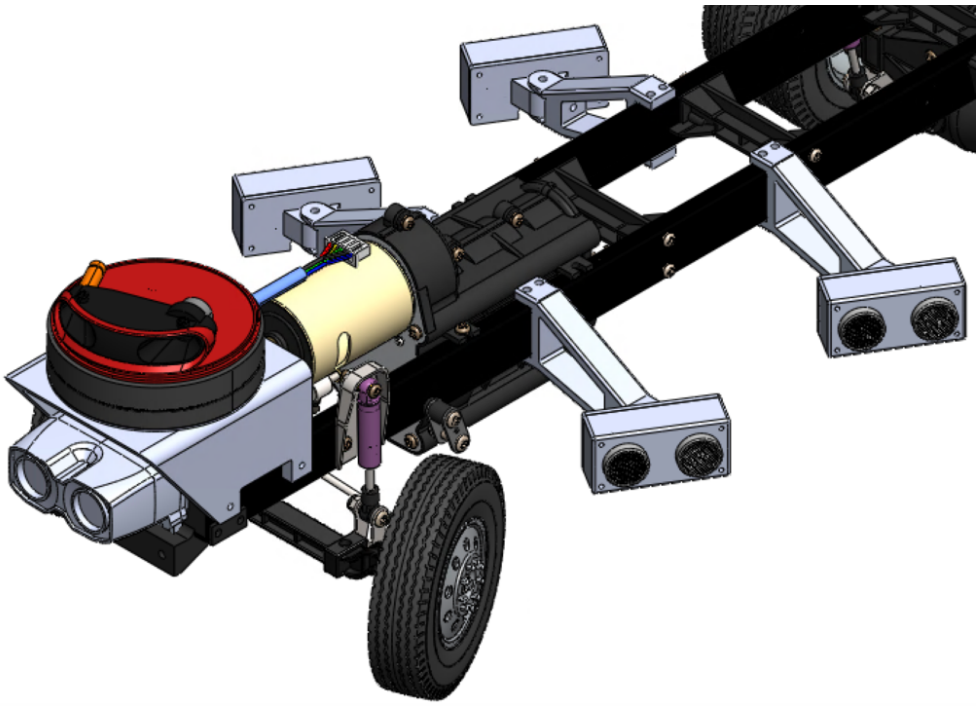


Figure 52: Full Sensor Assembly CAD Rendering

We currently have a plan to implement four Ultrasonic Range Finders to supplement the side views that the RPLidar cannot detect. Therefore, this assembly will be modular and allow us to secure the URFs along different points on the truck to find the best location for each sensor as testing continues.

6.2 Electrical Manufacturing

The bulk of electrical manufacturing is assembling the motherboard, with the remainder being modifications to wires.

6.2.3 Motherboard Assembly

The motherboard comes as a flat, blank circuit board with through-holes and pads for soldering components. All of the components come packaged in tape or tube form. To ensure that the motherboard is assembled correctly, the layout of the board is compared against both the digital board design and the schematic to cross-reference component location, connections, and values. A printout of the component list that summarizes the name, value, and footprint of each component will also be utilized to ensure maximum accuracy.

Proper soldering safety measures include always returning the soldering iron to its holder, being mindful of heated objects, and having a fan operating to remove fumes except for when the breeze may cause especially small componentry to blow away. Having a clear workspace with slack on the soldering iron ensures that nothing will accidentally be knocked off the workbench. Additional measures when using leaded solder include not touching the eyes or mouth after coming into contact and thoroughly washing hands after the work is done.

Using a soldering iron requires the tip of the iron to be cleaned of excess solder then wetted with a small length of molten solder to maximize heat conduction to the board and to the component. When soldering, the solder is ideally melted by heat coming from the part being soldered and not directly from the iron, ensuring that the solder actually adheres to the component.

Since there is a low overall number of components we are not investing in a stencil, but we *are* using solder paste to aid in the components with large heatsinking pads, such as the primary voltage regulators. Primarily we will be using solder rolls and a soldering iron to assemble the board, with a heat gun to remove components and for activating the solder paste. Attaching the surface-mount components first is easiest since it allows the board to lay flat. Surface-mount components should be handled with fine tweezers. Soldering down one end of the component at a time is recommended. A multimeter can be used to detect shorts and verify connections and component values as they are added.

All of the through-hole components (terminals and pin sockets in our case) can easily be soldered at the end of the process. Gripping the board and the pins being soldered with a soft vise grip or clips is recommended so that the board and the part being soldered do not have to be held by the operator while pins are soldered.

If a component needs to be replaced, the heat gun should be used to remove it, taking care to not damage the connections of adjacent components. If components have to be moved to a new blank PCB, again use the heat gun for easy removal of the original components. Large rows of headers and pins are difficult to remove from old boards, so typically we recommend buying new headers if making a new board.

6.2.1 Wiring Modifications

The only wiring modifications required are to the board side of the battery connectors. For the computing side, the ends of the purchased connector need to be stripped so that the wires will fit into the motherboard terminal. For the motor-side, the cables leading into the ESC need to be spliced so that power can be routed to both the ESC and the motherboard.

To splice the motor-side LiPo wires, we will cut and strip both wires of the current connector, then use a three-way splice connector to go from the battery to the ESC and the motherboard.

The actual crimp action can be performed with heavy-duty pliers or with a dedicated hand tool if available. After crimping, additional soldering can be performed to ensure maximum connection between the input and output as well as adding additional joint strength.

6.3 Software Manufacturing

A large portion of the project consisted of writing code to control all of the hardware and electronics that we have specified. Here we describe our approach to developing the software, and the differences between the Pi and the Teensy. We have started a GitHub repository for both the ROS code and Teensy code at https://github.com/nfurbeyr/daimtronics.

6.3.1 Developing ROS code for the Raspberry Pi

The Raspberry Pi has been loaded with a microSD card with a Linux Raspbian operating system. Raspbian is a light version based on the Debian OS that is designed to run specifically on the Raspberry Pi, which doesn't need the same amount of background processes as a typical computer. This allows for more computing power to go into applications that the user directly controls. Also installed on the microSD card is the Kinetic Kame distribution of ROS, with steps to carry out this process found on the ROS Wiki page [56]. This is what will govern control over all of the nodes and topics that are used in the semi-truck system.

The Raspberry Pi has an HDMI port for a monitor and four USB ports for servicing a mouse and keyboard for user interactivity. This gives the ability to run and debug code directly on the Raspberry Pi. The standard Bash terminal is used to run commands necessary for launching the ROS nodes. If a bug is found in the code for our project, it is easy to edit the files, recompile the source code, and relaunch the nodes for quick feedback. Built in Wi-Fi on the Raspberry Pi allows the device to access the internet and download additional code from repositories found on GitHub.

Tutorials for ROS have been consulted for getting the system running to where the multiple nodes shown in Figure 33 are communicating with each other. There were a few key steps, such as installing and configuring the ROS environment and building a ROS package that were required before ROS actually behaved as intended. There are plenty of online tutorials that cover this [56], so we won't go into much depth here.

6.3.2 Developing ChibiOS code for the Teensy

Programming the Teensy with ChibiOS code was less user friendly than programming the Raspberry Pi. A major inconvenience with programming the Teensy microcontroller is that there is no direct user interactivity with the device. Our code must be uploaded from a computer to the Teensy through the micro-USB port that is built into the chip. In order to expedite this process, we will used the Teensy Loader software (running on a PC) and the Halfkay Bootloader (which comes pre-flashed onto the Teensy) for quick debugging and uploading cycles. PJRC has a few tutorials regarding details of the process [57].

We used the Arduino IDE (Integrated Development Environment) for writing the code that runs on the Teensy. The Arduino IDE by itself does not support code for running on the Teensy architecture, but fortunately the Teensyduino add-on fixes this problem. Teensyduino simply adds another option in the IDE to select the board and libraries for the Teensy hardware.

ChibiOS came into play by creating a ChibiOS thread whose job is to run a task. For example, the imu task shown in Figure 35 was mapped to a thread that runs the task function of reading data from the IMU. Threads are given a few different parameters as shown in Figure 36 [58]. In the case of the imu, the initial priority was set to 3, the thread function points to the function with code to actually read from the IMU, and the thread parameter would be the system_data that holds the variable imu_angle shown in Figure 36. The stack size will need to be adjusted during development, and kept to the smallest size without getting any stack overflow problems when processing the task.

```
Thread *tp = chThdCreateFromHeap(NULL,            /* NULL = Default heap. */
                                 THD_WA_SIZE(128),/* Stack size.          */
                                 NORMALPRIO,      /* Initial priority.    */
                                 myThread,        /* Thread function.     */
                                 NULL);           /* Thread parameter.    */
```

Figure 53: Initializing a ChibiOS thread

When our system is running the list of tasks, a thread is designated to run each task. ChibiOS's kernel contains a scheduler service that gives each thread a small amount of time to run, depending on the thread's timing and priority requirements. ChibiOS's documentation recommended to not mess with the kernel scheduler, but instead use the API for ChibiOS synchronization [57]. Each task accesses the system_data structure that resides in shared memory within the main.ino file. We used ChibiOS's semaphore library to lock the system_data any time a task tried to write to it. If this had not been done, a task could have been interrupted in the middle of a write, and the data would be corrupted.

6.3.3 Software Manufacturing Updates for Final Design

Software manufacturing for the project went relatively smoothly. We were able to use existing code online for several of the device drivers, including the TOF, IMU, and both LIDAR sensors. Since most of the results of the coding manufacturing can be viewed from within the source code repository, we will not go too in depth here, but we wanted to highlight some of the major challenges that we faced.

6.3.3.1 Serial Communication

The serial communication between the raspberry pi and the teensy required the most in-depth coding on our parts. There were issues with how the bytes were transferred from one device to the other, including structure alignment, or byte packing. We found that even if we intended to send only one byte across the UART lines, in certain cases the C/C++ code for Arduino would automatically send an extra byte of junk data. This has to do with keeping the same structure as adjacent data that has just been sent before or after the sending of the single byte. Furthermore, the synchronicity problem for sending data caused us some issues for a while. When either the Raspberry Pi or the Teensy is executing its serial communication code, and the other device starts up and runs its code for serial communication halfway through, the data can become mismatched. For example, the Raspberry Pi might read a value that the Teensy intended to be the IMU angle, but based on in what order the Pi processed the data, it could be interpreted as a different sensor value, such as the wheel speed. To overcome this issue, we implemented a system so that every time a full set of sensor values was going to be sent, a device would first send a "sync value" that has a predetermined and consistent value that the other computing platform expects. If the receiver did not get the correct value, it empties the buffer up until it got the next sync value.

6.3.3.2 Updated tasks for wheel speed

The wheel_speed task on the Teensy has been rearranged to use the built in Hall sensor output (shown in Figure 54 below) from the Tekin motor instead of the QRE113 digital light sensor. The wheel_speed task itself was rearranged to track immediate speed and direction, while another task was needed to keep track of the ticks on the sensor.

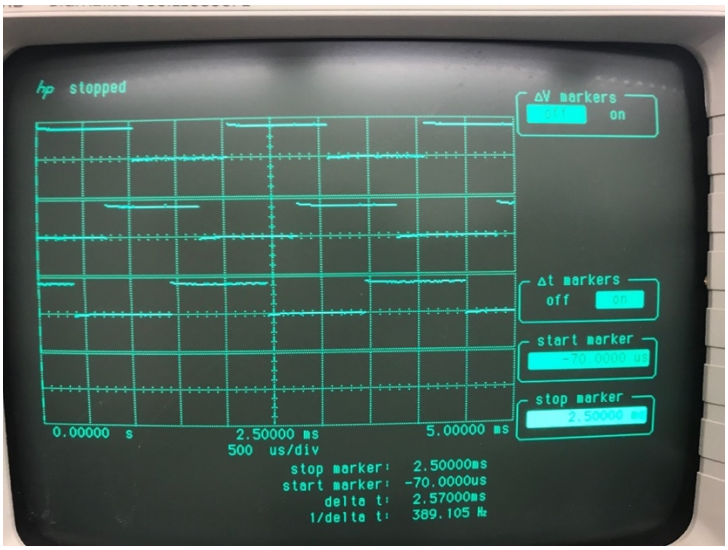


Figure 54: Hall Sensor output from the Tekin RX8 BLDC motor

The new task called the hall_sensor task was created to run on interrupts to count the number of ticks that have occurred during runtime. “Ticks” is a separate statically declared variable in the main file on the Teensy, that doesn’t reside within system_data. Wheel_speed runs on a standard timing basis within the ChibiOS scheduler, and updates the sensor_data variable based on the number of ticks that have passed since last run, and the time elapsed between the last run of the wheel_speed task. From this we are able to send the speed and direction of the truck to the ROS system on the Pi.

While we were quickly able to read the frequency of revolutions that the motor was outputting, we could not determine the direction consistently. We implemented the hall_sensor task using interrupts that would trigger every time the motor’s phase A pin had a leading edge. The electronics were setup so that phase B and phase C signals on the motor were also hooked up to pins on the Teensy. Based on the 120 degree offset between the phases, we could determine if the motor was going forwards or backwards based on whether phase B or phase C was high during an interrupt. In theory, our plan was solid, but actually writing the code for the sensor proved unfruitful. We were getting inconsistent, junk data for whether the pins were high or low during interrupts. After several days of attempting to debug this (both in software and hardware), we finally realized that the pins were initialized wrong in the setup section of the hall_sensor Teensy code. We had passed in to the pins’ setup functions that they should be in INPUT_PULLUP mode rather than just INPUT mode. It turns out that INPUT_PULLUP does not allow you to read a pin’s value like INPUT mode allows you to. Although the Hall sensor does require pull-up resistors to function, the new revision of the motherboard includes external pull-up resistors allowing us to switch from INPUT_PULLUP mode to INPUT mode.

6.3.3.3 Updated tasks for TOF

After scrapping the URF sensors that were not accurate enough for our purposes, we integrated the TOF sensors into the Teensy. All three of the sensors needed to run off I2C communication. These TOF’s in conjunction with the IMU would have used up all of the I2C ports. As is mentioned in the motherboard design, a I2C multiplexor was incorporated to free up some spaces in case additional devices running off

of I2C needed to be included. Unfortunately, we were not able to interface properly with the multiplexor. We were able to ping the device, and get the device address to display on the serial console hooked up to the Teensy, but trying to access anything downstream from the multiplexor was unsuccessful.

6.3.3.4 Updated RC Receiver task

The RC_receiver task on the Teensy has been split into two separate threads for two switches on the RC controller shown in Figure 55 below. Switch 1 is used as the dead man's switch that has to be pressed in order to output to the actuators during automatic mode. Switch 3 controls whether the platform is in manual mode (middle), automatic mode 1 (left), or automatic mode 2 (right).

Figure 55: Switch 1 and Switch 3 on the RC Controller

Each switch changes the PWM signal outputted from the RC receiver to the Teensy pins shown in Figure 56 below. Both threads function in the same way, with each thread monitoring a specific pin on the Teensy that correlates to each switch on the RC controller. This was accomplished using interrupts to monitor the pin changes and run the specific thread to read the PWM signal on that pin.

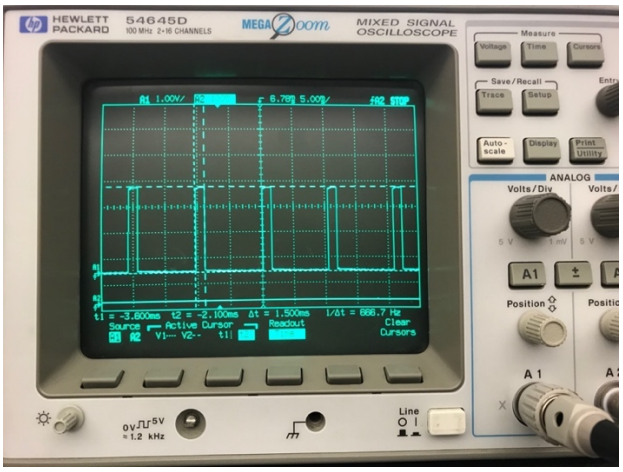


Figure 56: PWM signal sent out by the RC receiver for the Teensy to transcribe

An example of one of these threads is shown in Figure 57, below. The CH_IRQ_HANDLER declared on line 227 runs preemptive instructions that prepares the Chibios system to awaken the switch 1 thread by sending a message in chThdResumeI on line 232. When the thread is resumed, the timing of the PWM signals sent by the RC receiver to the Teensy is read and the appropriate mode is returned.

```
221  /**
222   * @brief RC Receiver Switch 1 Interrupt Handler: Runs preemptive Chibios Interrupt
223   * code and awakens the RC receiver Switch 1 thread.
224   */
225  static thread_reference_t rc_sw1_isr_trp = NULL;
226
227  CH_IRQ_HANDLER(RC_SW1_ISR_Fcn){
228      CH_IRQ_PROLOGUE();
229
230      /* Wakes up the thread.*/
231      chSysLockFromISR();
232      chThdResumeI(&rc_sw1_isr_trp, (msg_t)0x1337);  /* Resuming the thread */
233      chSysUnlockFromISR();
234
235      CH_IRQ_EPILOGUE();
236  }
237
238  /**
239   * @brief RC Receiver Switch 1 Thread: Reads auxiliary signals from the RC receiver switch 1
240   * for determining if the deadman switch is pressed.
241   *
242   * This thread calls RC_receiver_SW1_fn which is the primary function for
243   * the RC receiver switch 1 and whose implementation is found in RC_receiver.cpp
244   */
245  static THD_WORKING_AREA(rc_sw1_isr_wa_thd, 128);
246
247  static THD_FUNCTION(rc_sw1_handler, arg) {
248
249      int16_t deadman_mode;
250      while (true) {
251
252          chSysLock();
253          chThdSuspendS(&rc_sw1_isr_trp); // wait for resume thread message
254          chSysUnlock();
255
256          deadman_mode = RC_receiver_SW1_fn(RC_SW1_PIN);
257
258          chMtxLock(&sysMtx);
259          system_data.deadman = deadman_mode;
260          system_data.updated = true;
261          chMtxUnlock(&sysMtx);
262      }
263  }
```

Figure 57: Example Code for the RC receiver task for Switch 1, the deadman switch.

6.3.4 Documenting the Process

We have documented the entire software portion of this project using Doxygen. An index.html file has been generated and placed in daimtronics/doxygen/html, and can be opened in a browser to navigate through our written files and functions. In addition, a PDF has been generated and attached to this report as Appendix N. We suggest that future user focus primarily on the files and functions that reside within daimtronics/teensy_chibios/src/, and daimtronics/semi_catkin_ws/src/semi_truck/src/. These are the bulk of the code that we wrote, and will be key for understanding how the system works, and what can be done to add additional code to the platform.

7. Design Verification Plan

Verifying that our design is satisfactory requires both system-level and project-level verification to show we meet our overall project specifications, and intermediate tests that allow us to verify our design direction as we are able to. Appendix J gives a table for our DVPR with each of the tests that we conducted for verifying our design. All of our specifications that were determined in Table 2 have tests to verify whether or not the specification was met. In addition to these tests, there are some intermediary tests that helped us get to our target goals for the project.

7.1 Specification Tests

This section goes over how we intend on showing we meet each of our engineering specifications in Section 3.5. The number in parentheses in the section title is the corresponding item number in the DVPR, Appendix J.

7.1.1 Cost - (21)

To ensure we stay within budget we tracked our project expenditures by category and by transaction in our project ledger, delineating costs between the mechanical, electrical, motherboard, and computing aspects of the project. We have completed this project significantly under our total budget, spending about $1330, with a remaining budget of $1670 to be used for a second vehicle or further development. Table 5 in Section 5.5 details expenditures by category.

7.1.2 Control Loop Frequency - (22)

Because we were a little delayed in getting the entire system up and running with all of the sensors and actuators interfacing as we had hoped, we were not able to do extensive testing with the control loop frequency. We discovered fairly late in the game that when the Simulink External Mode is used the control loop doesn't actually run on the Raspberry Pi, but rather the PC running Simulink. Unsurprisingly, the rate at which the control loop is able to run heavily depends on how complex the calculations within the control loop are. This is something that the next group that takes over the platform will have to test and optimize.

7.1.3 Average Acceleration Time - (23)

To verify the acceleration time, we timed how long it took the vehicle to accelerate to top speed. Detecting full speed was a combination of expecting the time-to-completion to be similar to the time to brake, which is easier to detect and was the test we performed first; observing the speed of the vehicle; and listening for the pitch of the motor and wheels to match the tone heard at full speed and to be constant. Using

these three heuristics to estimate when top speed was achieved, we found the acceleration time to be slightly under 3 seconds, meeting the 4 second target of this specification.

7.1.4 Average Deceleration Time - (24)

We ran the vehicle at maximum throttle for several seconds to get up to full speed then braked with the RC controller, using the RC's built-in commands as a conservative estimate of the maximum braking capability of the motor. The vehicle consistently braked in 2.6-2.8 seconds, meeting the 4 second target of this specification.

7.1.5 Setup time - (25)

With the advent of the control loop running on a PC rather than directly on the Raspberry Pi, we have found that executing a model that has just been created is very fast and seamless. When the ROS nodes on the Pi are running, one can simple press "Run" within Simulink (having the correct model configuration parameters as mentioned in the User Manual) and have new model controlling the semi-truck in a matter of 3 to 4 seconds. For running control models using ROS inline code, the "catkin_make" command has to be used in the terminal of the Pi in the "semi_catkin_ws" directory. This will trigger a build of the new node, and takes 15-20 seconds.

7.1.6 Steering Curve and Max Angle - (26)

The max angle achieved for steering the semi-truck was determined by centering one of the front wheels on a protractor. We used the RC controller in manual mode to apply the maximum angle that the truck could achieve and measured the angle to be 20 degrees in both directions. Since we didn't change any of the hardware for steering capabilities from what the Daimscale team had produced, we consider this test to be successful. Our original target was less than 10% deviation from similitude relative to an actual truck, but semi-trucks can actuate their wheels far past what the Daimscale chassis is inherently capable of, so we are not considering the 10% value to be our actual target.

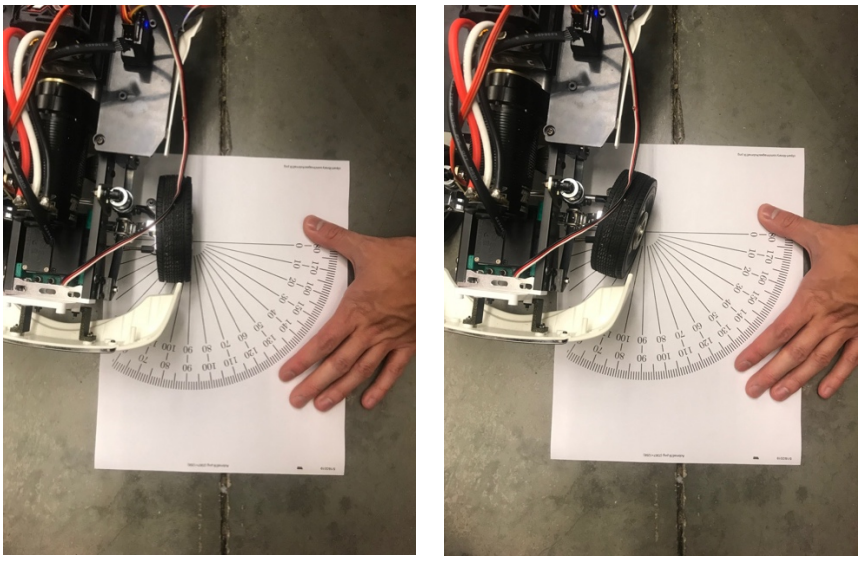


Figure 58: Steering Max Angle Testing in Manual Mode

7.1.7 Sensors Read in Simulink - (27)

To verify that we are able to use sensor values read by the Teensy within Simulink on the Raspberry Pi, we created a test controller that used output values from a time-of-flight sensors, inertial measurement unit, and ultrasonic rangefinder to modulate the output of the motor and steering servo, verifying that sensor values are directly usable in Simulink. Figure 59 shows the change in output for the steering servo over time in one of our simulations.

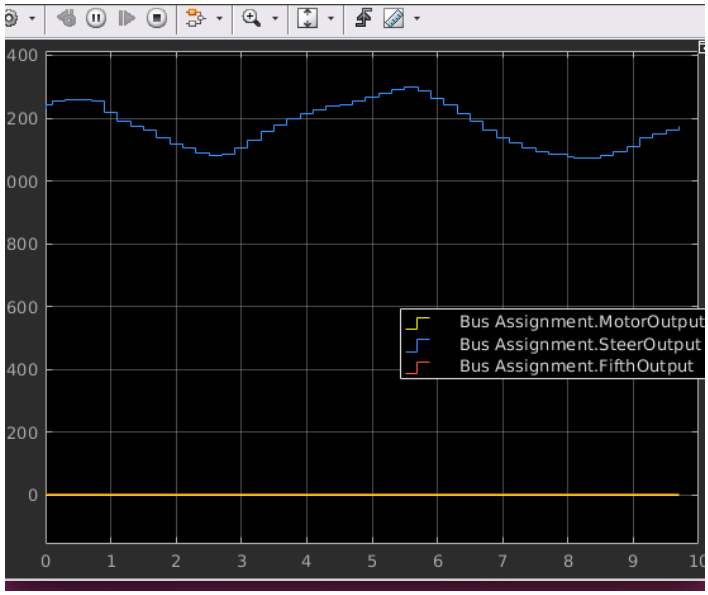


Figure 59: Simulink scope readings for actuator data

7.1.8 Top Speed - (28)

The test for the vehicle's top speed was performed by measuring the time it took the truck to traverse two markers spaced 20 feet apart with the truck having accelerated to top speed prior to passing the first mark. One person was stationed at the beginning to start the timer when the vehicle traversed the first marker, and another was stationed at the end to call out when the truck passed 20ft. This test comprises our statistical analysis.

The truck averaged 9.8mph, with a 95% T-based confidence interval spanning 9.05 – 10.54mph. The uncertainty in the speed is the combined propagated uncertainty of the total uncertainty of the distance and the total uncertainty of the time. The distance uncertainty was negligible compared to the time uncertainty, being almost 600 times smaller. The total time uncertainty was split between measurement uncertainty and of the statistical uncertainty. With a standard deviation of 28ms, the 95% T-distribution sample mean error is 34ms. The resolution of our timer was 10ms, but we considered the measurement uncertainty to be 100ms to account for the error in the operator's perception of when the truck crosses the first marker and the delay between the second operator calling out the end of the run.

With the time's measurement uncertainty being three times larger than the statistical uncertainty, the statistical uncertainty contributes only an additional 6ms to the total uncertainty of the average time, 106ms (the average time was 1.39sec). Due to this, essentially all of the uncertainty in the average velocity of the truck is due to the measurement uncertainty of the time.

Figure 60: Speed and acceleration test setup

Regardless, the highest value we can reasonably assume the top speed of the truck to be is only 10.5mph, beneath our original specification of 20mph. However, upon discussion with our sponsors, we decided that 10mph is a sufficient benchmark, since the 1:14 scaling equates this to 140mph, far above what we would want a truck to be travelling. The 20mph specification was derived using a 1:sqrt(14) scaling, a choice made by the Daimscale team that we have overruled. Further, the output of the motor is currently being artificially constrained by the RC controller with limits input by Daimscale. We did not remove these limits since 10mph is sufficient, but if faster speeds are required, they could be achieved. The output limits of the vehicle while in automatic mode used the values read off of the RC controller, so the automatic mode limits are currently the same constrained values as the manual RC controller limits.

7.1.9 Max Wi-Fi Range - (29)

Communication between the Raspberry Pi and laptop running Simulink was achieved through using the Personal Hotspot feature on an iPhone, though we assume it is generalizable to any mobile hotspot. The phone was moved 30m away from the other devices indoors and in multiple orientations; both the Raspberry Pi and laptop were able to maintain a connection to each other. This satisfies our minimum target of 10m.

7.1.10 Object Detection Range - (30)

Our primary intention for measuring the object detection range was to use the Lidar-Lite sensor for front facing obstacle detection. Unfortunately, the line of sight of the device hits the ground before it can reach

its maximum range. We have resorted to using the RPLIDAR for the object detection range. The data comes from a ROS LaserScan which has an array of ranges. The data for this specification test has been exported to a .mat file in Matlab for easy viewing. Figure 61 shows the ranges (in meters) that the RPLIDAR can detect objects for. In the case below, we have data that shows objects at 11m away from the truck, which passes the test we set out for ourselves.

Data:98	Data:99	Data:100	Data:101	Data:102	Data:103	Data:104
10.8510	11.0570	11.1390	11.1390	10.9190	10.4710	10.5880
10.8510	11.0570	11.1390	11.1390	10.9190	10.4710	10.5880
Inf	10.9650	11.2350	10.9650	10.7510	10.5240	10.2360
Inf	10.9650	11.2350	10.9650	10.7510	10.5240	10.2360
10.9990	10.9650	11.2230	11.1040	11.0110	10.6850	10.5130
10.9990	10.9650	11.2230	11.1040	11.0110	10.6850	10.5130
10.6850	10.9990	Inf	10.7400	11.1510	10.6410	10.3160
10.9190	10.8850	11.1870	11.1160	10.8060	10.5030	10.5340
10.9190	10.8850	11.1870	11.1160	10.8060	10.5030	10.5340
Inf	11.1040	10.7400	11.3820	11.3080	10.9070	10.5980
10.2860	11.1750	11.0920	10.9300	10.7730	10.5130	10.4500
10.2860	11.1750	11.0920	10.9300	10.7730	10.5130	10.4500
10.9190	10.8060	10.8960	10.9420	10.7180	10.6310	10.3160
10.5560	11.0450	10.9760	11.5070	10.9300	10.6090	10.3980
10.5560	11.0450	10.9760	11.5070	10.9300	10.6090	10.3980
11.2470	11.0810	Inf	11.0220	10.9880	10.9300	10.4190

Figure 61: Maximum range detected by the RPLIDAR of 11m

7.1.11 Object Detection Angle - (31)

Our object detection angle was determined primarily by testing the angles that the RPLIDAR was able to view. This data, which is in the form of a ROS LaserScan datatype, was analyzed in MatLab. Figure 62 shows the different types of information contained within the structure, including the starting and ending (minimum and maximum) angles of the scan, the angle increment, and the ranges.

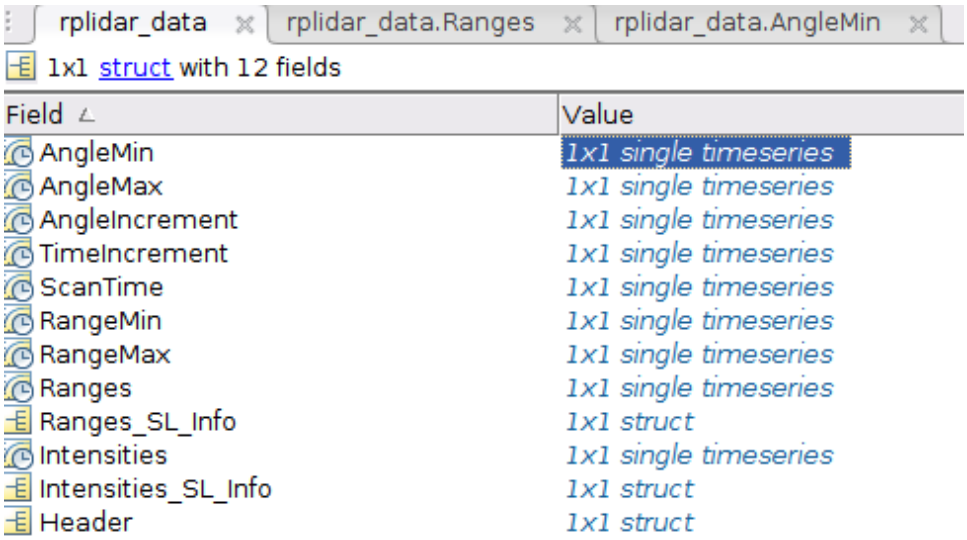


Figure 62: Data contained within the LaserScan for the RPLIDAR

The starting angle, along with the angle increment can be used to generate an array of angles for an entire scan. The output of this calculation matches up with the Ranges array that is built into the ROS datatype, seen in Figure 63. This array shows for each timestep, an array of roughly 320 ranges, which are the distances to the nearest object in meters. Any cell with the value “Inf" means that the LIDAR could not detect an object within its 12m range. Based on the output of the data, the LIDAR shows angles for a full 360-degree view but is obstructed by the motherboard casing for 30 degrees. As such, our system has achieved its goal of a detection angle of 180 degrees.

Time series name: Ranges

Time	Data:1	Data:2	Data:3	Data:4	Data:5	Data:6	Data:7	Data:8	Data:9	Data:10
0	5.9500	5.9840	5.9840	5.9700	6.0470	6.0780	Inf	6.1860	6.3320	6.3480
0.1000	5.9500	5.9840	5.9840	5.9700	6.0470	6.0780	Inf	6.1860	6.3320	6.3480
0.2000	5.9570	5.9400	6.0360	6.0260	6.0220	6.1390	6.2530	6.2010	6.3440	6.3550
0.3000	5.9570	5.9400	6.0360	6.0260	6.0220	6.1390	6.2530	6.2010	6.3440	6.3550
0.4000	5.9300	5.9880	6.0050	5.9980	6.0680	6.1570	6.1070	6.3210	6.1720	6.2750
0.5000	5.9300	5.9880	6.0050	5.9980	6.0680	6.1570	6.1070	6.3210	6.1720	6.2750
0.6000	5.9640	5.9370	6.0610	6.0050	6.0920	6.0820	6.1350	6.1720	6.2530	6.2600
0.7000	5.9880	5.9670	6.0470	6.0150	6.0920	6.1070	6.1860	6.1750	6.2560	6.3130
0.8000	5.9880	5.9670	6.0470	6.0150	6.0920	6.1070	6.1860	6.1750	6.2560	6.3130
0.9000	5.9670	5.9500	6.0400	6.0120	6.0960	6.1860	Inf	6.2190	6.3280	6.2600
1	5.9640	5.9570	6.0220	6.0330	6.0010	6.1140	6.1420	6.2420	6.3090	6.2340

Figure 63: The “ranges” data within the LaserScan

7.1.12 Battery Life - (32)

The battery life was confirmed to meet our specification during the Senior Project Expo. We had a demo running where the motor would actuate different levels based on a signal coming from the Lidar Lite’s detection of an object in front of the vehicle. The Expo lasted three hours, and for two of the hours the

platform was running. We estimate the motor and servos had been running for longer than the 30 minute limit for passing the battery life specification.

7.1.13 Software Application Layer - (33)

The software application layer requires that there be a library of easy-to-use functions for the user to call the interface with the sensors and actuators on the vehicle. The layer has been written in the semi_truck_api.cpp file, and future users should #include the semi_truck_api.h (in daimtronics/semi_catkin_ws/src/semi_truck/include) file when writing their algorithms to control the vehicle. The application layer consists of "setter" and "getter" functions to read from and write to each of the different sensors and actuators hooked up to the Teensy. As such, we have passed the test that actuators and sensors can be interfaced with through our API. As an example, Figure 64 shows the code for the call to set the motor output that is ultimately sent to the Teensy's motor_driver task.

```
/**
 * @brief sets the motor output of the actuators object to the passed in value
 * @param actuators a reference to a TeensyActuators object to alter
 * @param value the value to update the actuators with. This value should
 * range from 0 for full reverse power to 180 for full forwards power.
 */
void set_motor_output(semi_truck::Teensy_Actuators &actuators, int16_t value);
```

```
void set_motor_output(semi_truck::Teensy_Actuators &actuators, int16_t motor_output){
   actuators.motor_output = motor_output;
}
```

Figure 64: Application layer example for controlling the motor output

This application layer requires the pi_comm_node to be running and transmitting data between the Pi and the Teensy for the application layer function calls to work with data in real time.

7.1.14 Documentation Rating - (34)

We were unable to receive feedback from other mechatronics students on the documentation for this project due to time constraints. Because we were working on getting the full system to run up until the Senior Project Expo, we did not have time to meet with other students to test out the entire system.

7.1.15 Voltage Safety Rating - (35)

The minimum voltage rating for all electrical components at and around the battery inputs of the motherboard is 35V, slightly greater than the 30V minimum required for a safety factor of 2 with nominal battery voltages of 15V. Using a DC power supply, we tested an input of 20V, which we consider to be a reasonable extreme upper limit, to confirm the robustness of the motherboard. We do not see the benefit of directly testing further since we have only one motherboard currently available, and consider this test passed for our intents and purposes.

7.1.16 Replacement Test Time - (36)

Replacing the peripheral time-of-flight sensors takes about four minutes, including removing from mounting, removing wires, replacing wires, and putting back into mounting. [XX Include time to replace both upper and lower batteries, then call good]. This is beneath our target of 20min, satisfying this test.

7.2 Intermediate Testing

Intermediate testing encompasses the individual component testing and verification and was accomplished throughout the project, before the full assembly. The number in parentheses in the section title is the corresponding item number in the DVPR, Appendix J.

7.2.1 Motherboard Testing - (4 through 9)

The motherboard has been tested for any power and signal shorts, and successfully provides power to the Teensy, Pi, attached sensors, and the steering servo. The Teensy can successfully enable and disable the motor-side 5V regulator and the Pi can switch the relay between sourcing signals from the receiver (manual mode) and from the Pi (automatic mode). However, the I2C multiplexer is currently not functional. The motherboard passed all intermediate testing except for the multiplexer.

7.2.2 Ultrasonic Calibration - (12)

The URFs can determine a distance by emitting a sound and determining the amount of time it takes for the sound to return from an object. The URFs come with a recommended scaling factor of 58 microseconds per centimeter. We were able to collect data on this output timing and compare it with known distances to further calibrate and verify the URF timing scale.

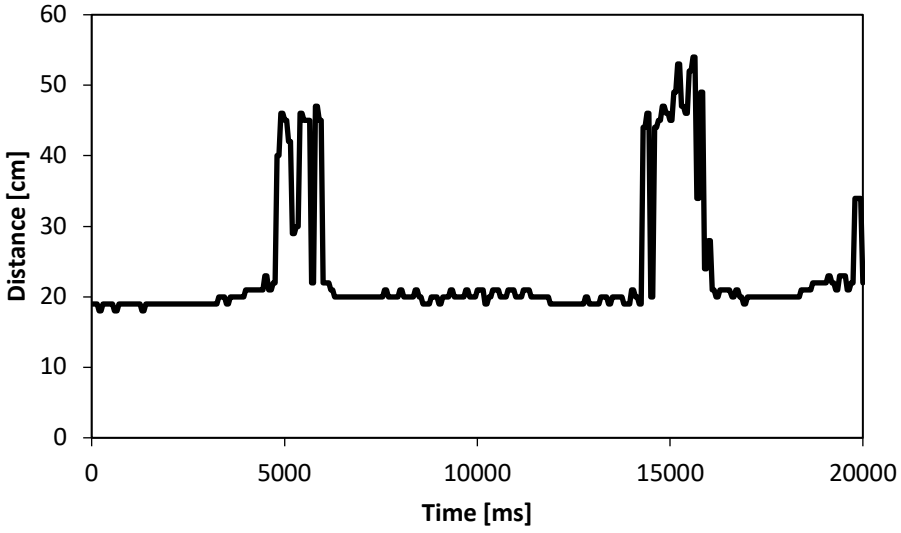


Figure 65: Ultrasonic reading a distance for an object placed square to the sensor and 20 cm away

In order to further confirm that the URFs were correctly reading distance, we tested a variety of different objects and ran the Ultrasonic script on the Teensy to output distance. We compared these distances with those found with a tape measure. The URFs were able to determine an accurate distance every few readings, but it was not consistent. Over time, random spikes in measured distance would be outputted by the URFs. A simple filtering script could be written to improve this accuracy, but we noticed that as objects became less square to the sensor, readings would go off the scale and become far too inaccurate.

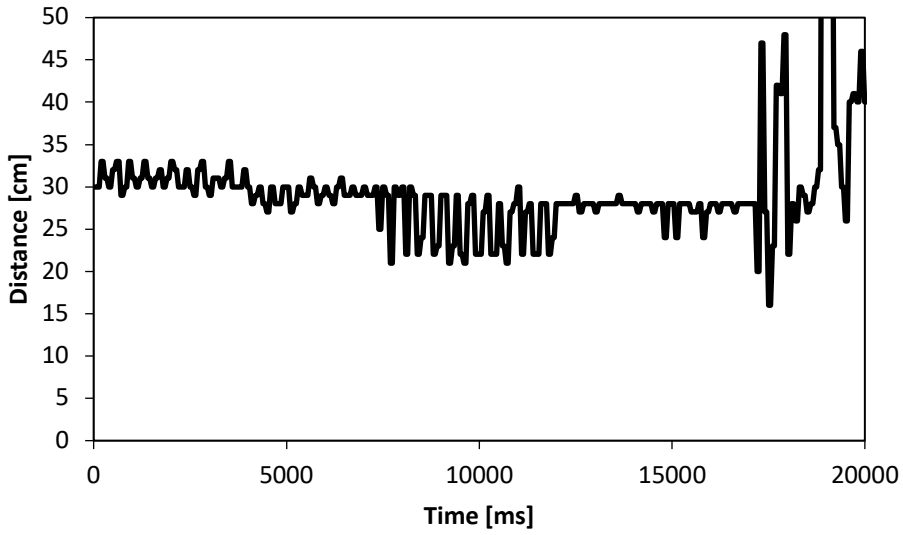


Figure 66: The same object tested at an angle of 30 degrees to the sensor and 20 cm away

An object at around a 30-degree angle to the URF would send data too inaccurate for a filter to properly function. For the same object rotated 30 degrees and 20 cm away from the sensor, Figure 66 shows that on average, the distance measured is about 30 cm. Additionally, any object that does not reflect sound easily, such as fabrics or people, introduces more noisy data that makes filtering less effective. For these reasons, we have determined that the Ultrasonic Range Finders have failed this test and have replaced them with an Adafruit VL53L0X Time of Flight sensor.

7.2.3 Lidar Lite Calibration - (13)

The Lidar Lite is successfully able to read distances to the nearest object directly in front of the semi-truck. This is achieved by simply using the ROS driver that cocasema [53] has written. It publishes its message as a ROS sensor_msgs/LaserScan message. The data is able to be read by Simulink when using External Mode. Unfortunately, we ran into issues that the line-of-sight of the sensor will hit the ground 5-7 meters in front of the vehicle. This is primarily due to the fact that the suspension tilts the vehicle downwards now that the entire computing system and batteries are mounted. This testing was not done with the trailer attached, so it is possible that the addition of the trailer will correct the line of sight issues that we were encountering. We have suggested some actions to take for fixing this problem in the Next Steps portion.

7.2.4 RPLidar Verification - (14)

The RPLidar successfully outputs its point-cloud data in a format that can be parsed in a way that is useful to a user. Currently its output is divided into equal radial zones, with the output of the sensor being the distance of the nearest reading within each zone. The data is currently nested within a ROS message that helps maintain structure and meta-data about the Lidar scan. Similar to the Lidar Lite, this sensor is controlled by a prebuilt ROS driver that outputs sensor_msgs/LaserScan data according to the ROS standard. There is an array of Ranges that contains the distance to the nearest objects in the 360 degree range that it is capable of. We have made suggestions in the Next Steps section on how to leverage the data coming from the RPLidar

7.2.5 Mounting Robustness - (10)

Due to completing the full platform relatively late in our planned timeline, we did not have time to extensively test the mounting robustness. We had some troubles with 3D printing the mounting, and this was one of the final items completed for our project. We subjected the platform to vibrations from manually shaking the tractor portion of the semi-truck. We did not do any crash simulations with the system, and would suggest future teams be mindful of nearby objects to the truck when starting out with testing autonomous algorithms.

7.2.6 RC Receiver Reading - (15)

The Teensy can successfully distinguish between multiple servo-style PWM signals coming from the receiver. This is currently being used to send an enable/disable signal from the dead man switch on the RC controller to the Teensy and to send whether to be in manual or automatic mode.

7.2.7 Relay Fidelity - (18)

We have tested that the relay is able to switch between the two PWM sources and pass it through to the servos and motor with sufficient quality. Originally, we were concerned that the Pi would not be able to hold the relay high due to a high current sourcing requirement, but the two solid-state relays have since been replaced with a single larger electromechanical relay that controls both the steering and throttle and consumes less current than the two combined solid-state relays. We sent throttle and steering commands from the RC receiver through the motherboard to the motor and servo, comparing the signals before and after the relay with an oscilloscope, taking pre-relay to be ground and post-relay as the measurement. The signal read as almost identically zero, confirming that the relay does not degrade the signal quality.

8. Project Management

In order to ensure that the requirements of Daimler and Dr. Birdsong are met, this project was conducted in close connection with our sponsors in order to maintain tight feedback loops as we progressed.

8.1 Project Overview

To facilitate the development of this project, we conducted research on alternative and preexisting autonomous semi-truck research solutions. By evaluating the preexisting literature and consulting with professors knowledgeable in the required areas, we were able to tune our ideation and were able to focus on the technical decisions needed to produce an effective research platform. This work along with sponsor feedback guided our development of the critical design review report, which was presented to the sponsor. After obtaining design approval, we will begin integrating the parts of our design to begin testing as soon as possible.

As of Winter 2019, this project is now fully owned by the Daimtronics team due to the completion of Daimscale's senior project. Weekly meetings with Daimler and Dr. Birdsong helped guide us to completing the platform.

8.2 Project Execution

Our team utilized Asana, a task management platform, for planning and organizing work done outside of meetings, to track small details and work that needs to be done, and to organize relevant information and documents that are necessary for work completion but not necessarily required material for the project. The overall project timeline was tracked in TeamGantt shown in Appendix B. The weekly team meetings consist of time spent completing objectives and of planning what to have done individually by the next meeting or by particular lab dates. Table 8 below spells out the key deliverables and their due dates.

Table 7: Key Deliverables and Expected Due Dates

Deliverable	Description	Due Date
Initial Task Diagrams	Multitasking design diagrams for microcontrollers. Includes relationship between boards.	November 1, 2018
Preliminary Design Review (PDR)	Report of team progress, review of initial solution designs.	November 11, 2018
Part Selection & Implementation	Final selection of sensors and actuators with plan for board integration.	December 2018
Interim Design Review (IDR)	Project update report including the results of the Daimscale project and our design progress	January 17, 2019
Manually-Operated Platform	Assembly of all actuators and sensors. Manual control-capable.	February 2019
Critical Design Review (CDR)	Presentation and report of information for final design and plan for completion.	February 8, 2019
Basic Autonomous Software	Completion of rudimentary software to demonstrate autonomous capabilities.	April 30, 2019
Final Design Review (FDR)	Detailed report of all design considerations and evaluation of product capabilities.	May 30, 2019
Senior Project Expo	Public presentation of the state of the project and design.	May 31, 2019
Final Vehicle Delivered	Completed platform with aforementioned software and bill of materials.	June 6, 2019

The Gantt chart in Appendix B shows the finalized completion time for each deliverable and the dependencies of each aspect of the project. This was utilized in conjunction with Asana to keep the project moving at an appropriate pace.

The Gantt chart helped us to develop internal deadlines for the project and Asana helped to track details and resources. As the project progressed, we were less strict about tracking our progress as we focused on trying to get the work done rather than documentation. While this did not result in any obvious detriment to our team and the project, it is a weak spot we all are taking into account when we go into our next design project. Having the clearest available picture of the project status is always useful in determining how best to allocate resources.

9. Conclusion

This report is a culmination of all the information discovered throughout our year-long endeavor to build a platform for testing autonomous vehicle navigation algorithms. This version updates the details of our final design, our outcomes for motherboard, mounting and software manufacturing, and describes the results of testing and design verification. Our revised final design includes more appropriate sensor models, more easy access for assembling mounting for the platform, and a second design and manufacturing cycle of the motherboard.

As is typical with Senior Projects, especially those revolved around mechatronics applications, we were fairly limited in time towards the end of the year. We only got our full system assembled by the day before the Senior Project Expo, and this did not leave us much time to do full system testing. While we were diligent in conducting and documenting the intermediate tests that we had set up on the path to a full system, we could have used a few more weeks to iron out all of the loose ends in the platform. In hindsight, it would have been best to get the motherboard and mounting completed halfway through our third quarter, rather than a week before the quarter ended. Furthermore, we could have frontloaded some more of our development on the sensor interfacing to free up time for system testing. With this being said, we are proud of the work we have completed, and are looking forward to hearing how the future users interact with the platform.

9.1 Next Steps

The project is ready for another group of engineers to start working with the platform on autonomous vehicle algorithms. The entire computing systems are hooked up to each other and passing data for controlling the semi-truck and sensing the truck's surroundings. While the next team could start working on in-depth control models (either in Simulink or programmatically in ROS) we suggest that some modifications are made to our final design.

9.1.1 Motherboard

There are several changes that need to be made for the next version of the motherboard. The relay traces need to be adjusted, the I2C multiplexer and accelerator circuitry needs to be diagnosed, the motor hall sensor data port pins need to be fixed, the 5V analog circuitry needs to be tested, power switches should be added to either the board or the battery-to-board wires, the pin enabling the motor-side 5V regulator should either be disconnected or routed to a different pin on the Teensy, a reset pushbutton should be added for the Teensy, and we recommend adding a protoboard area.

The negative pin of the relay's actuating coil needs to be rerouted from ground to a pin on the Pi so that reverse voltages can be applied. A pin next to the current control pin is preferable.

The I2C multiplexer and accelerator currently do not detect any devices downstream of the multiplexer. The cause of this is currently unknown. One known change that should be made is to reroute the multiplexer's channel from I2C channel 3 on the Teensy to any other channel, as channel 3 is an additional channel specific to the Teensy that is not fully supported in across a variety of I2C Arduino libraries.

The supply and temperature pins of the JST ZHR-6 Eagle device for the motor hall sensor data port are switched in the connections between its footprint and its schematic. The device file has been fixed already, so fixing the schematic and traces then printing the new board are the only actions needed to fix the issue.

The 5V analog input port has not been tested due to none of our sensors needing it. The converter chip communicates 5V SPI to the logic shifter chip, which translates the 5V SPI to 3.3V SPI to the Teensy. An SPI driver will need to be written for the Teensy to handle communication with this chip.

Switches to control the power supply of both batteries without having to plug and unplug the connectors every time will save some time. If the switches are attached in-line with the wires then they can improve the safety of assembling the vehicle by preventing the bare wire-to-board leads from sparking when connected to the battery.

The Teensy pin connected to the enable pin of the motor-side 5V regulator happens to also be the pin that controls the LED on the Teensy. This means the LED is not available for diagnosing the Teensy since it needs to be held high to keep the motor power on. This pin should be routed elsewhere or disconnected if control over the motor-side regulator is deemed not necessary.

A board pushbutton should be added to operate the reset pin of the Teensy. Currently the only way to reset the Teensy is to disconnect and reconnect the entire computing power supply, which also resets the Pi. The Pi takes much more time to restart than does the Teensy, unnecessarily increasing the time it takes to reset the platform.

Finally, we recommend adding a protoboard section to the motherboard. This is a grid of through-holes with connections similar to how a breadboard is set up. This allows for easy additions of unforeseen components or components that are very specific to a particular sensor or actuator, such as the capacitor on the Lidar Lite. This replaces the prior, unimplemented idea of using mini breadboards adjacent to the motherboard for this purpose, since a protoboard allows for easy soldering and will not allow components to come loose, a concern with using breadboards.

Our team will be available to advise whoever is working to implement the above; this document alone is not expected to be sufficient to implement the changes unless the next developer has prior PCB design experience.

9.1.2 Mounting

During testing, we found that with the addition of the motherboard, sensors and batteries, the front suspension was closer to the ground than we had expected. This resulted in the front facing LIDAR-Lite's line of vision to intercept the ground at approximately 7-8 meters. While this was not a deal-breaker for the max object detection range specification, we believe that a team could implement an easy fix to this

problem. We would suggest reprinting the mounting for the LIDAR-Lite so that the emitted laser travels as close to parallel with the ground as possible.

Another improvement to the mounting for the system is including some protection for the Lidar-Lite located at the front of the vehicle. As it stands, the sensor is slightly exposed (sticks out about an inch past its mounting) and is vulnerable to any head on collisions with other objects. The fix for this would be pretty simple, and would include taking the existing solid models, and extend the front edge of the model so that the Lidar-Lite does not protrude out. In the meantime, users should be cautious while working with the semi-truck, and should try to avoid any high speed collisions with objects.

The columns supporting the rear of the second battery housing need to be adjusted to be able to clear the finger tabs holding in the first battery without any modifications.

The 5th wheel servo currently does not have a linkage connecting it to the actual 5th wheel spring and lever. A thick, stiff wire should be sufficient since the servo only needs to pull on the lever; it should never have to be actively pushing the 5th wheel closed. This is likely close to an optimal solution, though a more in-depth linkage can be designed by a future user if desired.

9.1.3 Software

While we have successfully setup the RPLIDAR and read its value through both the ROS and Simulink methods, we have not formatted the information in any way. Currently, the data resides within a LaserScan ROS message. This message has a variety of different values within its structure, including the minimum and maximum angles, the angle increment, and an array of both ranges and intensities. We suggest transforming the minimum angle and angle increment to generate an array of angles (from -180° to 180°) that is directly related to the ranges array within the data type. Both of these arrays will be of the same size, and easy to use to map out the surroundings. We also suggest setting up polar plots to help visualize the data in a point-cloud format.

Sometimes the serial communication between the Teensy and the Pi gets into an error state where the data is mismatched, even with the advent of sending a synchronizing value (see the source code comments in the serial communication files for info). This will be fairly obvious to users, as data will not be updated even when the system is running. The fix for this is to relaunch all of the nodes using the command “roslaunch semi_truck_bags.launch” in the Raspberry Pi terminal. When the nodes are restarted, and more specifically the pi_comm_node, the data will be reset to proper transmission. We would recommend future users to find a more consistent way of avoiding this scenario by looking into the code and identifying any possible bugs.

Adding software for additional devices (sensors or computing platforms) is likely going to be desirable for future teams. If the device has a community supplied ROS driver, it is probably easiest to hook up that peripheral to the Raspberry Pi. The Teensy is more limited in its abilities to service all of its peripheral devices than the Raspberry Pi. Especially when using Simulink in External Mode (where the PC takes a lot of strain off of the Raspberry Pi) we suggest trying to interface additional devices with the Raspberry Pi. In regards to adding another computing platform, such as an NVIDIA Jetson, this should be relatively straight forward. ROS has done a lot of the legwork for communicating data between different devices

with their own IP addresses. As we found later in development of our project, the ROS network that was originally intended to be only on the Pi actually extended to the PC. The same idea would apply to any other computing platforms, where ROS subscribers and publishers can specify IP addresses to connect to.

9.2 Final Words

Our team is grateful for the opportunity to work on this project and we look forward to seeing its evolution in the future! We hope we have provided a capable foundation that is easy to work with. All members of our team intend to be available to advise future groups working with this platform or related projects. We can be reached at our individual non-academic emails and our group team email, listed on the cover page of this report.

Daimtronics Team

References

[1] Daimler Press Release. (2015). Freightliner Inspiration – the first licensed autonomous driving truck in the US. Stuttgart, Germany: Daimler Communications.

[2] Liburdi, Andrew. (2010). *Development of a Scale Vehicle Dynamics Test Bed.* Electronic Theses and Dissertations. Paper 195. University of Windsor.

[3] Gonzalez, J. "JuliaCon 2016 | Autonomous Driving for RC Cars with ROS and Julia | Jon Gonzales." *YouTube*, The Julia Language, 30 June 2016, youtu.be/bX4TXWO7dA0.

[4] Guldner, Owen, and Sertac Guldner. *RACECAR*, MIT RACECAR, 2017, mit-racecar.github.io/.

[5] "Tamiya America." Tamiya RC Freightliner Cascadia Evo, Tamiya America, https://www.tamiyausa.com/shop/114-tractor-trucks/rc-freightliner-cascadia-evo/

[6] "Billet Machined Rolling Chassis for Custom 1/14 Semi-Tractor Truck." Team Integy, https://www.integy.com/st_prod.html?p_prodid=22091#.W9ZdEzNlCUl

[7] Albert, Christian. Electronic Control Device for Controlling Autonomously Controllable Assemblies. 28 Jan. 2003.

[8] Coverdill, Cary, et al. Truck with Monitored and Resettable Electronic Control Unit. 30 Mar. 1999.

[9] Sharma, Manuj, et al. *Calibration of Multi-Sensor System*. 16 Sept. 2010.

[10] 汪迎春, et al. *Laser Distance Measuring Equipment.* 17 July 2018.

[11] Trepagnier, Paul Gerard, et al. Control and Systems for Autonomously Driven Vehicles. 28 Feb. 2012.

[12] Grant, C., Miley, J., & Phillips, E. (2017). *Small-Scale Intelligent Vehicle Design Platform* (Rep.). San Luis Obispo, CA: California Polytechnic State University.

[13] De Rosier, S., Riccoboni, D., & Rothhammer-Ruiz, P. (2018). *Small Scale Intelligent Vehicle Final Design Report* (Rep.). San Luis Obispo, CA: California Polytechnic State University.

[14] Bodmer, D., Marrale, C., & Sasaki, J. (2018). *Daimscale Critical Design Report* (Rep.). San Luis Obispo, CA: California Polytechnic State University.

[15] Tekin. (n.d.). RX8 GEN 3 1:8 Scale Race Electronic Speed Control. Retrieved October 18, 2018, from https://www.teamtekin.com/product8.html

[16] Raspberry Pi. (n.d.). Raspberry Pi 3 Model B. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[17] Digikey. (2017, January 05). Teensy 3.5 Development Boards. Retrieved November 2, 2018, from https://www.digikey.com/en/product-highlight/s/sparkfun/teensy-3-5-development-boards

[18] Litman, T. (2018). Autonomous Vehicle Implementation Predictions. *Victoria Transport Policy Institute*. Retrieved November 2, 2018, from https://www.vtpi.org/avip.pdf.

[19] Katrakazas, C., Quddus, M., Chen, W., & Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C,* 416-442. Retrieved November 4, 2018.

[20] Ainhauser, C., et al. (2013, November 5). Autonomous Driving Needs ROS. Retrieved October 31, 2018, from http://www.bmw-carit.com/downloads/presentations/AutonomousDrivingNeedsROSScript.pdf

[21] Chen, et al. "Cyber-Physical System Based Optimization Framework for Intelligent Powertrain Control." *SAE International Journal of Commercial Vehicles*, vol. 10, no. 1, 2017, p. 254+. *Academic OneFile*, Accessed 4 Nov. 2018.

[22] V. A. Butakov and P. Ioannou, "Personalized Driver/Vehicle Lane Change Models for ADAS," in IEEE Transactions on Vehicular Technology, vol. 64, no. 10, pp. 4422-4431, Oct. 2015.

[23] HardKernel. (n.d.). Odroid-C2. Retrieved from https://www.hardkernel.com/shop/odroid-c2/

[24] Arduino. (n.d.). Arduino Uno Rev3. Retrieved from https://store.arduino.cc/usa/arduino-uno-rev3

[25] Nvidia. (n.d.). NVIDIA JETSON AGX SYSTEMS. Retrieved from https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[26] Maxon Motor ag. (2012). Maxon EC Flat Motors. Retrieved November 15, 2018, from https://www.maxonmotorusa.com/maxon/view/content/ec-flat-motors

[27] HobbyKing. (n.d.). Turnigy Trackstar Brushless Motor. Retrieved November 15, 2018, from https://hobbyking.com/en_us/turnigy-trackstar-1-8th-sensored-brushless-motor-1900kv.html

[28] HobbyKing. (n.d.). Multistar Elite Motor. Retrieved November 15, 2018, from https://hobbyking.com/en_us/multistar-elite-4006-740kv-multi-rotor-motor.html

[29] Malta, Delaney Michelle. “Preliminary Sensor Ranges for Autonomous Research Buggy.” 2019.

[30] Meetup.com, 17 May 2018, www.meetup.com/ai-ml-for-robotics-and-iot/messages/boards/thread/50451824?_cookie-check=dyasZp3LJovouPiq.

[31] ROS Start Guide. (n.d.). Retrieved from http://wiki.ros.org/ROS/StartGuide

[32] United States, Congress, Office of Vehicle Safety Compliance. “Electronic Stability Control Systems for Heavy Vehicles” Nhtsa.gov, 10 Feb. 2016.

[33] Li, L., Wang, F., (2007). “Advances in Vision Based Vehicle Detection.” *Advanced Motion Control and Sensing for Intelligent Vehicles*. Springer Science & Business Media.

[34] RobotShop. (n.d.). RPLIDAR A2M8 360 Laser Scanner. Retrieved from https://www.robotshop.com/en/rplidar-a2m8-360-laser-scanner.html

[35] Slamtec. (2017, May 15). RPLIDAR Introduction to Standard SDK. Retrieved January 29, 2019, from http://bucket.download.slamtec.com/351a5409ddfba077ad11ec5071e97ba5bf2c5d0a/LR002_SLAMTEC_rplidar_sdk_v1.0_en.pdf

[36] Garmin. (2018, April). LIDAR-Lite 3 Laser Rangefinder High Performance (LLV3HP). Retrieved January 29, 2019, from https://cdn.sparkfun.com/assets/9/a/6/a/d/LIDAR_Lite_v3HP_Operation_Manual_and_Technical_Speci fications.pdf

[37] ElecFreaks. (n.d.). HC-SR04 User Guide. Retrieved January 29, 2019, from www.elecfreaks.com

[38] *LM2678 Datasheet*. Texas Instruments, Feb. 2017, www.ti.com/lit/ds/symlink/lm2678.pdf.

[39] "Kinetis K66 Sub-Family Datasheet." Freescale Semiconductor, Inc., 1 Jan. 2016. https://www.pjrc.com/teensy/K66P144M180SF5V2.pdf

[40] "FAQs." *Raspberry Pi FAQs*, Raspberry Pi Foundation, www.raspberrypi.org/documentation/faqs/#pi-power.

[41] *BNO055 Datasheet*. Bosch Sensortec, Nov. 2014, cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf.

[42] *QRE1113 Datasheet*. Sparkfun, Sept. 2009, www.sparkfun.com/datasheets/Robotics/QR_QRE1113.GR.pdf.

[43] *HC - SR04 Datasheet*. EleFreaks, cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[44] *Lidar Lite V3 Operation Manual*. Garmin, Oct. 2016, static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf.

[45] *RPLidar A2M8 Introduction and Datasheet*. Slamtec, 15 May 2017, cdn.sparkfun.com/assets/e/a/f/9/8/LD208_SLAMTEC_rplidar_datasheet_A2M8_v1.0_en.pdf.

[46] *ProTek RC 100T Standard Digital "High Torque" Metal Gear Servo*. AMain Hobbies, 9 Oct. 2012, www.amainhobbies.com/protek-rc-100t-standard-digital-high-torque-metal-gear-servo-ptk-100t/p228478.

[47] *WEBENCH® Power Designer* . Texas Instruments, www.ti.com/tools-software/design-center/webench-power-designer.html.

[48] Sweethome, and goldilocks. "Could Voltages over 5v Damage a Pi?" *Raspberry Pi Stack Exchange*, Stack Exchange, 17 Oct. 2016, 19:01, raspberrypi.stackexchange.com/questions/56435/could-voltages-over-5v-damage-a-pi.

[49] *Teensy Pinouts*. PRJC, www.pjrc.com/teensy/pinout.html.

[50] Robopeak. (2018, August 24). RPLIDAR ROS. Retrieved from https://github.com/robopeak/rplidar_ros

[51] Cocasema. (2017, January 29). Ros-lidar-lite. Retrieved from https://github.com/cocasema/ros-lidar-lite

[52] MathWorks. (n.d.). Get Started with ROS in Simulink. Retrieved February 2, 2019, from https://www.mathworks.com/help/robotics/examples/get-started-with-ros-in-simulink.html

[53] Iacono, S. D. (2018, December 11). A detailed explanation of multithreading in ChibiOS/RT. Retrieved from https://www.playembedded.org/blog/explanation-multithreading-chibios/

[54] Adafruit. (2018, December 3). Adafruit_BNO055. Retrieved from https://github.com/adafruit/Adafruit_BNO055

[55] Bosch Sensortec. (2014, November). BNO055 Intelligent 9-axis absolute orientation sensor. Retrieved from https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf

[56] ROS. (n.d.). ROS. Retrieved from http://wiki.ros.org/ROS/

[57] PJRC. (n.d.). Tutorial 1: Software Setup. Retrieved from https://www.pjrc.com/teensy/tutorial.html

[58] ChibiOS. (n.d.). How to create a thread. Retrieved from http://wiki.chibios.org/dokuwiki/doku.php?id=chibios:howtos:createthread

[59] Malta, Delaney Michelle. “Final Sensor Ranges for Autonomous Research Buggy.” 2019.

Appendices

Appendix A: QFD House of Quality

Appendix B: Gantt Chart

Appendix C: Computing Platform Decision Matrix

Appendix D: Motor Decision Matrix

Appendix E: Middleware Decision Matrix

Appendix F: Sensor Selection Process

Appendix G: Design Hazard Checklist

Appendix H: Overall Power Routing

Appendix I: Webench LM2678 Design Report

Appendix J: DVPR

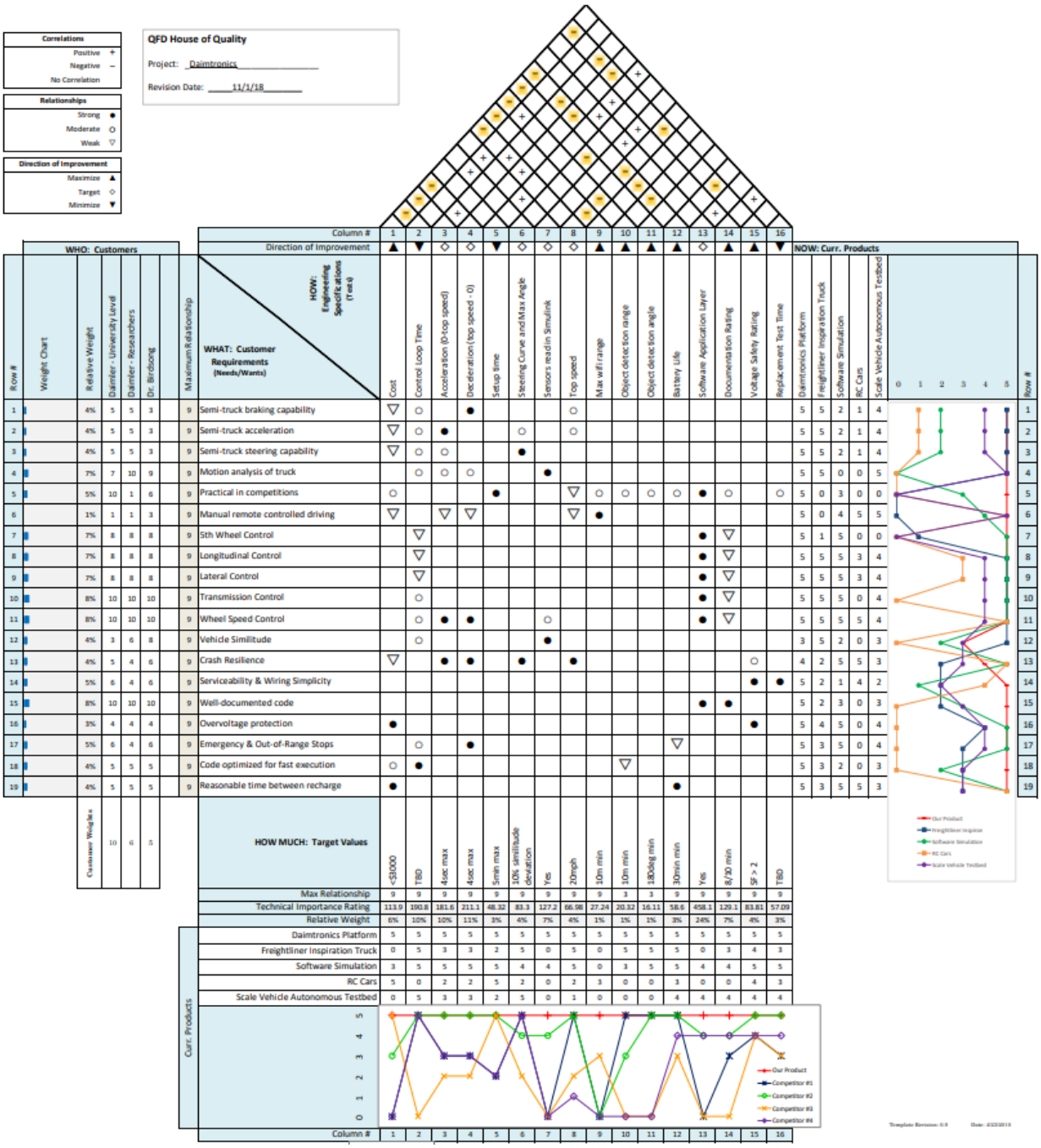
Appendix K: Drawing Package

Appendix L: Speed Uncertainty Analysis

Appendix M: Daimtronics Operator's Manual

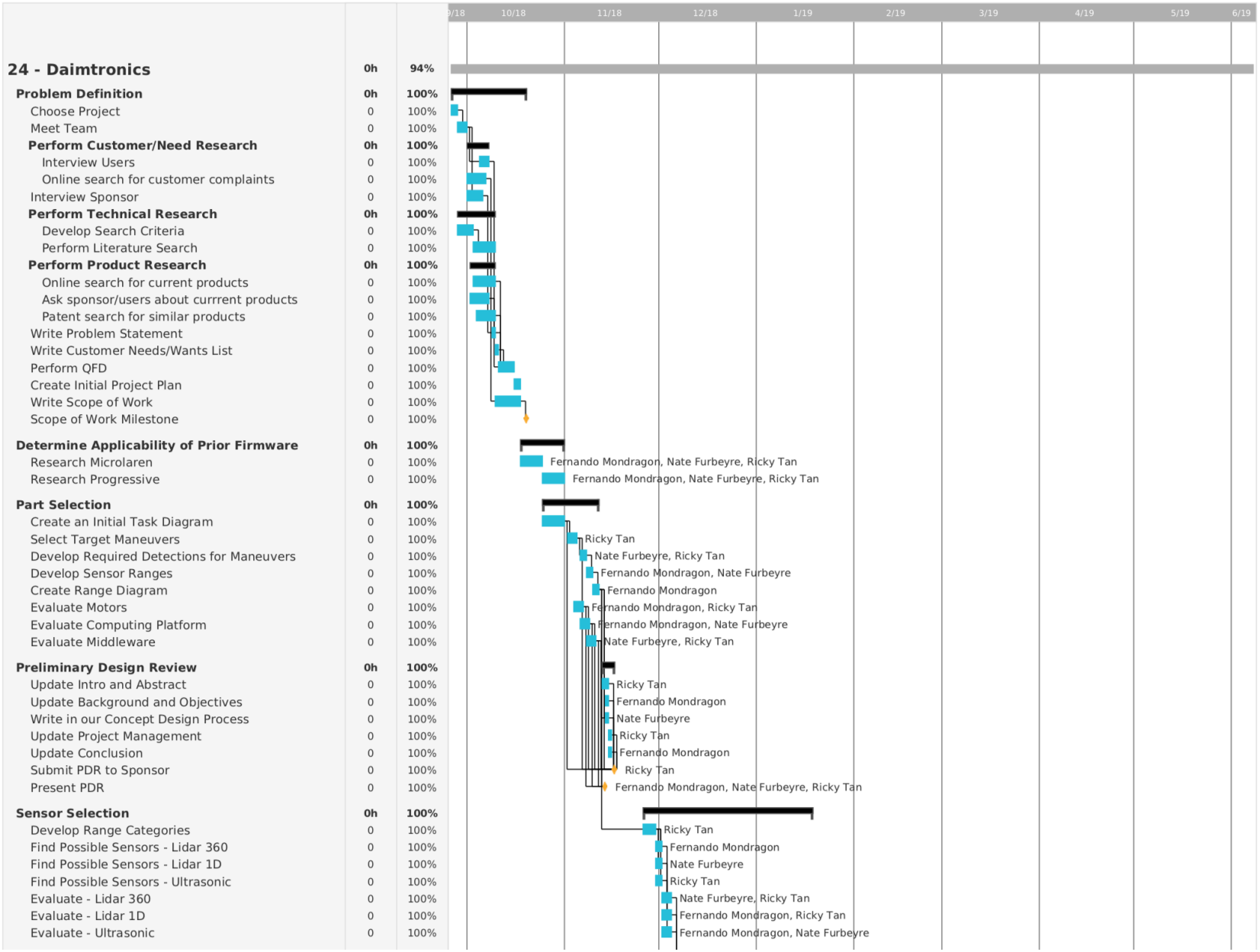
Appendix N: Doxygen Report for Software Documentation

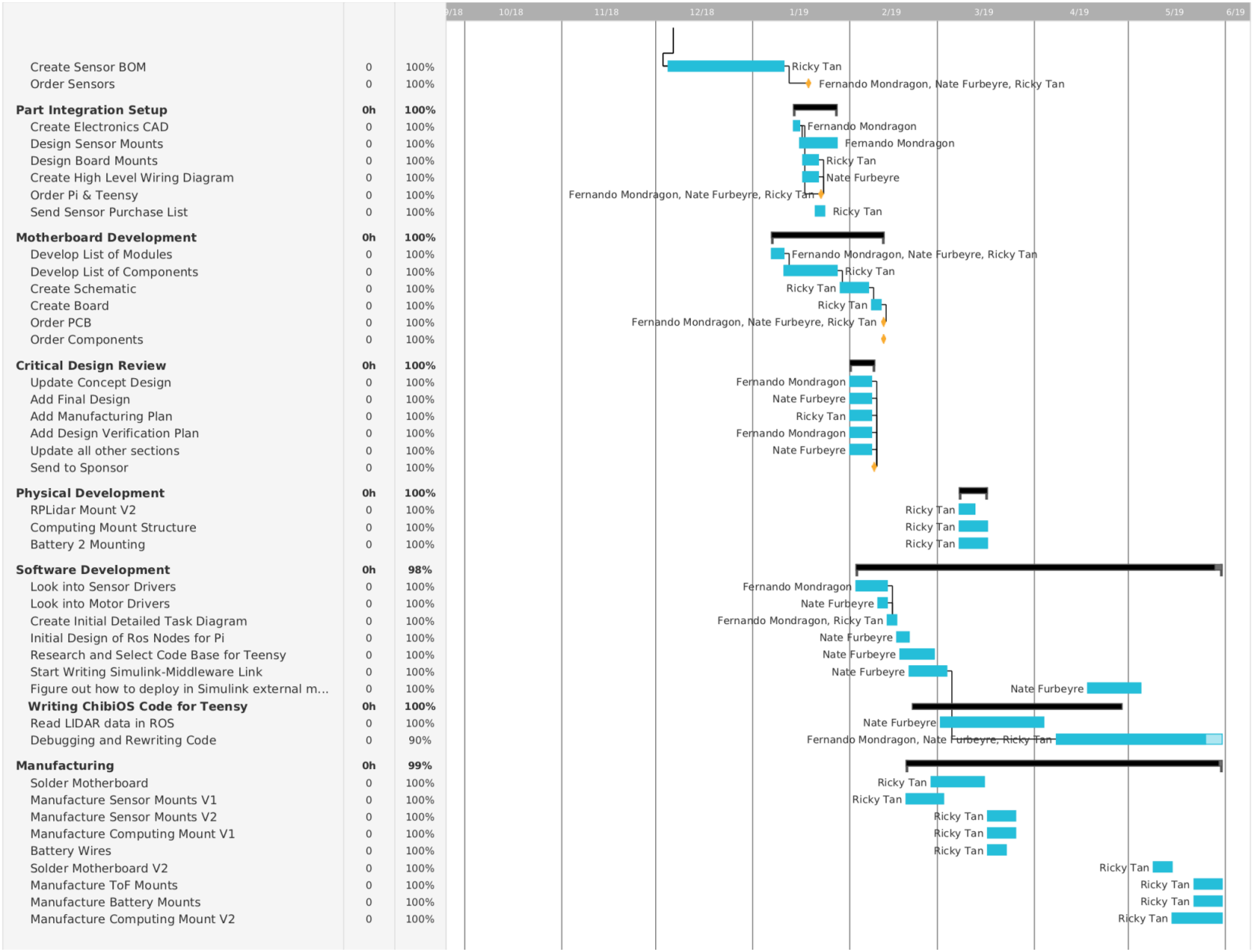
Appendix A: QFD House of Quality



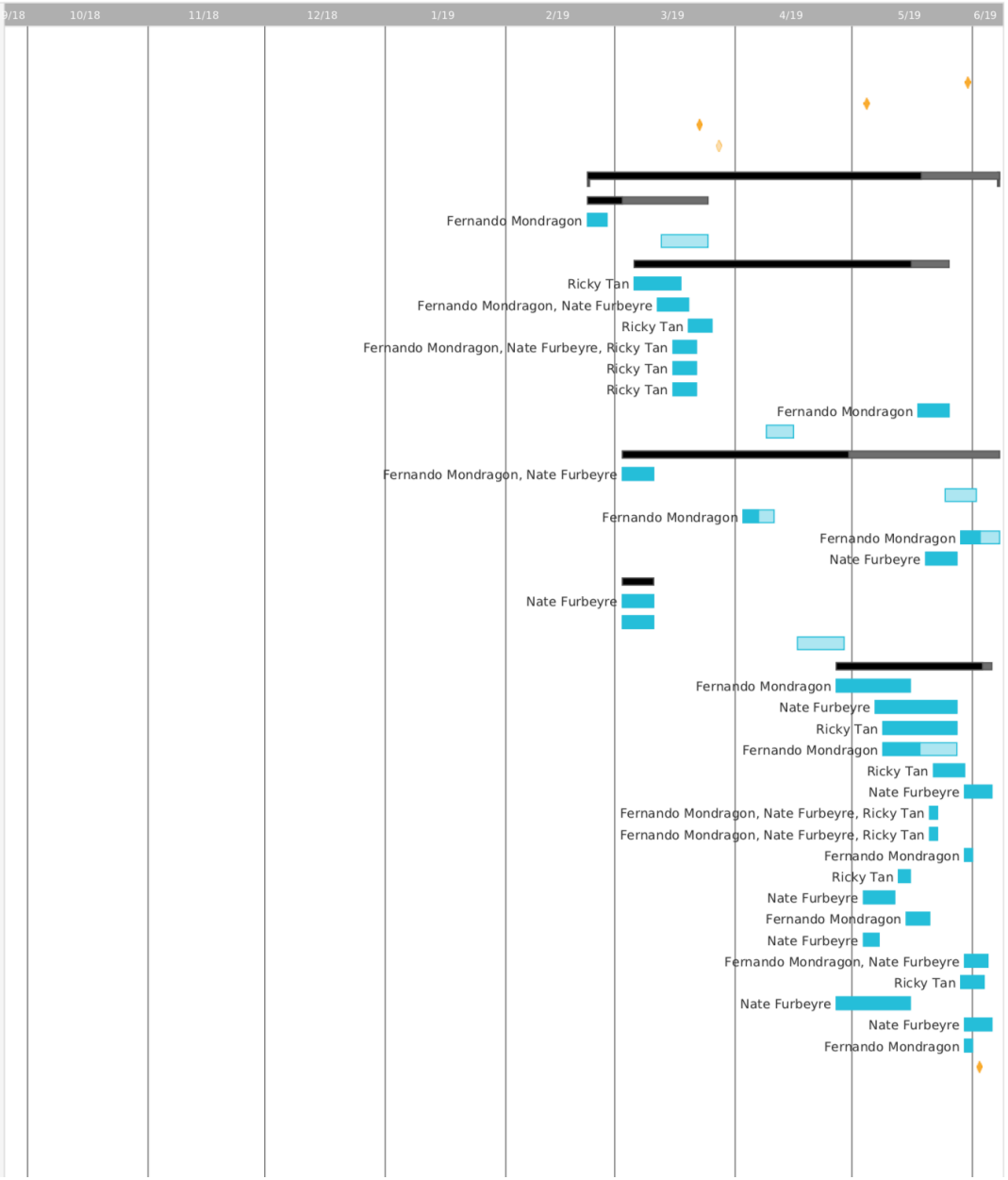
Appendix B: Gantt Chart

(Next page)


9/18
10/18
11/18
12/18
1/19
2/19
3/19
4/19
5/19
6/19
24 - Daimtronics 0h 94%
Problem Definition 0h 100%
Choose Project 0 100%
Meet Team 0 100%
Perform Customer/Need Research 0h 100%
Interview Users 0 100%
Online search for customer complaints 0 100%
Interview Sponsor 0 100%
Perform Technical Research 0h 100%
Develop Search Criteria 0 100%
Perform Literature Search 0 100%
Perform Product Research 0h 100%
Online search for current products 0 100%
Ask sponsor/users about currrent products 0 100%
Patent search for similar products 0 100%
Write Problem Statement 0 100%
Write Customer Needs/Wants List 0 100%
Perform QFD 0 100%
Create Initial Project Plan 0 100%
Write Scope of Work 0 100%
Scope of Work Milestone 0 100%
Determine Applicability of Prior Firmware 0h 100%
Research Microlaren 0 100%
Research Progressive 0 100%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Part Selection 0h 100%
Create an Initial Task Diagram 0 100%
Select Target Maneuvers 0 100%
Develop Required Detections for Maneuvers 0 100%
Develop Sensor Ranges 0 100%
Create Range Diagram 0 100%
Evaluate Motors 0 100%
Evaluate Computing Platform 0 100%
Evaluate Middleware 0 100%
Ricky Tan
Nate Furbeyre, Ricky Tan
Fernando Mondragon, Nate Furbeyre
Fernando Mondragon
Fernando Mondragon, Ricky Tan
Fernando Mondragon, Nate Furbeyre
Nate Furbeyre, Ricky Tan
Preliminary Design Review 0h 100%
Update Intro and Abstract 0 100%
Update Background and Objectives 0 100%
Write in our Concept Design Process 0 100%
Update Project Management 0 100%
Update Conclusion 0 100%
Submit PDR to Sponsor 0 100%
Present PDR 0 100%
Ricky Tan
Fernando Mondragon
Nate Furbeyre
Ricky Tan
Fernando Mondragon
Ricky Tan
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Sensor Selection 0h 100%
Develop Range Categories 0 100%
Find Possible Sensors - Lidar 360 0 100%
Find Possible Sensors - Lidar 1D 0 100%
Find Possible Sensors - Ultrasonic 0 100%
Evaluate - Lidar 360 0 100%
Evaluate - Lidar 1D 0 100%
Evaluate - Ultrasonic 0 100%
Ricky Tan
Fernando Mondragon
Nate Furbeyre
Ricky Tan
Nate Furbeyre, Ricky Tan
Fernando Mondragon, Ricky Tan
Fernando Mondragon, Nate Furbeyre


9/18
10/18
11/18
12/18
1/19
2/19
3/19
4/19
5/19
6/19
Create Sensor BOM 0 100%
Ricky Tan
Order Sensors 0 100%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Part Integration Setup 0h 100%
Create Electronics CAD 0 100%
Fernando Mondragon
Design Sensor Mounts 0 100%
Fernando Mondragon
Design Board Mounts 0 100%
Ricky Tan
Create High Level Wiring Diagram 0 100%
Nate Furbeyre
Order Pi & Teensy 0 100%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Send Sensor Purchase List 0 100%
Ricky Tan
Motherboard Development 0h 100%
Develop List of Modules 0 100%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Develop List of Components 0 100%
Ricky Tan
Create Schematic 0 100%
Ricky Tan
Create Board 0 100%
Ricky Tan
Order PCB 0 100%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Order Components 0 100%
Critical Design Review 0h 100%
Update Concept Design 0 100%
Fernando Mondragon
Add Final Design 0 100%
Nate Furbeyre
Add Manufacturing Plan 0 100%
Ricky Tan
Add Design Verification Plan 0 100%
Fernando Mondragon
Update all other sections 0 100%
Nate Furbeyre
Send to Sponsor 0 100%
Physical Development 0h 100%
RPLidar Mount V2 0 100%
Ricky Tan
Computing Mount Structure 0 100%
Ricky Tan
Battery 2 Mounting 0 100%
Ricky Tan
Software Development 0h 98%
Look into Sensor Drivers 0 100%
Fernando Mondragon
Look into Motor Drivers 0 100%
Nate Furbeyre
Create Initial Detailed Task Diagram 0 100%
Fernando Mondragon, Ricky Tan
Initial Design of Ros Nodes for Pi 0 100%
Nate Furbeyre
Research and Select Code Base for Teensy 0 100%
Nate Furbeyre
Start Writing Simulink-Middleware Link 0 100%
Nate Furbeyre
Figure out how to deploy in Simulink external m... 0 100%
Nate Furbeyre
Writing ChibiOS Code for Teensy 0h 100%
Read LIDAR data in ROS 0 100%
Nate Furbeyre
Debugging and Rewriting Code 0 90%
Fernando Mondragon, Nate Furbeyre, Ricky Tan
Manufacturing 0h 99%
Solder Motherboard 0 100%
Ricky Tan
Manufacture Sensor Mounts V1 0 100%
Ricky Tan
Manufacture Sensor Mounts V2 0 100%
Ricky Tan
Manufacture Computing Mount V1 0 100%
Ricky Tan
Battery Wires 0 100%
Ricky Tan
Solder Motherboard V2 0 100%
Ricky Tan
Manufacture ToF Mounts 0 100%
Ricky Tan
Manufacture Battery Mounts 0 100%
Ricky Tan
Manufacture Computing Mount V2 0 100%
Ricky Tan

Assemble Full Platform	0	100%	
Hardware/Safety Demo	0	100%	
Manufacturing & Test Review	0	100%	
All physical components finished	0	0%	
Testing	**0h**	**81%**	
Mounting Testing	**0h**	**29%**	
Mount Tests V1	0	100%	Fernando Mondragon
Computing Mount Ruggedness	0	0%	
Motherboard Testing	**0h**	**88%**	
Check for Power Shorts	0	100%	Ricky Tan
Communication Port Verification	0	100%	Fernando Mondragon, Nate Furbeyre
Main Power from Battery	0	100%	Ricky Tan
Powering Computing	0	100%	Fernando Mondragon, Nate Furbeyre, Ricky Tan
Noisy Supply Rails	0	100%	Ricky Tan
Quiet Supply Rails	0	100%	Ricky Tan
Relay Actuation	0	100%	Fernando Mondragon
ADC + Logic Shfiter	0	0%	
Sensor Testing	**0h**	**60%**	
IMU Calibration	0	100%	Fernando Mondragon, Nate Furbeyre
Wheel Speed Calibration	0	0%	
URF Calibration	0	50%	Fernando Mondragon
Lidar Lite Calibration	0	50%	Fernando Mondragon
RPLidar Calibration	0	100%	Nate Furbeyre
Software Testing	**0h**	**100%**	
Teensy Pi Basic Serial	0	100%	Nate Furbeyre
Teensy MicroUSB Upload	0	100%	
Setup Tests	0	0%	
Perform tests to verify Specs	**0h**	**94%**	
Verify Dead Man Switch	0	100%	Fernando Mondragon
Verify Motor Automatic Mode	0	100%	Nate Furbeyre
Verify Servo Automatic Mode	0	100%	Ricky Tan
Verify 5th Wheel Lock/Unlock	0	50%	Fernando Mondragon
Verify Budget	0	100%	Ricky Tan
Verify Control Loop Frequency	0	100%	Nate Furbeyre
Verify average acceleration time	0	100%	Fernando Mondragon, Nate Furbeyre, Ricky Tan
Verify average deceleration time	0	100%	Fernando Mondragon, Nate Furbeyre, Ricky Tan
Verify setup time	0	100%	Fernando Mondragon
Verify max steering angle	0	100%	Ricky Tan
Verify sensors read in Simulink	0	100%	Nate Furbeyre
Verify top speed	0	100%	Fernando Mondragon
Verify Max Wi-Fi Range	0	100%	Nate Furbeyre
Verify object detection range	0	100%	Fernando Mondragon, Nate Furbeyre
Verify sufficient battery life	0	100%	Ricky Tan
Verify sensors read from application layer	0	100%	Nate Furbeyre
Verify sufficient documentation rating	0	100%	Nate Furbeyre
Verify replacement time	0	100%	Fernando Mondragon
Project Expo	0	100%	



Appendix C: Computing Platform Decision Matrix

Computing Platform		Options											
		R-Pi (3B+) + Teensy		R-Pi (3b+) Only		Nvidia Jetson		Arduino Uno		Odroid + Uno		Custom Platform	
Criteria	**Weighting**	**Score**	***Total***	**Score**	***Total***	**Score**	**Tot*al***	**Score**	***Total***	**Score**	***Total***	**Score**	***Total***
Processing Power	*5*	4	20	2	10	5	25	1	5	4	20	3	15
Time to Implement	*4*	5	20	4	16	4	16	3	12	2	8	1	4
Num. Ports/Pins	*4*	5	20	2	8	3	12	2	8	4	16	4	16
Cost	*3*	5	15	5	15	1	3	4	12	3	9	3	9
Replacement Time	*3*	3	9	4	12	4	12	4	12	3	9	1	3
Documentation	*2*	3	6	4	8	4	8	4	8	3	6	1	2
Power Draw	*1*	3	3	4	4	2	2	4	4	3	3	3	3
Totals:			**93**		73		78		61		71		52

Appendix D: Motor Decision Matrix

Motor		Options							
		Tekin		Maxon		Turnigy		Multistar	
Criteria	**Weighting**	**Score**	***Total***	**Score**	***Total***	**Score**	***Total***	**Score**	***Total***
Cost	*5*	5	25	1	5	2	10	2	10
Top Speed	*3*	5	15	5	15	4	12	3	9
Time to Implement	*3*	5	15	3	9	1	3	1	3
Torque Output	*3*	4	12	4	12	5	15	5	15
Documentation	*2*	4	8	5	10	1	2	1	2
Efficiency	*2*	4	8	3	6	4	8	2	4
Weight	*1*	3	3	4	4	3	3	3	3
Totals:			**86**		61		53		46

Appendix E: Middleware Decision Matrix

Middleware		**Options**							
		ROS		**AUTOSAR**		**YARP**		**Custom**	
Criteria	**Weighting**	**Score**	***Total***	**Score**	***Total***	**Score**	***Total***	**Score**	***Total***
Time to Implement	*5*	5	25	4	20	4	20	2	10
Simulink Compatible	*4*	4	16	3	12	3	12	2	8
Flexibility	*4*	3	12	3	12	3	12	4	16
Documentation	*3*	5	15	4	12	3	9	4	12
Performance	*3*	4	12	5	15	4	12	2	6
Computer Vision	*2*	4	8	1	2	3	6	2	4
Totals:			**88**		73		71		56

Appendix F: Sensor Selection Process

Detection Requirements				
Maneuver	**Object**	**Range (m)**	**Lower Angle (deg)**	**Upper Angle (deg)**
Forward Object Avoidance	Vehicles ahead	14.00	0	6
	Road debris	14.00	0	6
	Road blockages	14.00	0	6
	Medium-sized objects	14.00	0	6
	Objects falling out of vehicles	14.00	0	6
Side Object Avoidance	Vehicles to either side	0.50	30	150
Lane Following	Lane markers	5.00	0	10
	Curbs	0.50	60	120
	Road transition	0.50	60	120
	Freeway rails	0.50	60	120
	Walls	0.50	60	120
Lane Changing	Lane markers	1.00	0	10
	Curbs	0.50	60	120
	Road transition	0.50	30	150
	Freeway rails	0.50	30	150
	Walls	0.50	30	150
	Vehicles to either side	0.50	30	150
	Vehicles ahead	2.00	0	30
Adaptive Cruise Control	Vehicles ahead	14.00	0	10
	Lane markers	5.00	0	10
J-Turn	Vehicles from oncoming lane	14.00	0	6
	Curbs	0.75	60	120
	Lane Markers	5.00	0	10
Parking	Stationary objects	0.50	150	180
	Space between vehicles	0.50	150	180
	Space between walls	0.50	150	180
	Parking lot markers	0.50	150	180
	Loading dock ledges	0.50	150	180
Trailer Pickup	Trailer	0.50	150	180
	Stationary objects	0.50	150	180
	Space between walls	0.50	150	180

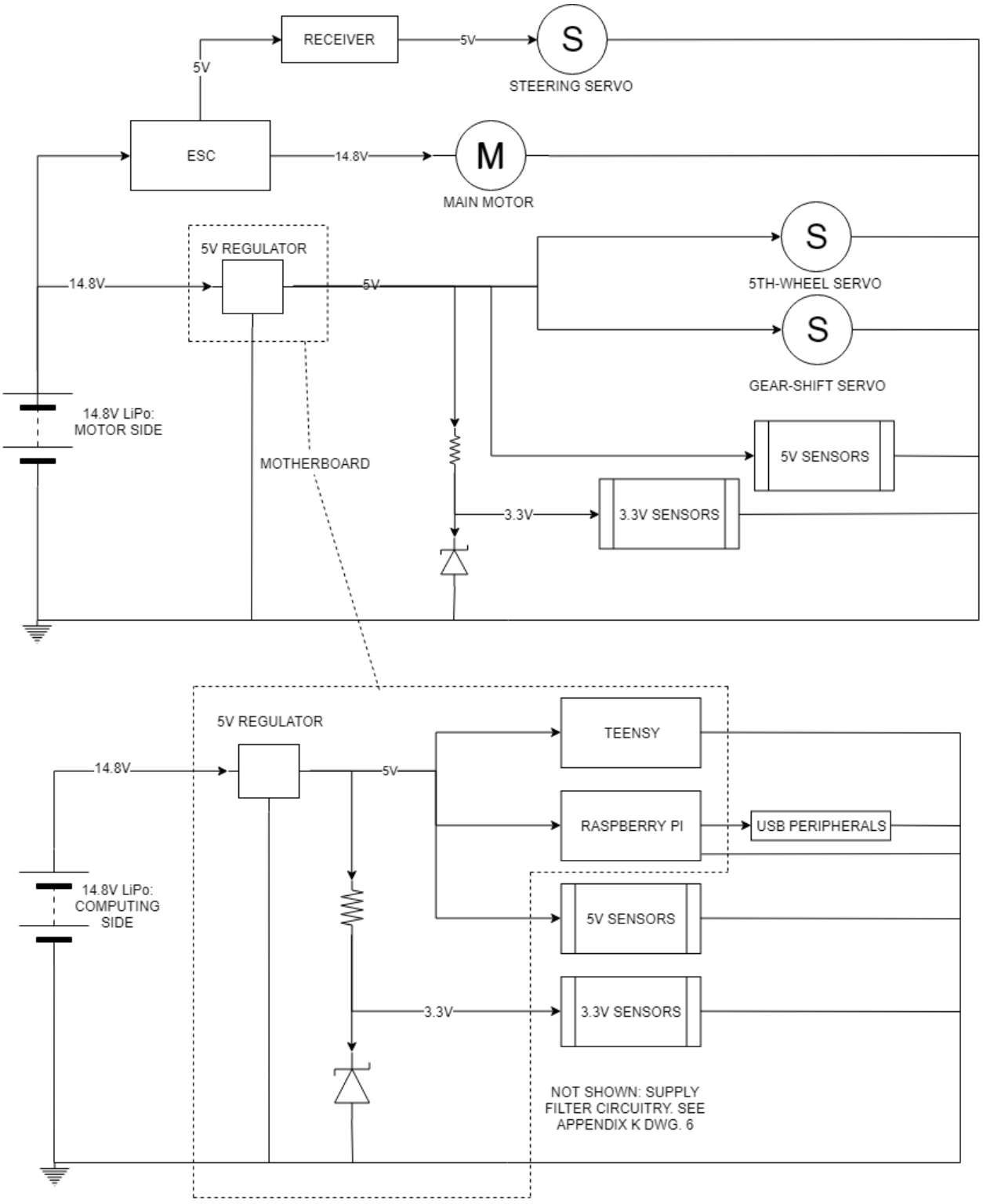
Appendix G: Design Hazard Checklist

DESIGN HAZARDS CHECKLIST

Y	N	
✔		1. Will the system include hazardous revolving, running, rolling, or mixing actions? [The truck]
	✔	2. Will the system include hazardous reciprocating, shearing, punching, pressing, squeezing, drawing, or cutting actions?
✔		3. Will any part of the design undergo high accelerations/decelerations? [The truck]
✔		4. Will the system have any large (>5 kg) moving masses or large (>250 N) forces? [The truck]
	✔	5. Could the system produce a projectile?
	✔	6. Could the system fall (due to gravity), creating injury?
	✔	7. Will a user be exposed to overhanging weights as part of the design?
	✔	8. Will the system have any burrs, sharp edges, shear points, or pinch points?
	✔	9. Will any part of the electrical systems not be grounded?
	✔	10. Will there be any large batteries (over 30 V)?
	✔	11. Will there be any exposed electrical connections in the system (over 40 V)?
	✔	12. Will there be any stored energy in the system such as flywheels, hanging weights or pressurized fluids/gases?
✔		13. Will there be any explosive or flammable liquids, gases, or small particle fuel as part of the system? [LiPo batteries]
	✔	14. Will the user be required to exert any abnormal effort or experience any abnormal physical posture during the use of the design?
✔		15. Will there be any materials known to be hazardous to humans involved in either the design or its manufacturing? [Leaded solder]
	✔	16. Could the system generate high levels (>90 dBA) of noise?
	✔	17. Will the device/system be exposed to extreme environmental conditions such as fog, humidity, or cold/high temperatures, during normal use?
✔		18. Is it possible for the system to be used in an unsafe manner?
✔		19. For powered systems, is there an emergency stop button? [Dead-man switch on RC + disabled default state]
	✔	20. Will there be any other potential hazards not listed above? If yes, please explain on reverse.

Description of Hazard	Planned Corrective Action	Planned Date	Actual Date
The roughly 40 lb semi-truck will be able to move at speeds up to 20 mph (Applies to Checklist #1, #3 and #4)	The user manual will warn the user that the system may behave unpredictably while reprogramming the vehicle (e.g. rapid acceleration and deceleration).	12/14/2018	
A user can control this vehicle in an unsafe manner (Applies to Checklist #18).	The user manual will warn the user to not drive the vehicle recklessly, and to be cautious of nearby people and objects.	12/14/2018	

Appendix H: Overall Power Routing



Appendix I: Bill of Materials

(Next page)

Category	Subassembly	Part	Description	ernal Par	Supplier	Site Product Code	Unit Cost	Qty.	Total Cost	rdered?
Computing	Teensy	Teensy 3.6	Teensy 3.6	C001	N/A	DEV-14057	$ 29.25	1	$ 29.25	Y
	Raspberry Pi	Raspberry Pi 3B+	Raspberry Pi 3B+	C002	N/A	3775	$ 35.00	1	$ 35.00	Y
Electrical	Batteries	LiPo Battery	True Spec Premium 14.8V 3700	E001	SMC Racing	4585-4S1P	$ 43.00	2	$ 86.00	Y
		Motor Power Connectors	XT90 Male Female Connectors	E002	Hobby King	9015000199-0	$ 3.99	1	$ 3.99	Y
	Connectors	RC Connectors	10cm Female to Female Servo L	E003	Hobby King	15000058	$ 3.00	1	$ 3.00	Y
		RC Connectors	20cm Female to Female Servo L	E004	Hobby King	015000170-0	$ 5.29	1	$ 5.29	Y
		Motor Splice Connector	12-10 Ga. Butt Splice 3-Way	E005	Ace Hardware	3482361	$ 3.87	1	$ 3.87	Y
	Fans	Small Fan	5V Centrifugal Fan	E006	Mouser	108-BFB0505HHA-CR00	$ 13.46	1	$ 13.46	Y
	Wires	Jumper Wires	M-M, M-F, F-F 0.1" Jumper Wires	E007	Amazon	B07QDL1R3Y	$ 7.99	1	$ 7.99	Y
Motherboard	Motherboard	Motherboard	Blank Printed Circuit Board	MB001	PCBWay	N/A	$ 50.00	1	$ 50.00	Y
	12V Regulator	12V Voltage Regulator	TPS54308DDCR	MB002	Digikey	296-47880-1-ND	$ 1.63	1	$ 1.63	Y
		12V Input Capacitor	CL32B106KAJNNNE	MB003	Digikey	1276-1807-1-ND	$ 0.47	2	$ 0.94	Y
		12V Input Capacitor X	CC0805KRX7R8BB104	MB004	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
		12V Boost Capacitor	CC0805KRX7R8BB104	MB005	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
		12V Inductor	7447798181	MB006	Digikey	732-2417-1-ND	$ 3.34	1	$ 3.34	Y
		12V Top Feedback Resistor	ERJ-P06F1003V	MB007	Digikey	P16060CT-ND	$ 0.20	1	$ 0.20	Y
		12V Bottom Feedback Resis	ERJ-PB6D5231V	MB008	Digikey	P21101CT-ND	$ 0.25	1	$ 0.25	Y
		12V Output Capacitor	CL32B226KAJNFNE	MB009	Digikey	1276-3391-1-ND	$ 0.98	2	$ 1.96	Y
	3.3V Regulators	3.3V Voltage Regulator	AZ1117EH-3.3TRG1	MB010	Digikey	AZ1117EH-3.3TRG1DICT-I	$ 0.44	2	$ 0.88	Y
		3.3V Capacitors	CL32B106KAJNNNE	MB011	Digikey	1276-1807-1-ND	$ 0.47	2	$ 0.94	Y
		3.3V Capacitor Bypass	CC0805KRX7R8BB104	MB012	Digikey	311-1141-1-ND	$ 0.16	2	$ 0.32	Y
		3.3V Capacitors Sm.	GMK212BBJ106KG-T	MB013	Digikey	587-4893-1-ND	$ 0.89	2	$ 1.78	Y
	Analog Filtering	Analog Dampening Capacit	CL32B106KAJNNNE	MB014	Digikey	1276-1807-1-ND	$ 0.47	1	$ 0.47	Y
		Analog Bypass Capacitor	CC0805KRX7R8BB104	MB015	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
	Pi Headers	Pi Female GPIO Port	NPPC202KFMS-RC	MB016	Digikey	S5729-ND	$ 3.92	1	$ 3.92	Y
		Pi Male GPIO Port	SBH11-NBPC-D20-SM-BK	MB017	Digikey	S9132-ND	$ 1.68	1	$ 1.68	Y
	Motor Data Port	Motor Data Pullup Resistors	CRCW08055K00FKTA	MB018	Digikey	541-4321-1-ND	$ 0.16	3	$ 0.48	Y
	5th-Wheel Current Sens	Current Sense IC	INA250A4PW	MB019	Digikey	296-45003-5-ND	$ 3.00	1	$ 3.00	Y
		Current Sense Dampening C	CL32B106KAJNNNE	MB020	Digikey	1276-1807-1-ND	$ 0.47	1	$ 0.47	Y
		Current Sense Bypass Capa	CC0805KRX7R8BB104	MB021	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
	I2C Superbus	I2C Accelerator	LTC4311CSC6#TRMPBF	MB022	Digikey	LTC4311CSC6#TRMPBFC	$ 3.99	1	$ 3.99	Y
		I2C Pullups	CRGH0805F10K	MB023	Digikey	A126417CT-ND	$ 0.11	3	$ 0.33	Y
		I2C Acc. Bypass Cap	CC0805KRX7R8BB104	MB024	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
		I2C Multiplexer IC	TCA9548APWR	MB025	Digikey	296-34905-1-ND	$ 1.55	1	$ 1.55	Y
		I2C Mult. Damp Cap	CL32B106KAJNNNE	MB026	Digikey	1276-1807-1-ND	$ 0.47	1	$ 0.47	Y
		I2C Mult. Bypass Cap	CC0805KRX7R8BB104	MB027	Digikey	311-1141-1-ND	$ 0.16	1	$ 0.16	Y
		I2C Pullups	CRGH0805F10K	MB028	Digikey	A126417CT-ND	$ 0.11	16	$ 1.76	Y
	Relay	DPDT EMR	TXS2SA-L-3V	MB029	Digikey	255-3945-5-ND	$ 5.23	1	$ 5.23	Y
	Buzzer	Buzzer	AUDIO MAGNETIC INDICATOR 2	MB030	Digikey	102-3740-ND	$ 2.52	1	$ 2.52	Y
	5V Regulators	5V Voltage Regulator	LM2678SX-5.0/NOPB	MB031	Digikey	LM2678SX-5.0/NOPBCT	$ 6.00	2	$ 12.00	Y
		5V Cb	GMK212BBJ106KG-T	MB032	Digikey	587-4893-1-ND	$ 0.89	2	$ 1.78	Y
		5V Cin	20SVPF120M	MB033	Digikey	P16498CT-ND	$ 1.36	2	$ 2.72	Y
		5V Cinx	CC0805KRX7R8BB104	MB034	Digikey	311-1141-1-ND	$ 0.16	2	$ 0.32	Y
		5V Cout	8TPE100MAZB	MB035	Digikey	P16641CT-ND	$ 1.55	6	$ 9.30	Y
		5V Diode	STPS30M60	MB036	Digikey	497-12322-1-ND	$ 3.64	2	$ 7.28	Y
		5V Inductor	XAL8080-103MEB	MB037	Mouser	994-XAL8080-103MEB	$ 5.83	2	$ 11.66	Y
	Battery Terminals	Terminal	039380-0102	MB038	Digikey	WM13944-ND	$ 1.44	2	$ 2.88	Y
	5V Analog	5V ADC	MCP3008T	MB039	Digikey	MCP3008T-I/SLCT-ND	$ 2.26	1	$ 2.26	Y
		Logic shifter	TXB0104DB	MB040	Digikey	296-21928-1-ND	$ 1.32	1	$ 1.32	Y
		Shifter R	ERJ-P06J102V	MB041	Digikey	P1.0KADCT-ND	$ 0.13	1	$ 0.13	Y
		Shifter C	CC0805KRX7R8BB104	MB042	Digikey	311-1141-1-ND	$ 0.16	2	$ 0.32	Y
	FBC Filtering	Ferrite Bead	ILHB1206ER121V	MB043	Digikey	541-2304-1-ND	$ 0.23	1	$ 0.23	Y
		FBC Cbyp	CC0603JRX7R9BB104	MB044	Digikey	311-1779-1-ND	$ 0.14	1	$ 0.14	Y
		FBC Cdamp	GMK212BBJ106KG-T	MB045	Digikey	587-4893-1-ND	$ 0.89	1	$ 0.89	Y
Mechanical	Filament	Damping Components	TPU Filament Spool	ME001	Amazon	AMU2951752-10	$ 27.00	1	$ 27.00	Y
		Mounting Components	PETG Filament Spool	ME002	Amazon	3D PETG-1KG1.75-WHT	$ 20.00	2	$ 40.00	Y
	Lidar Mounting	Lidar Bumper	1-D Lidar and Front Bumper Mou	ME003	Ricky	N/A	$ -	1	$ -	Y
		RPLidar Mount	360 RPLidar Mount	ME004	Ricky	N/A	$ -	1	$ -	Y
	ToF Mounting	ToF Front Mount	Time-of-Flight Front Mount	ME005	Ricky	N/A	$ -	4	$ -	Y
		ToF Back Mount	Time-of-Flight Back Mount	ME006	Ricky	N/A	$ -	4	$ -	Y
		ToF Bracket	Time-of-Flight Bracket	ME007	Ricky	N/A	$ -	4	$ -	Y
	Computing Mounting	Case Column Left	Computing Mount Case Column	ME008	Ricky	N/A	$ -	1	$ -	Y
		Case Column Right	Computing Mount Case Column	ME009	Ricky	N/A	$ -	1	$ -	Y
		Pi Spring	Computing Mount Pi Spring	ME010	Ricky	N/A	$ -	4	$ -	Y
		Motherboard Spring	Computing Mount Motherboard S	ME011	Ricky	N/A	$ -	2	$ -	Y
		Soft Washers	Computing Mount Soft Washers	ME012	Ricky	N/A	$ -	6	$ -	Y
		M2.5 Short Bolt	18-8 Stainless Steel Socket Hea	ME013	McMaster	91292A012	$ 5.15	1	$ 5.15	Y
		M2.5 Long Bolt	18-8 Stainless Steel Socket Hea	ME014	McMaster	91292A019	$ 12.26	1	$ 12.26	Y
		M2.5 Hex Nut	18-8 Stainless Steel Thin Hex Nu	ME015	McMaster	90710A025	$ 2.35	1	$ 2.35	Y
		M2.5 Washer	18-8 Stainless Steel Washer M2.	ME016	McMaster	93475A196	$ 1.58	1	$ 1.58	Y
		Case Walls	Computing Mount Case Walls	ME017	Ricky	N/A	$ -	1	$ -	Y
		Case Floor	Computing Mount Case Floor	ME018	Ricky	N/A	$ -	1	$ -	Y
		Case Gate	Computing Mount Case Gate	ME019	Ricky	N/A	$ -	1	$ -	Y
		Case Lid	Computing Mount Case Lid	ME020	Ricky	N/A	$ -	1	$ -	Y
		Pi-Motherboard Standoffs	Pi-Motherboard Standoffs	ME021	Ricky	N/A	$ -	4	$ -	Y
	Battery Mounting	Battery Housing	Battery Housing	ME022	Ricky	N/A	$ -	1	$ -	Y
		Battery Feet Front	Battery Feet Front	ME023	Ricky	N/A	$ -	3	$ -	Y
		Battery Feet Back	Battery Feet Back	ME024	Ricky	N/A	$ -	2	$ -	Y
		Battery Fingers	Battery Fingers	ME025	Ricky	N/A	$ -	4	$ -	Y
	5th-Wheel Mounting	Servo Clasp	5th-Wheel Servo Clasp	ME026	Ricky	N/A	$ -	2	$ -	Y
Sensors	Sensors	360 Lidar	RPLidar A2M8 360 Laser Scann	S001	RobotShop	RB-Rpk-02	$ 300.00	1	$ 300.00	Y
		1D Lidar	LIDAR-Lite 3 Laser Rangefinder	S002	RobotShop	RB-Pli-17	$ 150.00	1	$ 150.00	Y
		Ultrasonic Rangefinder	HC-SR04 Ultra01+	S003	RobotShop	RB-Elf-143	$ 3.50	8	$ 28.00	Y
		Digital Wheel Speed	SparkFun Line Sensor Breakout	S004	Sparkfun	ROB-09454	$ 2.95	4	$ 11.80	Y
		Higher Quality URF	Maxbotix Ultrasonic Rangefinder	S005	Adafruit	980	$ 24.95	1	$ 24.95	Y
		Time of Flight	Adafruit VL53L0X	S006	Adafruit	3317	$ 14.95	1	$ 14.95	Y

Appendx J: DVPR

(next page)

Senior Project DVP&R

Date: 2/7/2019	Team: Daimtronics	Sponsor: Dr. Birdsong and Daimler Trucks NA	Description of System: Autonomous Model Semi-Truck Platform	DVP&R Engineer:

TEST PLAN / TEST REPORT

Item No	Specification #	Test Description	Acceptance Criteria	Test Responsibil	Test Stage	SAMPLES		TIMING		TEST RESULTS			NOTES
						Quantity	Type	Start date	Finish date	Test Result	Quantity Pass	Quantity Fail	
1	--	Verify that Teensy and Pi can be powered by motherboard	Microcontrollers turn on and stay on	Fernando Mondragon	FP	3	Sub	February 22th	March 1st				
2	--	Verify that mounts hold all of the sensors and microcontrollers	Pass	Ricky Tan	FP	1	Sub	February 22th	March 1st				
3	--	Verify that all ports on the motherboard are connected to the correct pins of microcontrollers	Pass	Fernando Mondragon	FP	1	Sub	February 22th	March 1st				
4	--	Verify that the Teensy code can be uploaded through the micro-USB	Pass	Nate Furbeyre	FP	1	C	March 1st	March 8th				
5	--	Verify that messages can be sent from the Teensy to the Pi to be displayed on a monitor	Pass	Ricky Tan	FP	3	Sub	March 1st	March 8th				
6	--	Verify wheel speed sensors are calibrated	Sensors read up to the 20mph top speed	Fernando Mondragon	FP	3	C	March 8th	March 15th				
7	--	Verify Ultrasonic Range Finders are Calibrated	URFs detect objects within 0.5 m range	Nate Furbeyre	FP	3	C	March 8th	March 15th				
8	--	Verify Lite Lidar is Calibrated	Lite Lidar detects objects within 40m range	Ricky Tan	FP	3	C	March 8th	March 15th				
9	--	Verify RPLidar outputs point cloud data	RPLidar point cloud data can be viewed in SLAM	Fernando Mondragon	FP	3	C	March 8th	March 15th				
10	--	Verify that the IMU is calibrated correctly	The IMU is calibrated and can read semi-truck heading data.	Nate Furbeyre	FP	3	C	March 8th	March 15th				
11	--	Verify that the dead man switch works properly	Releasing switch on the RC controller stops the vehicle	Nate Furbeyre	FP	3	Sub	March 15th	March 22nd				
12	--	Verify the motor can be controlled in an automatic mode	Semi-truck can acccelerate without manually controlled input from controller to receiver	Nate Furbeyre	FP	1	C	March 15th	March 22nd				
13	--	Verify the steering servo can be controlled in an automatic mode	Semi-truck can steer without manually controlled input from controller to receiver	Ricky Tan	FP	1	C	March 15th	March 22nd				
14	--	Verify the 5th wheel can be locked and unlocked	The servo controlling the 5th wheel can toggle between 2 positions	Fernando Mondragon	FP	1	C	March 15th	March 22nd				
15	1	Verify parts are within budget	project < $3000	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
16	2	Verify control loop frequency	System runs smoothly	Ricky Tan	FP	5	Sys	April 15th	April 30th				
17	3	Verify average acceleration time	Top speed in < 4sec	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
18	4	Verify average deceleration time	Complete stop < 4sec	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
19	5	Verify setup time	New autonomous mode uploaded in < 4 sec	Ricky Tan	FP	5	Sys	April 15th	April 30th				
20	6	Verify max steering angle	10% deviation from similitude value	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
21	7	Verify sensors read in Simulink	Simulink can read data from each sensor	Ricky Tan	FP	3	Sys	April 15th	April 30th				
22	8	Verify top speed	System reaches 20mph	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
23	9	Verify Max Wi-Fi Range	Minimum of 10m	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
24	10	Verify object detection range	Object detection for > 10m range	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
25	11	Verify object detection angle	Object detection for > 180° angle	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
26	12	Verify sufficient battery life	System can run for > 30min	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
27	13	Verify sensors read from application layer	There is a library function to read each sensor	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
28	14	Verify sufficient documentation rating	Survey for documentation usability > 8/10	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
29	15	Verify voltage safety rating	Factor of safety for > 2	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
30	16	Verify replacement time	Replacing part takes < 1 hr	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				

Senior Project DVP&R

Date: 2/7/2019	Team: Daimtronics	Sponsor: Dr. Birdsong and Daimler Trucks NA	Description of System: Autonomous Model Semi-Truck Platform	DVP&R Engineer:

TEST PLAN

TEST REPORT

Item No	Specification #	Test Description	Acceptance Criteria	Test Responsibil	Test Stage	SAMPLES		TIMING		TEST RESULTS			NOTES
						Quantity	Type	Start date	Finish date	Test Result	Quantity Pass	Quantity Fail	
1	--	Verify that Teensy and Pi can be powered by motherboard	Microcontrollers turn on and stay on	Fernando Mondragon	FP	3	Sub	February 22th	March 1st				
2	--	Verify that mounts hold all of the sensors and microcontrollers	Pass	Ricky Tan	FP	1	Sub	February 22th	March 1st				
3	--	Verify that all ports on the motherboard are connected to the correct pins of microcontrollers	Pass	Fernando Mondragon	FP	1	Sub	February 22th	March 1st				
4	--	Verify that the Teensy code can be uploaded through the micro-USB	Pass	Nate Furbeyre	FP	1	C	March 1st	March 8th				
5	--	Verify that messages can be sent from the Teensy to the Pi to be displayed on a monitor	Pass	Ricky Tan	FP	3	Sub	March 1st	March 8th				
6	--	Verify wheel speed sensors are calibrated	Sensors read up to the 20mph top speed	Fernando Mondragon	FP	3	C	March 8th	March 15th				
7	--	Verify Ultrasonic Range Finders are Calibrated	URFs detect objects within 0.5 m range	Nate Furbeyre	FP	3	C	March 8th	March 15th				
8	--	Verify Lite Lidar is Calibrated	Lite Lidar detects objects within 40m range	Ricky Tan	FP	3	C	March 8th	March 15th				
9	--	Verify RPLidar outputs point cloud data	RPLidar point cloud data can be viewed in SLAM	Fernando Mondragon	FP	3	C	March 8th	March 15th				
10	--	Verify that the IMU is calibrated correctly	The IMU is calibrated and can read semi-truck heading data.	Nate Furbeyre	FP	3	C	March 8th	March 15th				
11	--	Verify that the dead man switch works properly	Releasing switch on the RC controller stops the vehicle	Nate Furbeyre	FP	3	Sub	March 15th	March 22nd				
12	--	Verify the motor can be controlled in an automatic mode	Semi-truck can acccelerate without manually controlled input from controller to receiver	Nate Furbeyre	FP	1	C	March 15th	March 22nd				
13	--	Verify the steering servo can be controlled in an automatic mode	Semi-truck can steer without manually controlled input from controller to receiver	Ricky Tan	FP	1	C	March 15th	March 22nd				
14	--	Verify the 5th wheel can be locked and unlocked	The servo controlling the 5th wheel can toggle between 2 positions	Fernando Mondragon	FP	1	C	March 15th	March 22nd				
15	1	Verify parts are within budget	project < $3000	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
16	2	Verify control loop frequency	System runs smoothly	Ricky Tan	FP	5	Sys	April 15th	April 30th				
17	3	Verify average acceleration time	Top speed in < 4sec	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
18	4	Verify average deceleration time	Complete stop < 4sec	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
19	5	Verify setup time	New autonomous mode uploaded in < 4 sec	Ricky Tan	FP	5	Sys	April 15th	April 30th				
20	6	Verify max steering angle	10% deviation from similitude value	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
21	7	Verify sensors read in Simulink	Simulink can read data from each sensor	Ricky Tan	FP	3	Sys	April 15th	April 30th				
22	8	Verify top speed	System reaches 20mph	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
23	9	Verify Max Wi-Fi Range	Minimum of 10m	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
24	10	Verify object detection range	Object detection for > 10m range	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
25	11	Verify object detection angle	Object detection for > 180° angle	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
26	12	Verify sufficient battery life	System can run for > 30min	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
27	13	Verify sensors read from application layer	There is a library function to read each sensor	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
28	14	Verify sufficient documentation rating	Survey for documentation usability > 8/10	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
29	15	Verify voltage safety rating	Factor of safety for > 2	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
30	16	Verify replacement time	Replacing part takes < 1 hr	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				

Senior Project DVP&R

Date: 2/7/2019	Team: Daimtronics	Sponsor: Dr. Birdsong and Daimler Trucks NA	Description of System: Autonomous Model Semi-Truck Platform	DVP&R Engineer:

TEST PLAN

TEST REPORT

Item No	Specification #	Test Description	Acceptance Criteria	Test Responsibil	Test Stage	SAMPLES		TIMING		TEST RESULTS			NOTES
						Quantity	Type	Start date	Finish date	Test Result	Quantity Pass	Quantity Fail	
1	--	Verify that Teensy and Pi can be powered by motherboard	Microcontrollers turn on and stay on	Fernando Mondragon	FP	3	Sub	February 22th	March 1st				
2	--	Verify that mounts hold all of the sensors and microcontrollers	Pass	Ricky Tan	FP	1	Sub	February 22th	March 1st				
3	--	Verify that all ports on the motherboard are connected to the correct pins of microcontrollers	Pass	Fernando Mondragon	FP	1	Sub	February 22th	March 1st				
4	--	Verify that the Teensy code can be uploaded through the micro-USB	Pass	Nate Furbeyre	FP	1	C	March 1st	March 8th				
5	--	Verify that messages can be sent from the Teensy to the Pi to be displayed on a monitor	Pass	Ricky Tan	FP	3	Sub	March 1st	March 8th				
6	--	Verify wheel speed sensors are calibrated	Sensors read up to the 20mph top speed	Fernando Mondragon	FP	3	C	March 8th	March 15th				
7	--	Verify Ultrasonic Range Finders are Calibrated	URFs detect objects within 0.5 m range	Nate Furbeyre	FP	3	C	March 8th	March 15th				
8	--	Verify Lite Lidar is Calibrated	Lite Lidar detects objects within 40m range	Ricky Tan	FP	3	C	March 8th	March 15th				
9	--	Verify RPLidar outputs point cloud data	RPLidar point cloud data can be viewed in SLAM	Fernando Mondragon	FP	3	C	March 8th	March 15th				
10	--	Verify that the IMU is calibrated correctly	The IMU is calibrated and can read semi-truck heading data.	Nate Furbeyre	FP	3	C	March 8th	March 15th				
11	--	Verify that the dead man switch works properly	Releasing switch on the RC controller stops the vehicle	Nate Furbeyre	FP	3	Sub	March 15th	March 22nd				
12	--	Verify the motor can be controlled in an automatic mode	Semi-truck can acccelerate without manually controlled input from controller to receiver	Nate Furbeyre	FP	1	C	March 15th	March 22nd				
13	--	Verify the steering servo can be controlled in an automatic mode	Semi-truck can steer without manually controlled input from controller to receiver	Ricky Tan	FP	1	C	March 15th	March 22nd				
14	--	Verify the 5th wheel can be locked and unlocked	The servo controlling the 5th wheel can toggle between 2 positions	Fernando Mondragon	FP	1	C	March 15th	March 22nd				
15	1	Verify parts are within budget	project < $3000	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
16	2	Verify control loop frequency	System runs smoothly	Ricky Tan	FP	5	Sys	April 15th	April 30th				
17	3	Verify average acceleration time	Top speed in < 4sec	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
18	4	Verify average deceleration time	Complete stop < 4sec	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
19	5	Verify setup time	New autonomous mode uploaded in < 4 sec	Ricky Tan	FP	5	Sys	April 15th	April 30th				
20	6	Verify max steering angle	10% deviation from similitude value	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
21	7	Verify sensors read in Simulink	Simulink can read data from each sensor	Ricky Tan	FP	3	Sys	April 15th	April 30th				
22	8	Verify top speed	System reaches 20mph	Fernando Mondragon	FP	5	Sys	April 15th	April 30th				
23	9	Verify Max Wi-Fi Range	Minimum of 10m	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
24	10	Verify object detection range	Object detection for > 10m range	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
25	11	Verify object detection angle	Object detection for > 180° angle	Nate Furbeyre	FP	5	Sys	April 15th	April 30th				
26	12	Verify sufficient battery life	System can run for > 30min	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
27	13	Verify sensors read from application layer	There is a library function to read each sensor	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
28	14	Verify sufficient documentation rating	Survey for documentation usability > 8/10	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				
29	15	Verify voltage safety rating	Factor of safety for > 2	Nate Furbeyre	FP	1	Sys	April 15th	April 30th				
30	16	Verify replacement time	Replacing part takes < 1 hr	Nate Furbeyre	FP	3	Sys	April 15th	April 30th				

Appendix K: Drawing Package

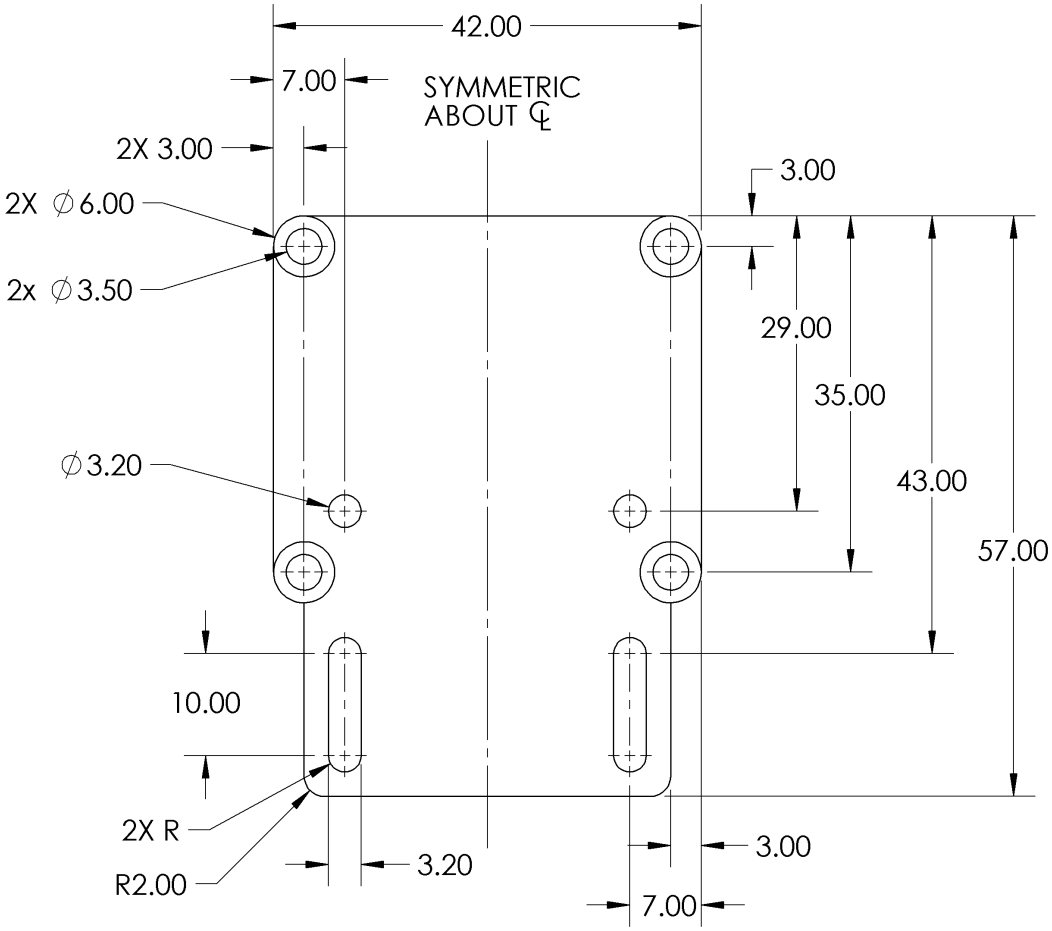
Attached in the next pages:

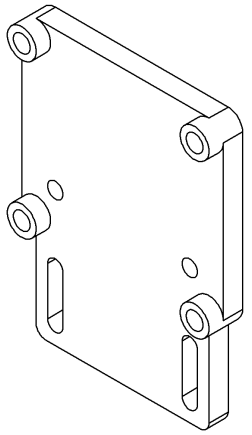
1. ME003 – LIDAR BUMPER
2. ME004 – RPLIDAR MOUNT
3. ME005 – TOF FRONT MOUNT
4. ME006 – TOF BACK MOUNT
5. ME007 – TOF BRACKET
6. ME008 – COMPUTING COLUMN LEFT
7. ME009 – COMPUTING COLUMN RIGHT
8. ME010 – PI SPRING
9. ME011 – MOTHERBOARD SPRING
10. ME012 – SOFT WASHER
11. ME017 – CASE WALLS
12. ME018 – CASE FLOOR
13. ME019 – CASE GATE
14. ME020 – CASE LID
15. ME021 – PI-MOTHERBOARD STANDOFFS
16. ME022 – BATTERY HOUSING
17. ME023 – BATTERY FEET FRONT

18. ME024 – BATTERY FEET BACK
19. ME025 –BATTERY FINGER
20. ME026 – SERVO CLASP
21. MB001 – MOTHERBOARD SCHEMATIC
    a. MB001-1: SUPPLY CIRCUITRY
    b. MB001-2: COMPUTING BREAKOUT
    c. MB001-3: SUPPORT CIRCUITRY

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: POLYETHYLENE TEREPHTHALATE GLYCOL (PETG)
4. COLOR: WHITE, GRAY, OR BLACK





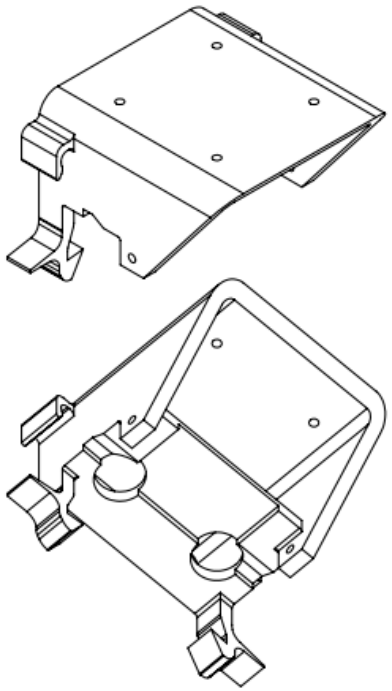
ISOMETRIC SCALE: 1:1

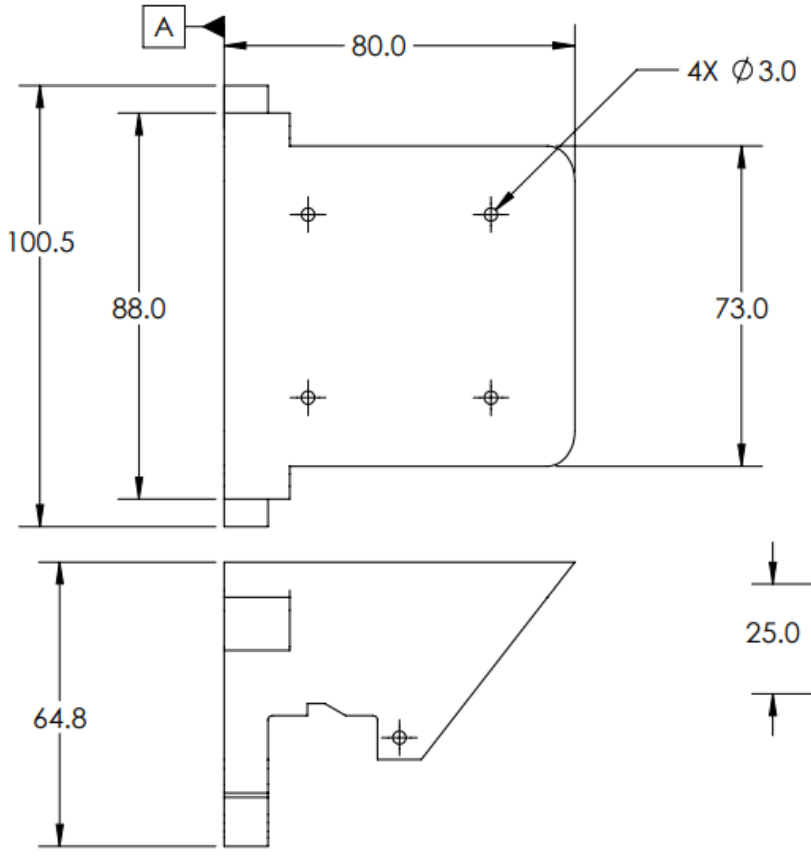
Cal Poly Mechanical Engineering	Lab Section: 02	Team: DAIMTRONICS	Title: LIDAR BUMPER		Drwn. By: R. TAN
ME 429 - Winter 2019	Dwg. #: ME003	Nxt Asb:	Date: 2-7-19	Scale: 3:2	Chkd. By: F. MONDRAGON

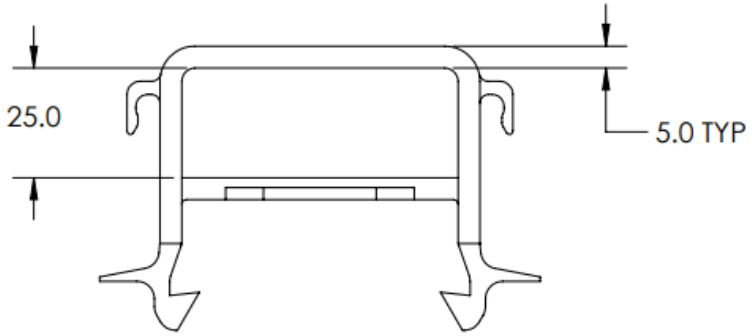
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

ISOMETRIC SCALE: 1:2





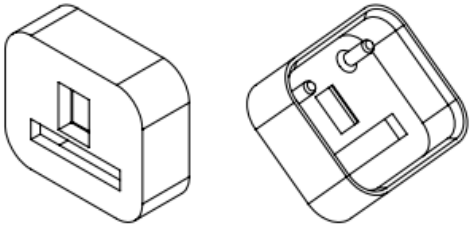


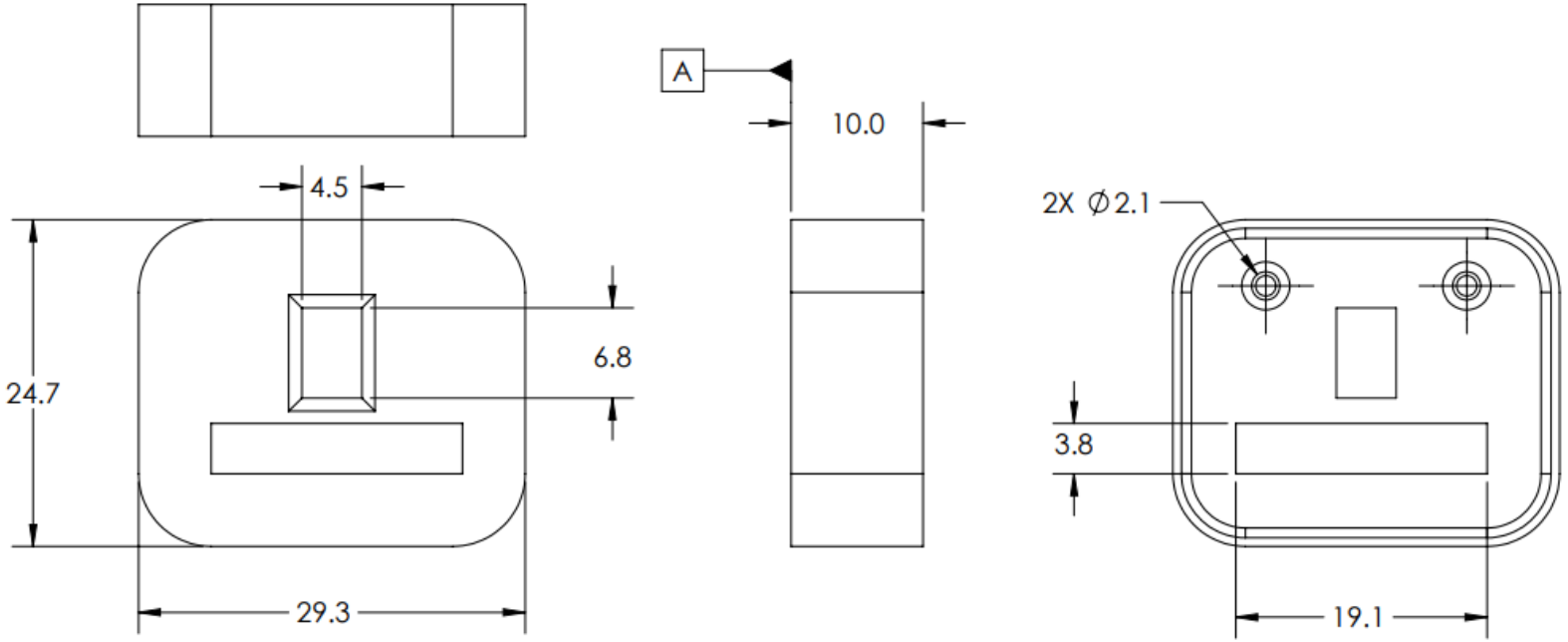
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: RPLIDAR MOUNT		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME004	Nxt Asb:	Date: 5-29-19	Scale: 2:3	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

ISOMETRIC SCALE: 1:1



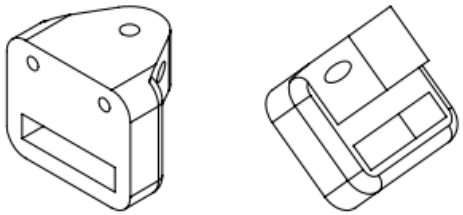


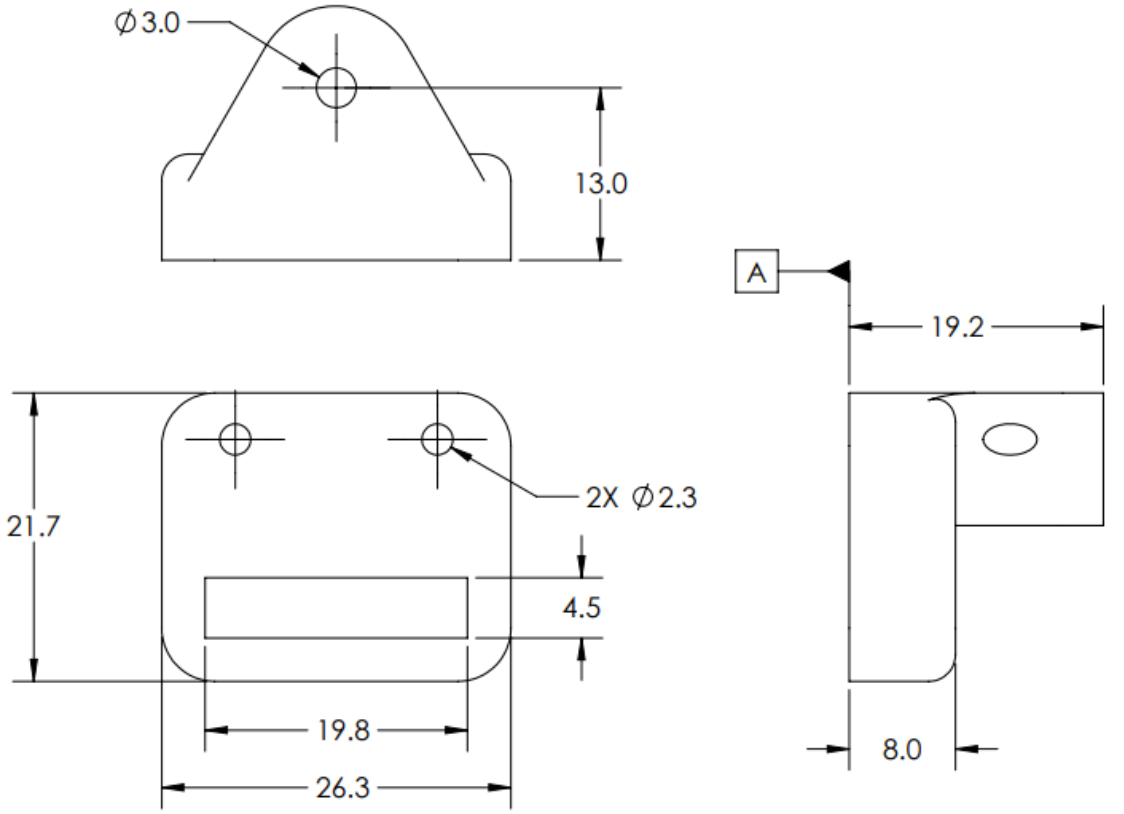
Cal Poly Mechanical Engineering ME 430 - Winter 2019	Lab Section: 01	Team: DAIMTRONICS	Title: TOF FRONT MOUNT		Drwn. By: R. TAN
	Dwg. #: ME005	Nxt Asb:	Date: 5-29-19	Scale: 2:1	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

ISOMETRIC SCALE: 1:1

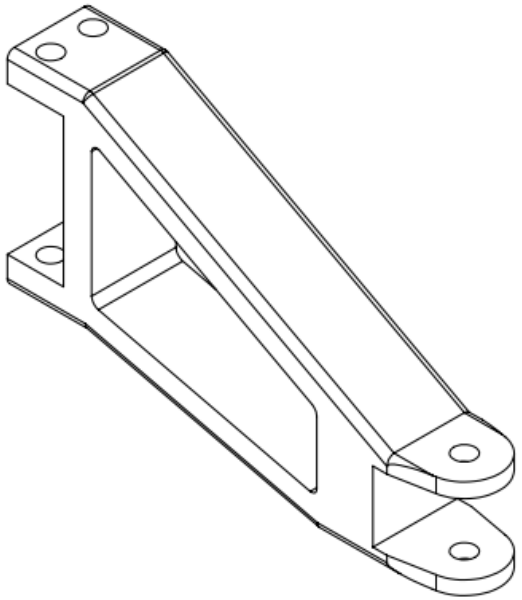


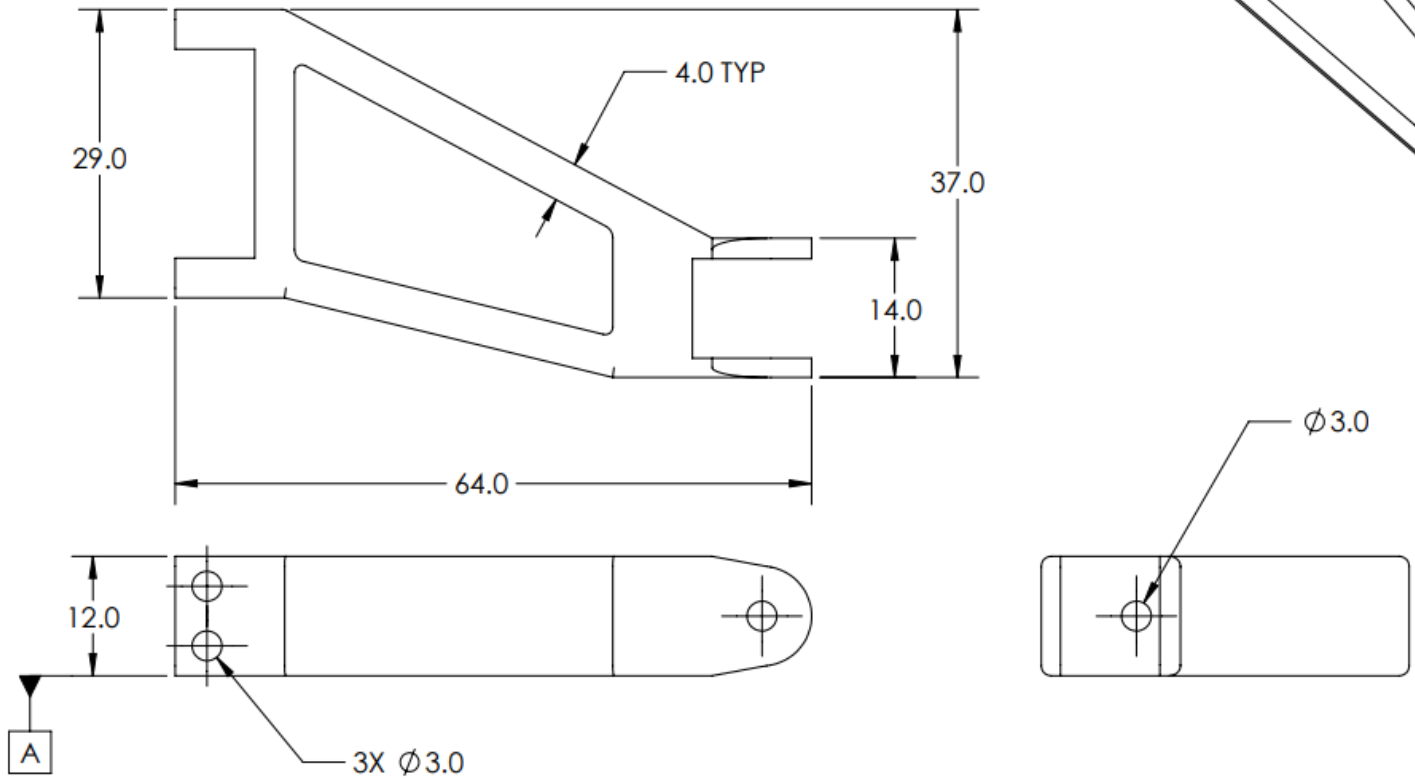


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: TOF BACK MOUNT		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME006	Nxt Asb:	Date: 5-29-19	Scale: 2:1	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

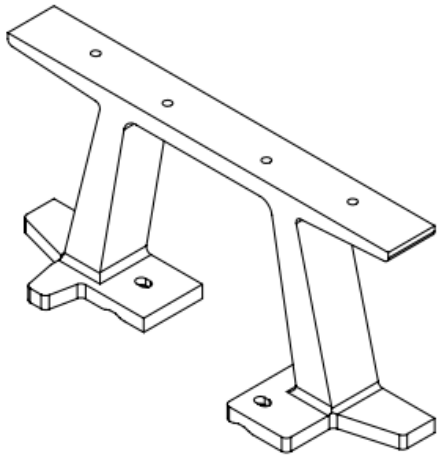


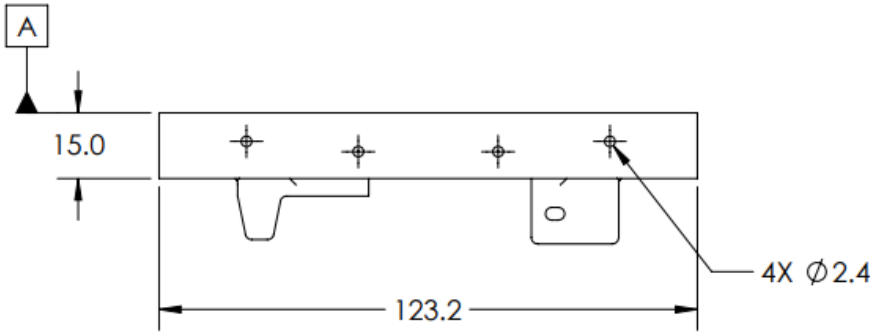


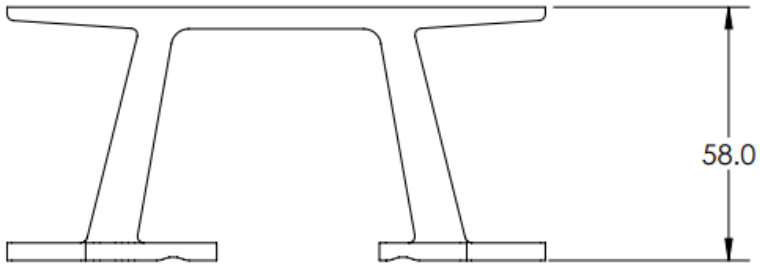
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: TOF BRACKET		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME007	Nxt Asb:	Date: 5-29-19	Scale: 3:2	Chkd. By:

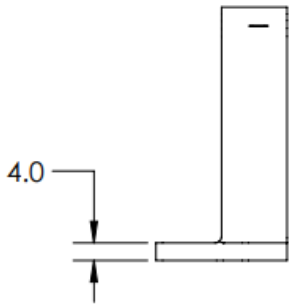
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING





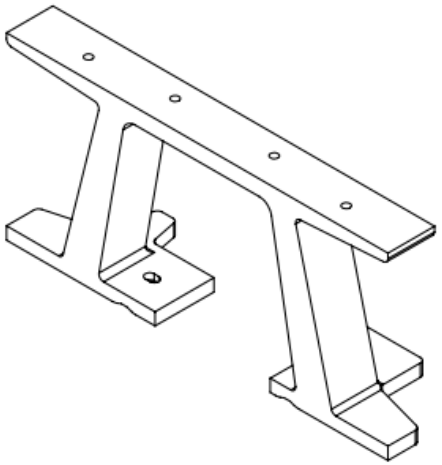


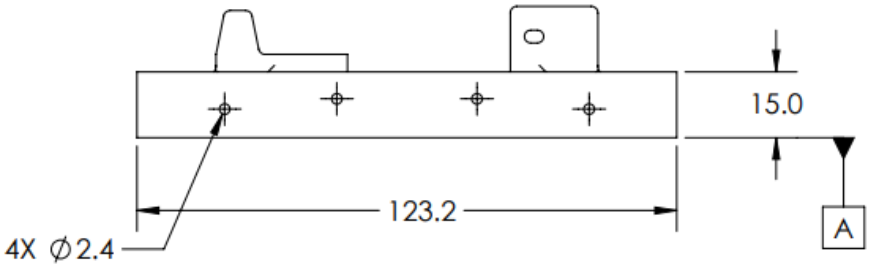


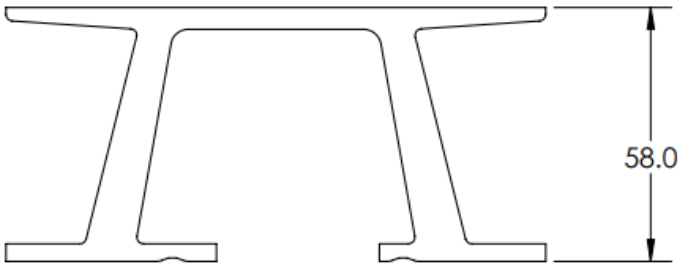
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: COMPUTING COLUMN LEFT		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME008	Nxt Asb:	Date: 5-29-19	Scale: 2:3	Chkd. By:

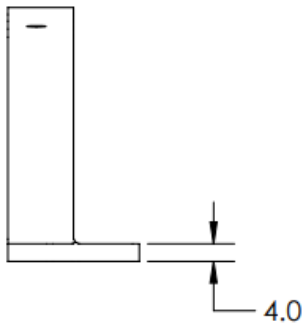
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING







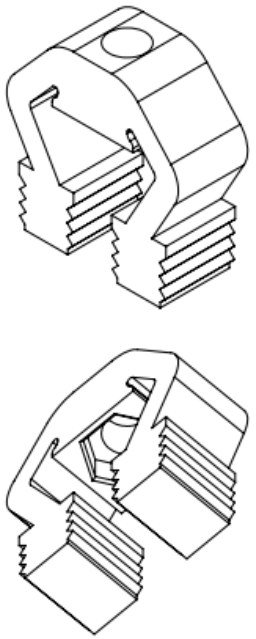


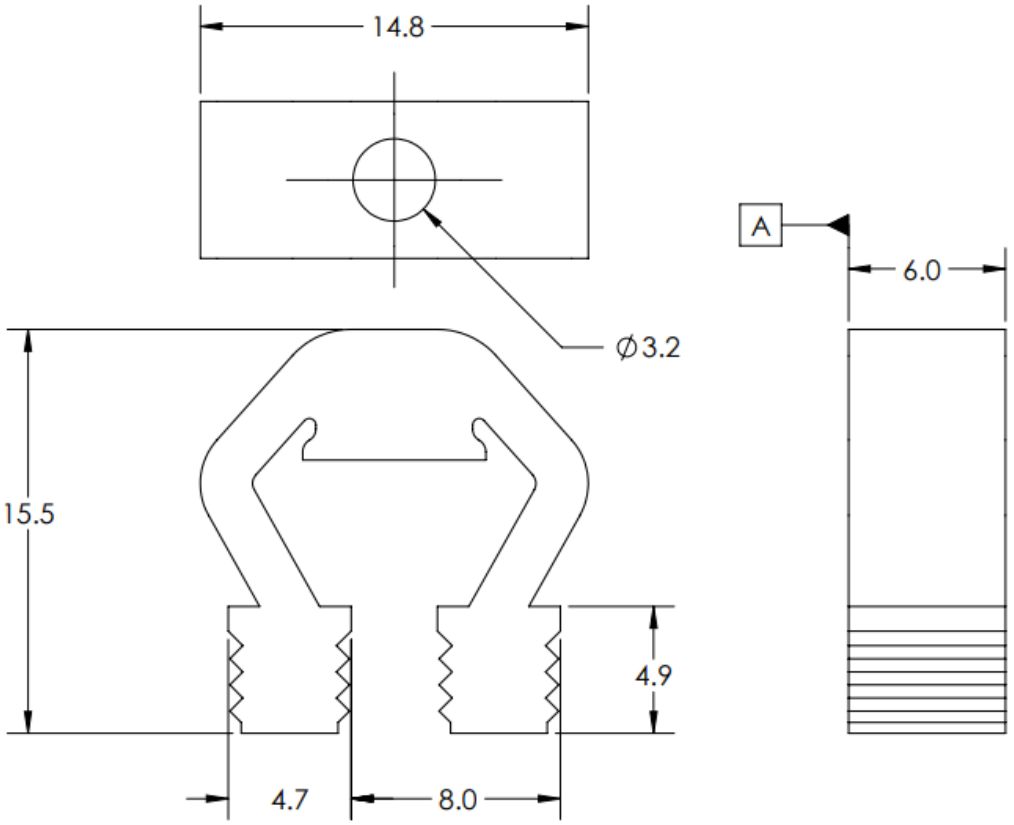
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: COMPUTING COLUMN RIGHT		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME009	Nxt Asb:	Date: 5-29-19	Scale: 2:3	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: TPU (THERMOPLASTIC POLYURETHANE)
4. COLOR: BLACK
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

ISOMETRIC SCALE: 5:2



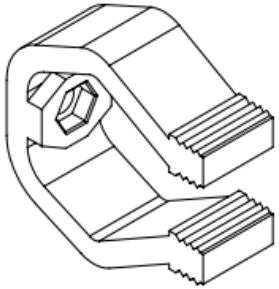


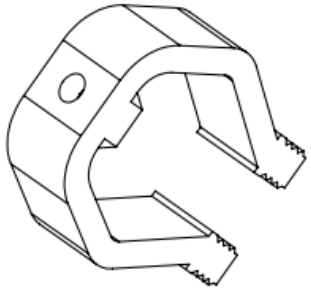
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: PI SPRING		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME010	Nxt Asb:	Date: 5-29-19	Scale: 4:1	Chkd. By:

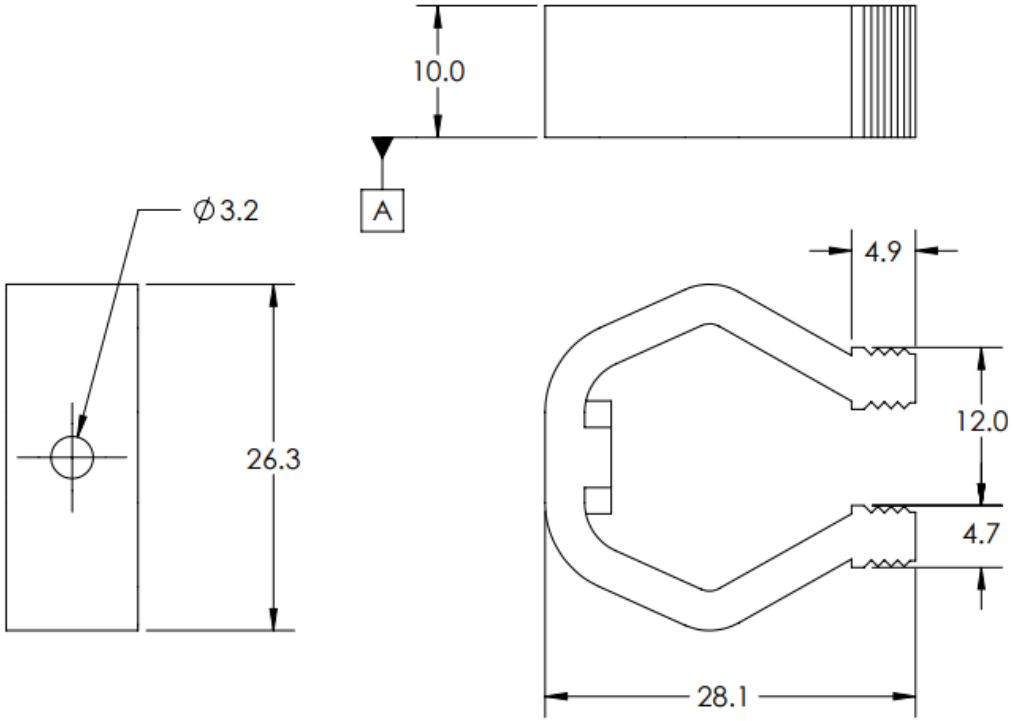
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: TPU (THERMOPLASTIC POLYURETHANE)
4. COLOR: BLACK
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

ISOMETRIC SCALE: 3:2



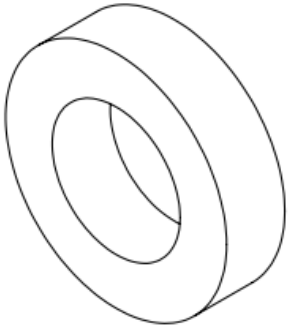


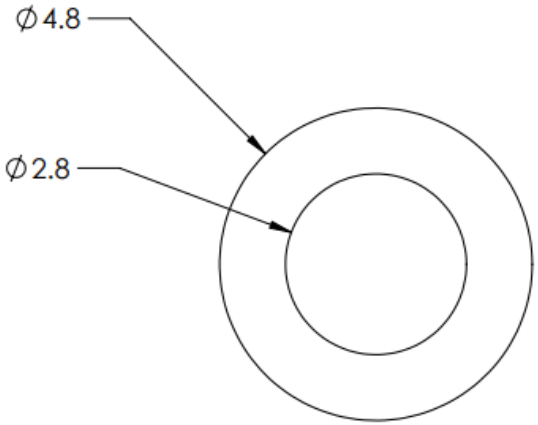


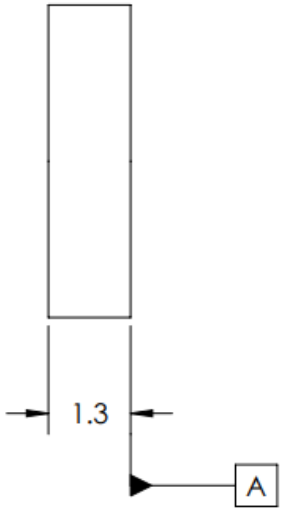
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: MOTHERBOARD SPRING		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME011	Nxt Asb:	Date: 5-29-19	Scale: 2:1	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: TPU (THERMOPLASTIC POLYURETHANE)
4. COLOR: BLACK
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING



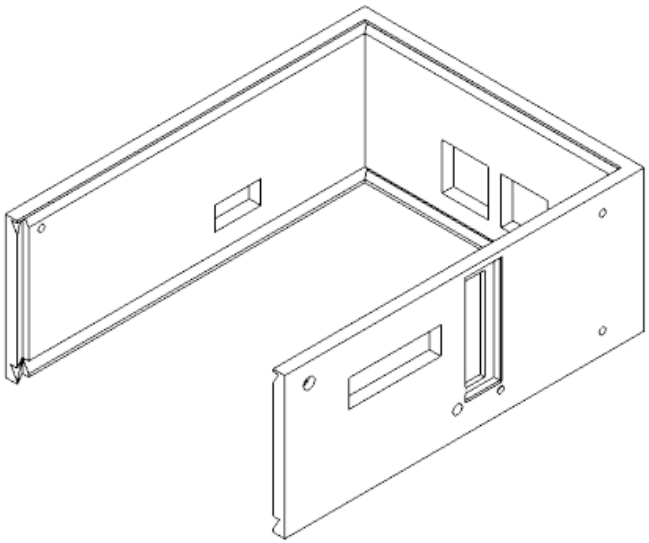


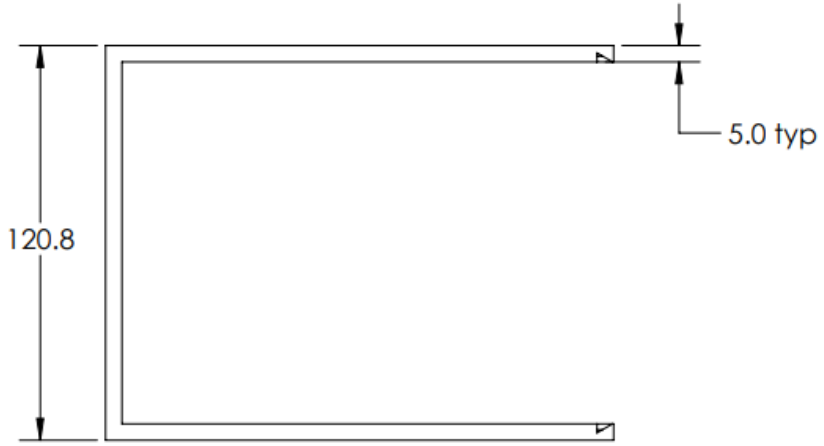


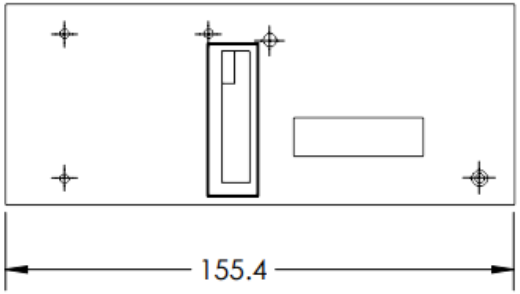
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: SOFT WASHER		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME012	Nxt Asb:	Date: 5-29-19	Scale: 10:1	Chkd. By:

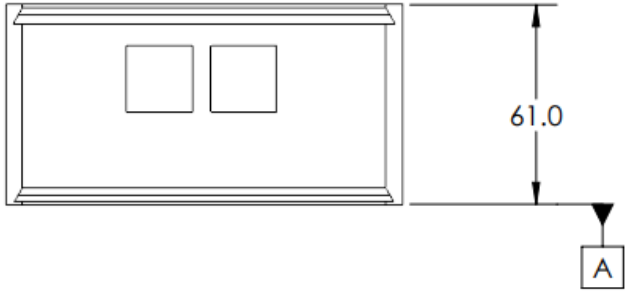
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING





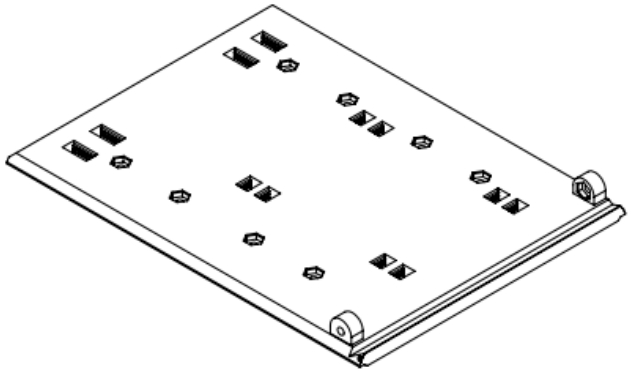


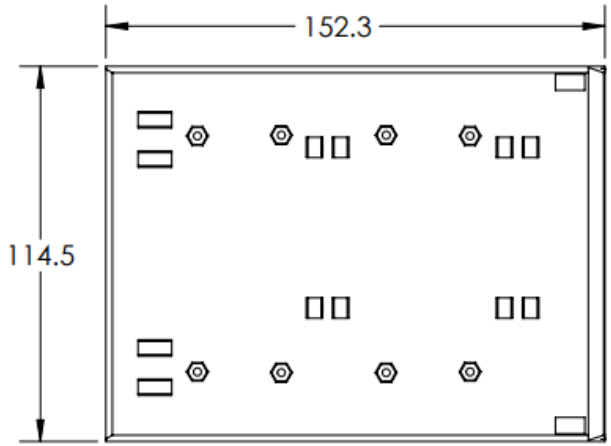


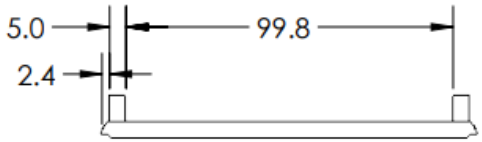
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: CASE WALLS		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME017	Nxt Asb:	Date: 5-29-19	Scale: 1:2	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING



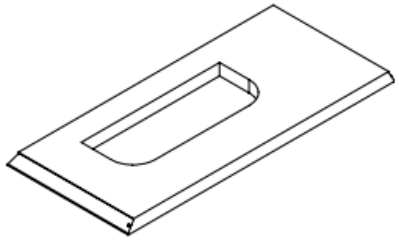


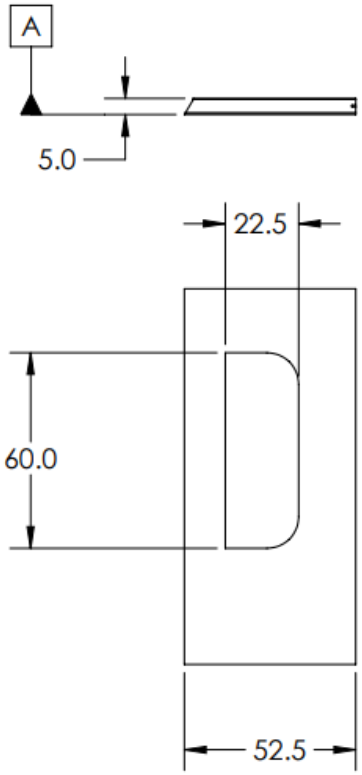


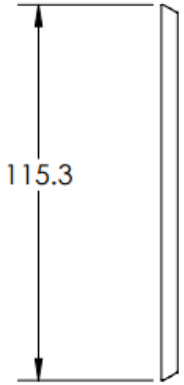
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: CASE FLOOR		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME018	Nxt Asb:	Date: 5-29-19	Scale: 1:2	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING



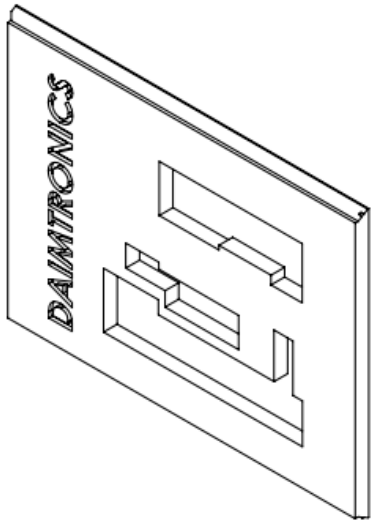


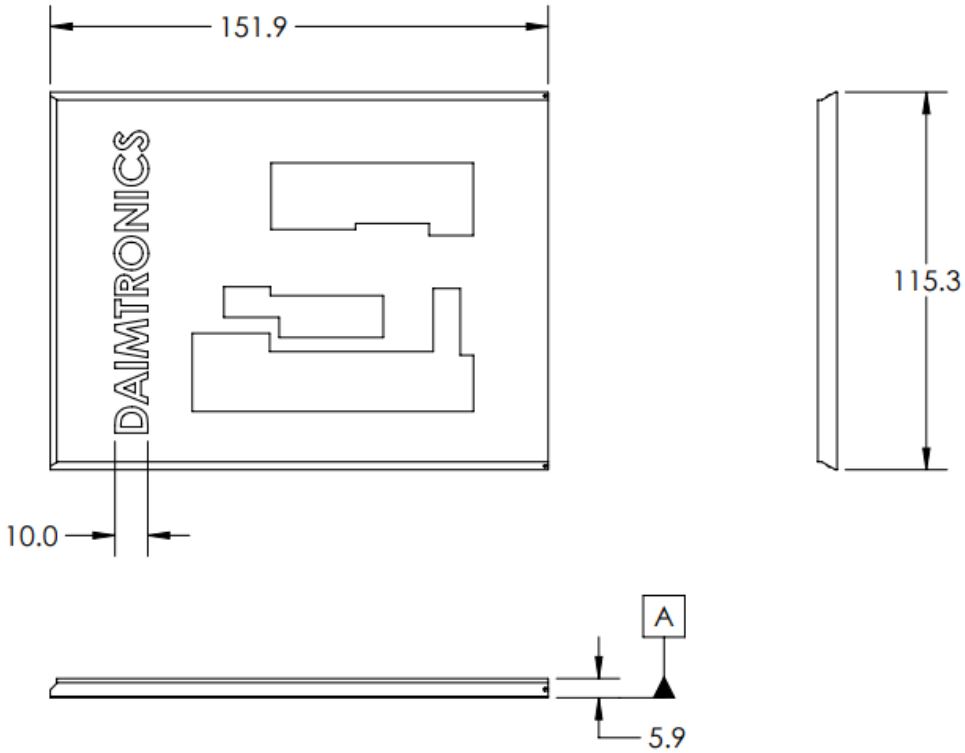


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: CASE GATE		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME019	Nxt Asb:	Date: 5-29-19	Scale: 1:2	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

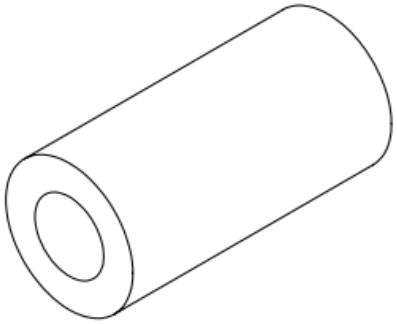


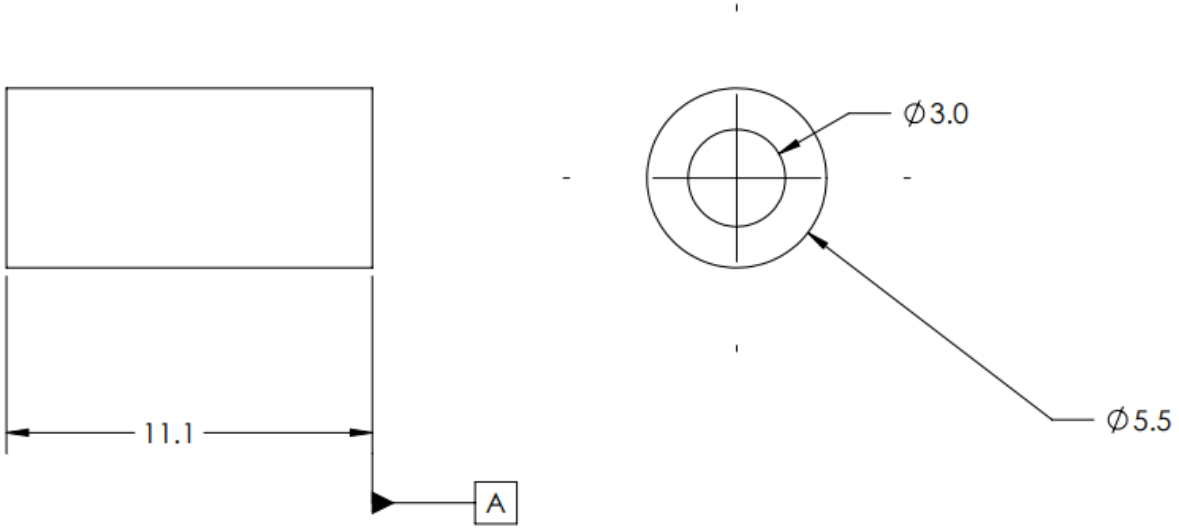


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: CASE LID		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME020	Nxt Asb:	Date: 5-29-19	Scale: 1:2	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

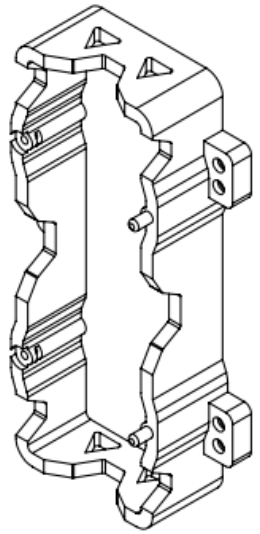


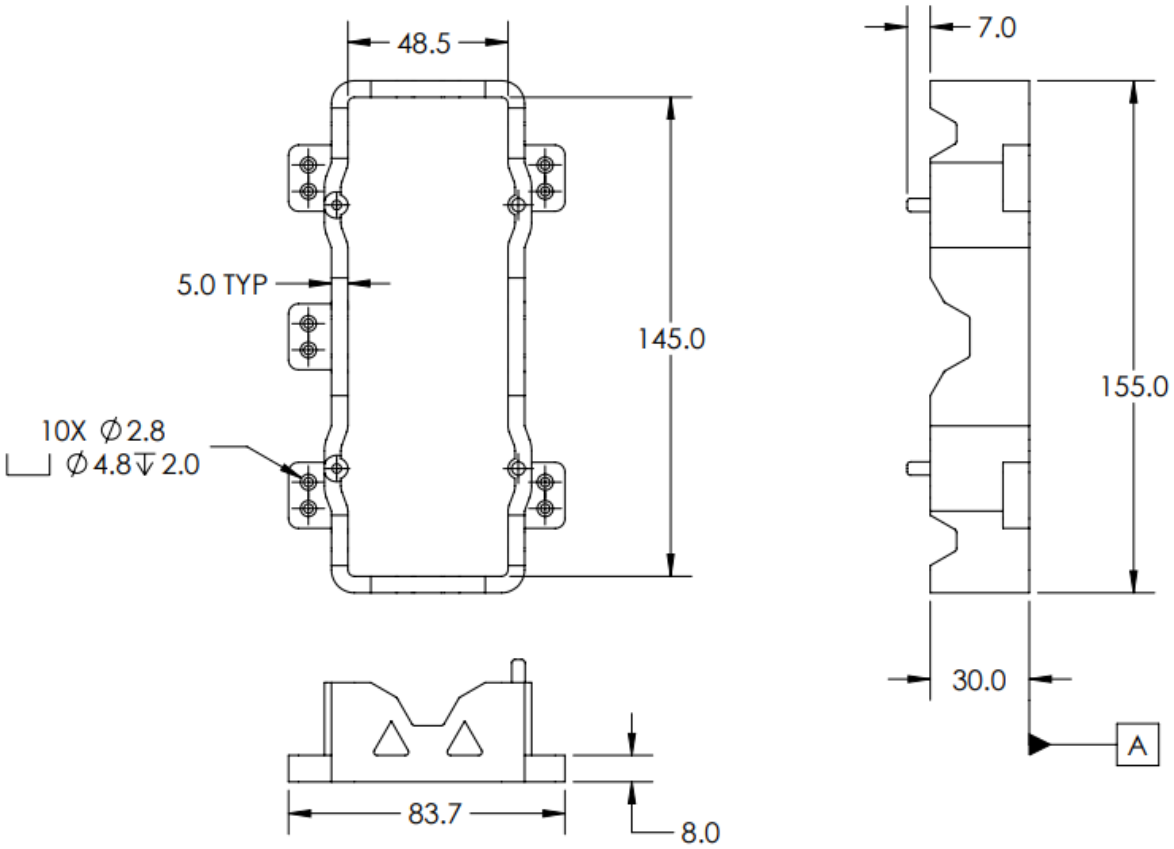


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: PI-MOTHERB. STANDOFF		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME021	Nxt Asb:	Date: 5-29-19	Scale: 5:1	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

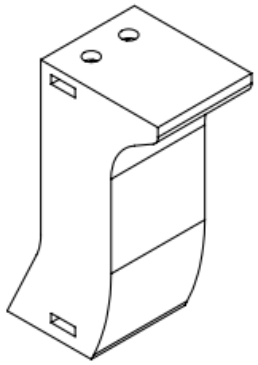


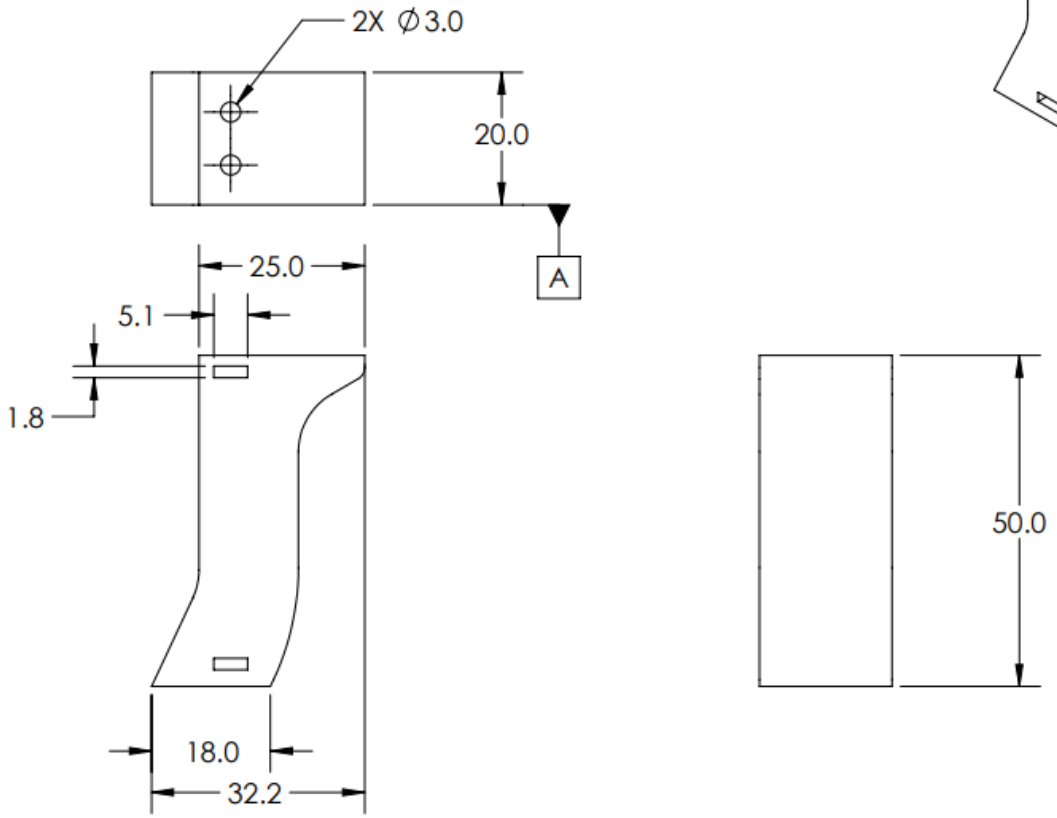


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: BATTERY HOUSING		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME022	Nxt Asb:	Date: 5-29-19	Scale: 1:2	Chkd. By:

NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

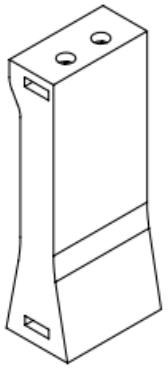


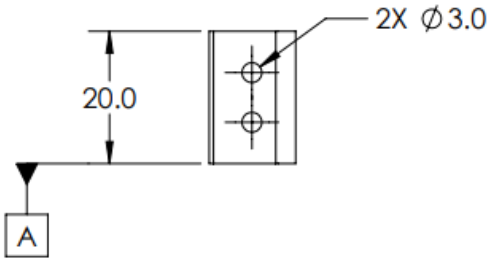


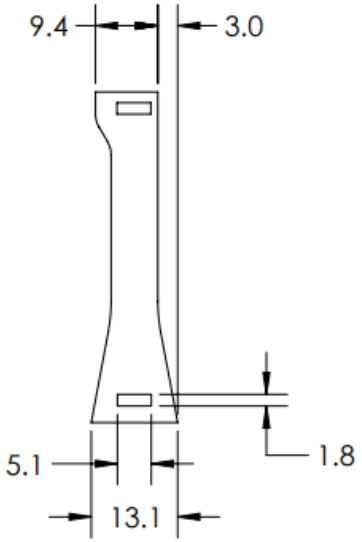
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: BATTERY FEET FRONT		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME023	Nxt Asb:	Date: 5-29-19	Scale: 1:1	Chkd. By:

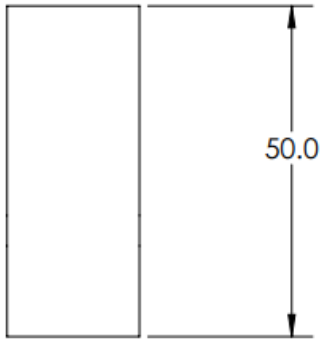
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING





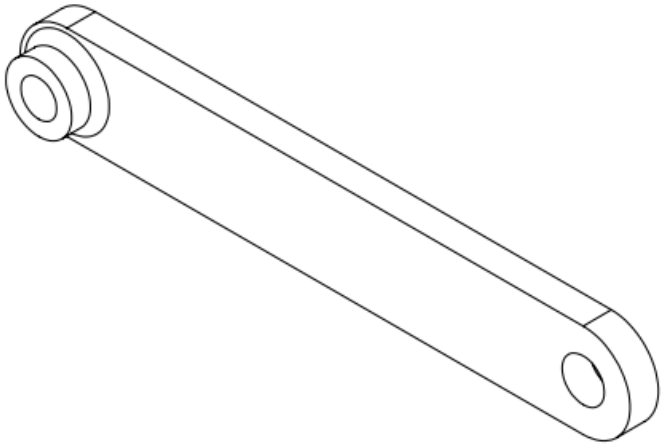


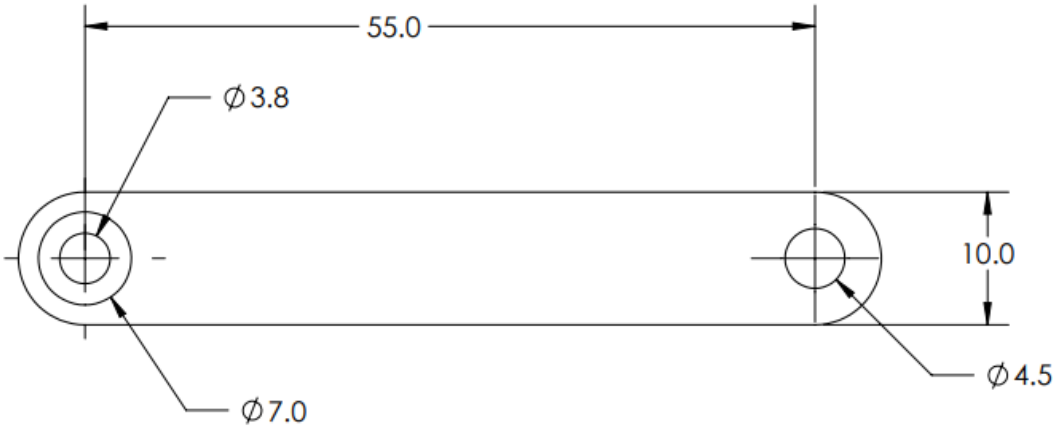


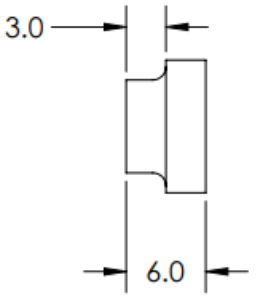
Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: BATTERY FEET BACK		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME024	Nxt Asb:	Date: 5-29-19	Scale: 1:1	Chkd. By:

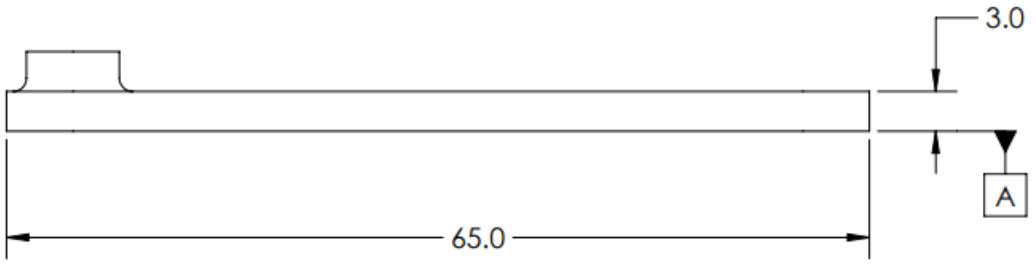
NOTES

1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING





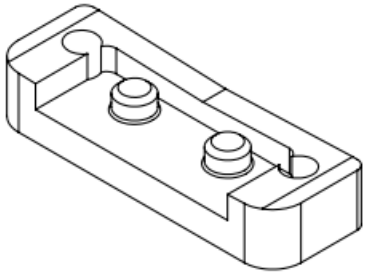


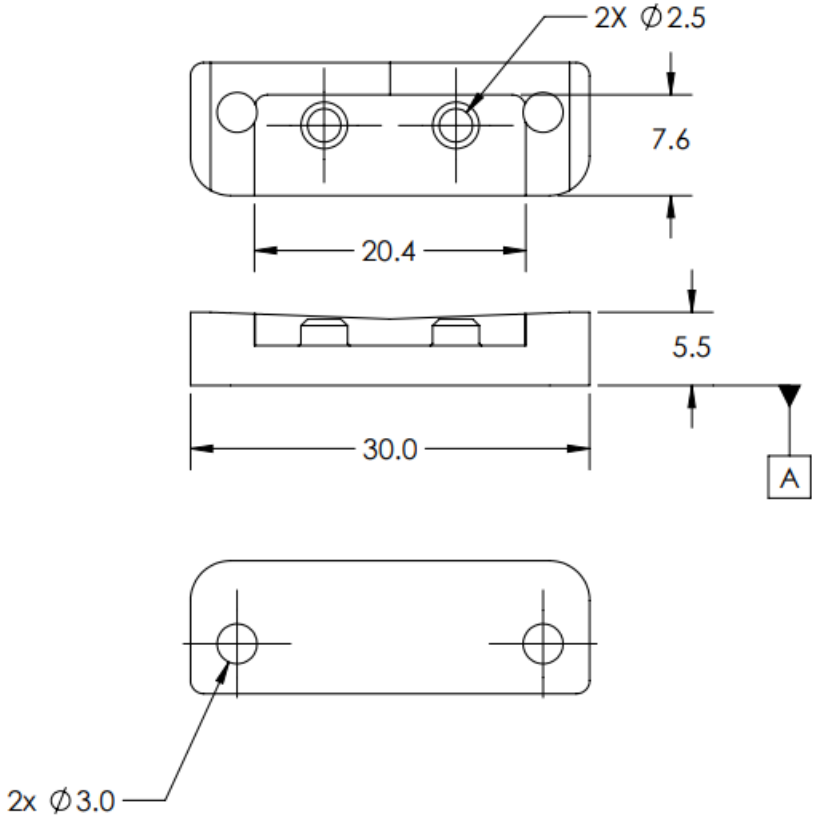


Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: BATTERY FINGER		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME025	Nxt Asb:	Date: 5-29-19	Scale: 2:1	Chkd. By:

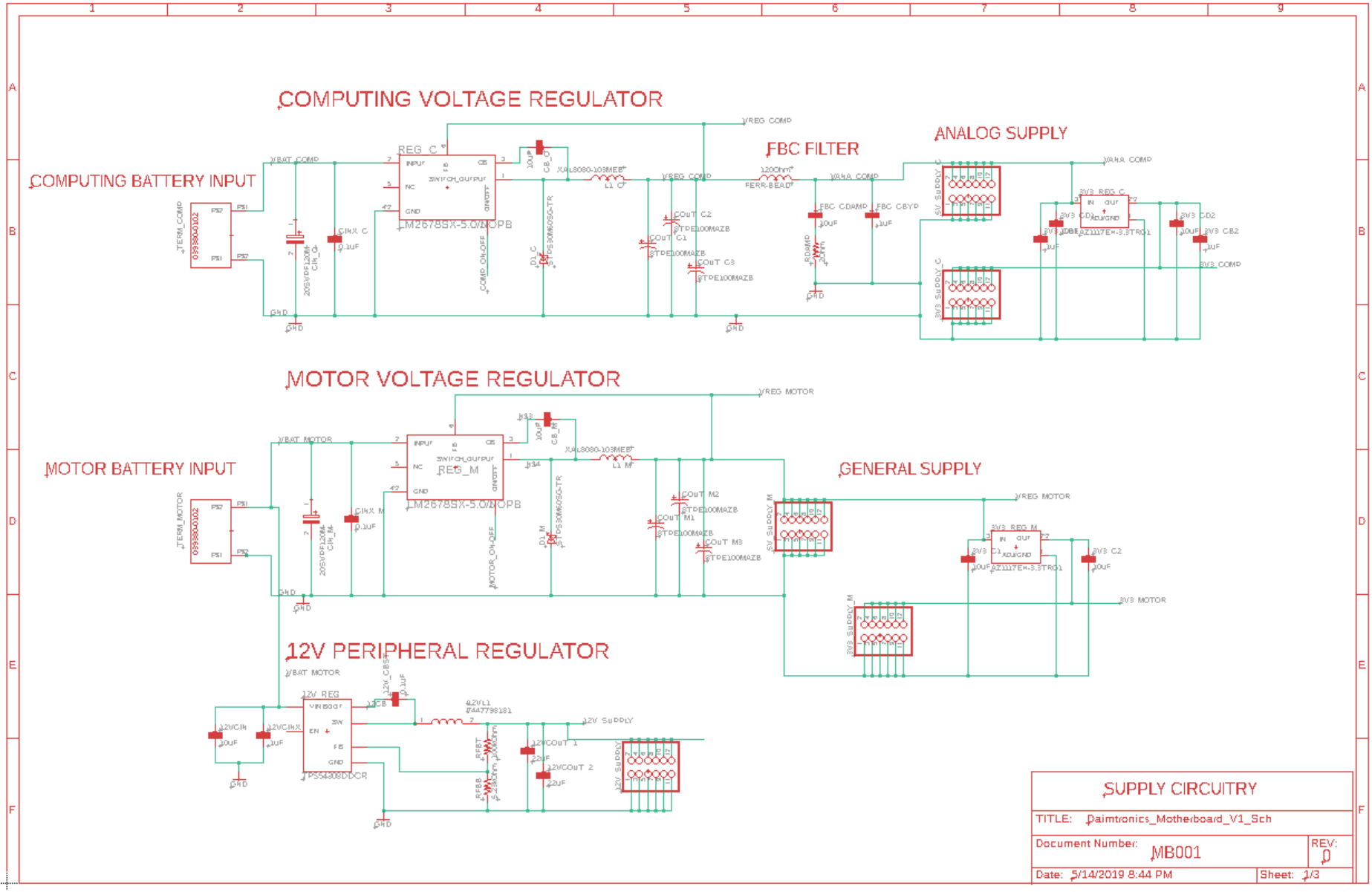
NOTES

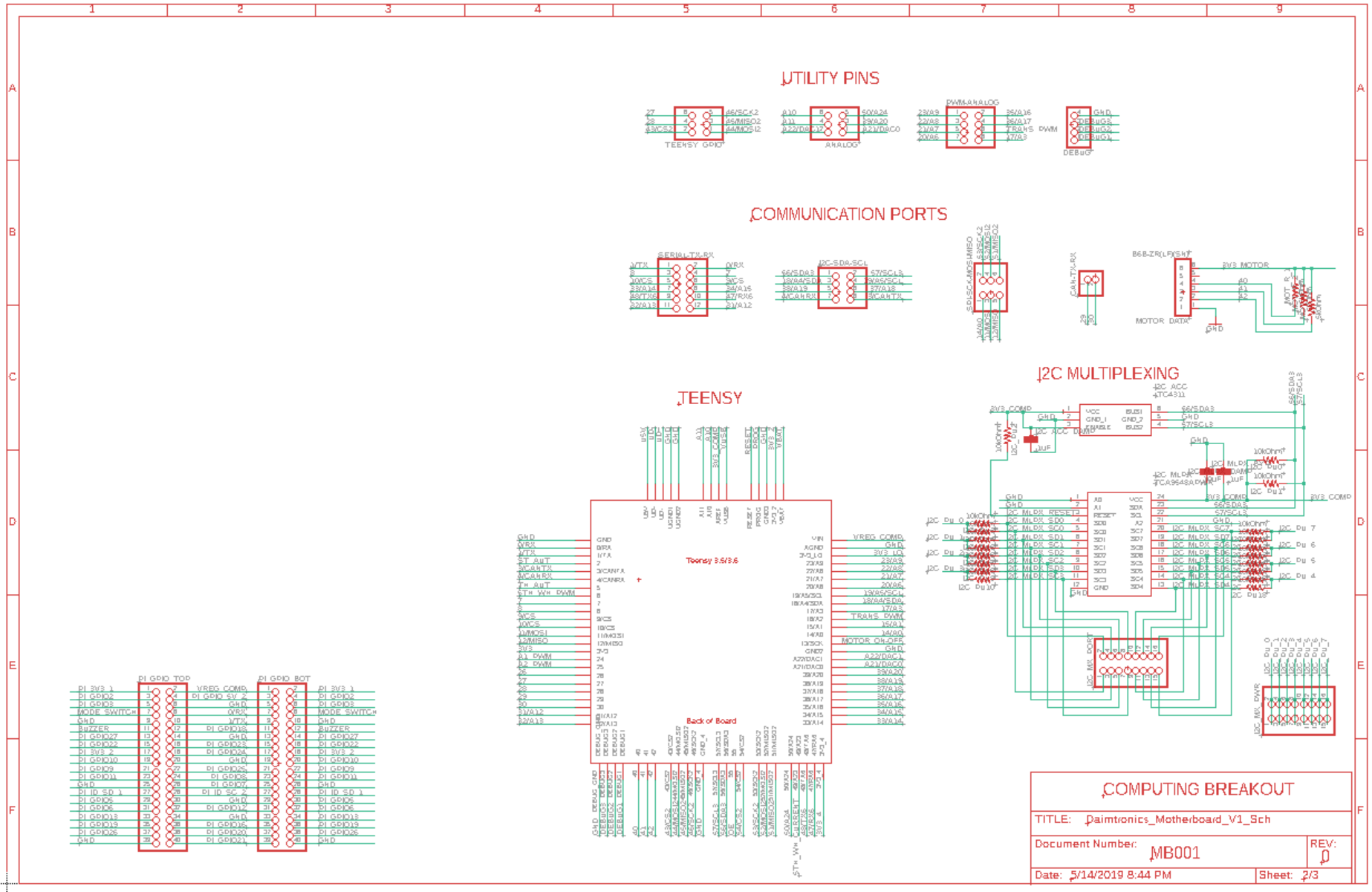
1. REFER TO SOLID MODEL FOR DIMENSIONS NOT INDICATED ON THIS DRAWING.
2. INTERPRET IN ACCORDANCE WITH ASME Y14.5-2009
3. MATERIAL: PETG OR ABS
4. COLOR: WHITE
5. DATUM A IS THE INTENDED PRINTING BUILD PLATE SURFACE
6. ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED
7. ALL TOLERANCES ARE ±.2MM; PART DIMS ARE INTENDED FOR FDM PRINTING

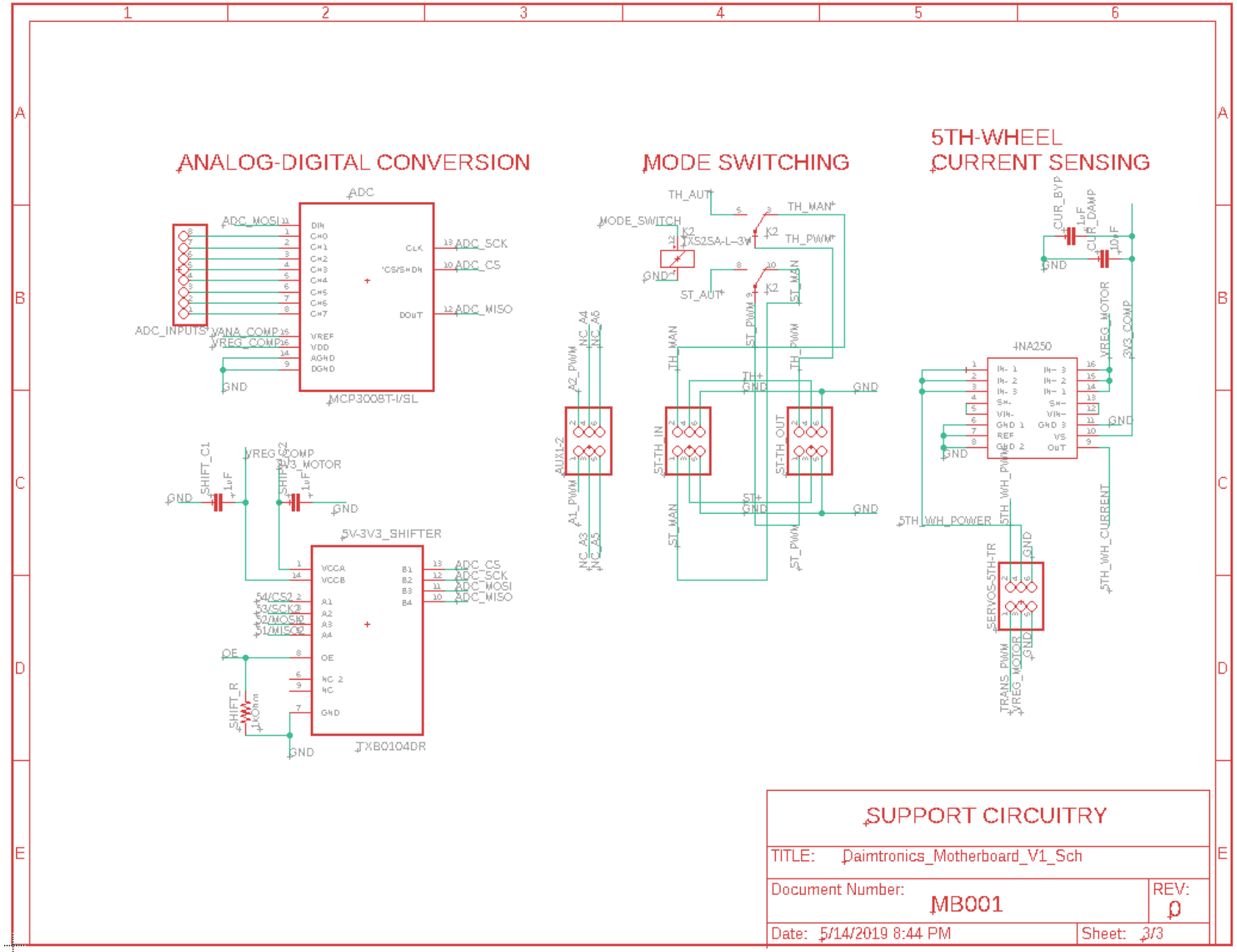




Cal Poly Mechanical Engineering	Lab Section: 01	Team: DAIMTRONICS	Title: SERVO CLASP		Drwn. By: R. TAN
ME 430 - Winter 2019	Dwg. #: ME026	Nxt Asb:	Date: 5-29-19	Scale: 2:1	Chkd. By:


COMPUTING VOLTAGE REGULATOR
COMPUTING BATTERY INPUT
FBC FILTER
ANALOG SUPPLY
MOTOR VOLTAGE REGULATOR
MOTOR BATTERY INPUT
GENERAL SUPPLY
12V PERIPHERAL REGULATOR
SUPPLY CIRCUITRY
TITLE: Daimtronics_Motherboard_V1_Sch
Document Number: MB001
REV: D
Date: 5/14/2019 8:44 PM
Sheet: 1/3


UTILITY PINS
TEENSY GPIO
ANALOG
PWM/ANALOG
PWM
DEBUG
COMMUNICATION PORTS
SERIAL-TX-RX
I2C-SDA-SCL
CAN-TX-RX
B6B-ZR(LF)(SN)
3V3_MOTOR
MOTOR DATA
I2C MULTIPLEXING
I2C ACC
TC4311
I2C MLDX
TCA9548A
I2C MX DOUT
I2C MX PWR
TEENSY
Teensy 3.5/3.6
Back of Board
PI GPIO TOP
PI GPIO BOT
COMPUTING BREAKOUT
TITLE: Paimtronics_Motherboard_V1_Sch
Document Number: MB001
REV: D
Date: 5/14/2019 8:44 PM
Sheet: 2/3


ANALOG-DIGITAL CONVERSION
MODE SWITCHING
5TH-WHEEL
CURRENT SENSING
ADC
MCP3008T-I/SL
5V-3V3_SHIFTER
TXB0104DR
SUPPORT CIRCUITRY
TITLE: Daimtronics_Motherboard_V1_Sch
Document Number: MB001
REV: 0
Date: 5/14/2019 8:44 PM
Sheet: 3/3

MEASUREMENTS		
Times:	1.41	sec
	1.38	sec
	1.42	sec
	1.40	sec
	1.35	sec
	(^ 'times')	

CONSTANTS		
(ft/s)/(mph):	1.46667	('fps')
Confidence:	95%	

This analysis utilizes a T-distribution due to only having 5 data points. The total uncertainty is calculated for the distance, time, and speed by combining the measurement, statistical, and propagated uncertainty for each. Only time has direct statistical uncertainty, and only speed has underlying propagated uncertainty, which is also its only source of uncertainty since everything is contained within the time and distance uncertainties upstream. Formulas are copied and shown where used.

UNCERTAINTY ANALYSIS			
Distance:	20	ft	
Measurement Uncertainty:	0.003	ft	=1/16/12/2
Statistical Uncertainty:	0.000	ft	
Propagated Uncertainty:	0.000	ft	
Total Uncertainty:	0.003	ft	=SQRT(SUMSQ(G4:G6))
Distance Upper:	20.003	ft	=G3+G7
Distance Lower:	19.997	ft	=G3-G7
Average Time:	1.392	sec	=AVERAGE(times)
Measurement Uncertainty:	0.100	sec	
Statistical Uncertainty:	0.034	sec	=CONFIDENCE.T(1-C12,
Propagated Uncertainty:	0.000	sec	STDEV.S(times),COUNT(times))
Total Uncertainty:	0.106	sec	=SQRT(SUMSQ(G13:G15))
Time Upper:	1.498	sec	=G12+G16
Time Lower:	1.286	sec	=G12-G16

Average Speed:	14.37	ft/s
Measurement Uncertainty:	0.00	ft/s
Statistical Uncertainty:	0.00	ft/s
Distance Prop. Uncert.:	0.002	ft/s
Time Prop. Uncert.:	1.09	ft/s
Propagated Uncertainty:	1.09	ft/s
Total Uncertainty:	1.09	ft/s
Speed Upper:	15.46	ft/s
Speed Lower:	13.28	ft/s
Average Speed:	**9.80**	**mph**
Measurement Uncertainty:	0.00	mph
Statistical Uncertainty:	0.00	mph
Distance Prop. Uncert.:	0.00	mph
Time Prop. Uncert.:	0.74	mph
Propagated Uncertainty:	0.74	mph
Total Uncertainty:	**0.74**	**mph**
Speed Upper:	**10.54**	**mph**
Speed Lower:	**9.05**	**mph**

=G3/G12	
=G7/G12	
=G3/G12^2*G16	
=SQRT(SUMSQ(K6:K7))	
=SQRT(SUMSQ(K8,K5,K4))	
=K3+K9	
=K3-K9	
=K3/fps	
(etc)	

Appendix M: Daimtronics Operator's Manual

Battery Charging Procedure:

1. Charge the batteries with the provided charger by plugging in the charger to an outlet and connecting the battery terminals to the charger ports.
2. Disconnect the batteries from the charger when the lights indicate a fully charged battery.

Setup Procedure:

1. Attach bare leads of battery-to-board wires to the motherboard. DO NOT ATTACH THE LIPO BATTERIES YET. Attaching the batteries first risks sparking the bare leads.
2. Plug the LiPo batteries in once the leads are secured on the motherboard.
3. Turn on the ESC with the switch underneath the chassis.
4. Turn on the RC controller.
5. The Teensy and Pi will now be running whatever has been loaded into their memory.

Uploading Algorithm Procedure:

Warning: **Do not plug in a USB to the Raspberry Pi while it is being powered by the battery!**

Running the motherboard off battery power while plugging in a USB to upload might cause the Raspberry Pi to short-circuit.

The first thing to do is get all of the source code for the Daimtronics project. Go to https://github.com/nfurbeyr/daimtronics and clone or download the repository to your host computer.

Setting up ROS on the Pi and Running a Simulink Model to Communicate with the System

1. Make sure that you have the following Raspberry Pi and ROS packages for Simulink / MATLAB installed on your laptop. You can download these through the "Add-On ->Get Add-Ons" option in MATLAB (Figure 1). Refer to MathWorks documentation for additional help if needed.
    a. Robotics System Toolbox Interface for ROS Custom Messages
    b. Embedded Coder
    c. Simulink Coder
    d. MATLAB Coder
    e. Robotics System Toolbox
    f. Simulink Support Package for Raspberry Pi Hardware
    g. MATLAB Support Package for Raspberry Pi Hardware

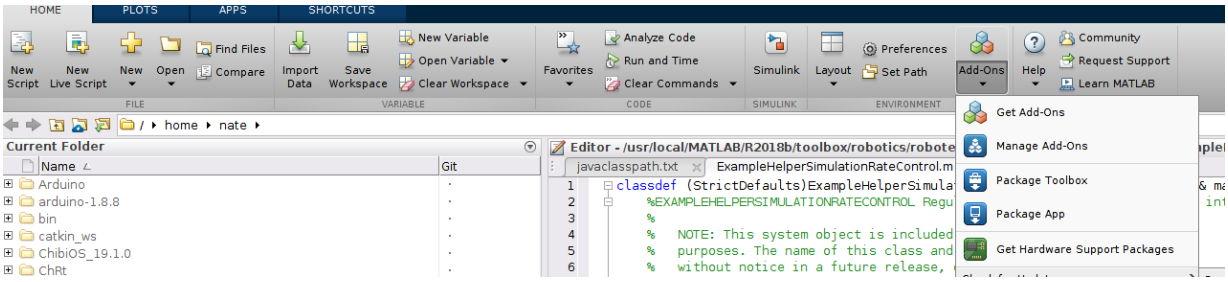


Figure 67: Downloading required packages in Simulink / MATLAB

2. On a PC with connected to Wi-Fi, open the “example_model.slx" Simulink model found within “daimtronics/simulink_models.”
3. Open “Model Configuration Parameters” (Ctrl-E) and select “Hardware Implementation"
    a. Select “Raspberry Pi – Robot Operating System (ROS)" under “Hardware Board.”
        i. For this option to be available you need to have both the Robotics System Toolbox and Simulink Support Package for Raspberry Pi Hardware addons.
    b. Under “Target Hardware Resources -> Board Parameters” enter the Raspberry Pi IP address for Device Address and “pi” and “daimtronics” as the username and password, respectively.
        i. The Pi must be hooked up to the same WiFi network as the PC. If you have a mobile device that can create a HotSpot connection, this a viable option.
        ii. To first connect the Pi to WiFi, you will need to interface with the Pi by hooking up a monitor to the HDMI port and a keyboard / mouse to the USB ports.
        iii. Once connected to Wifi, to get the Raspberry Pi IP address, you can open a terminal on the Pi and type ‘ifconfig’ into the command line to get a number: XX.XXX.XX.XXX.
    c. Under “Target Hardware Resources -> Build options” select “Build and run” and for the Catkin workspace select “~/daimtronics/semi_catkin_ws”

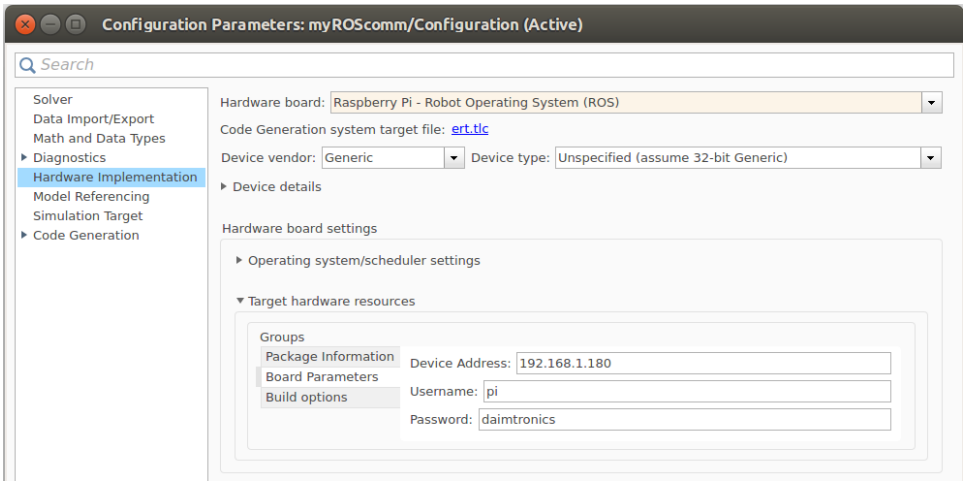


Figure 68: Configuring Simulink to build Simulink model as a ROS node on the Raspberry Pi

Click the drop-down arrow next to “Code Generation” (left side of Configuration Parameters window) and select the “Interface” option that appears.

d. Activate “External Mode” by clicking on the check-box.
e. Select “tcpip” for the "Transport Layer" under the external mode configuration

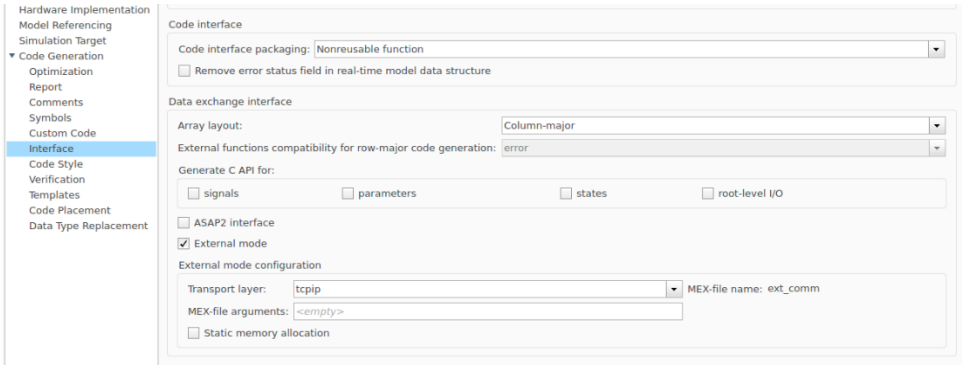


Figure 69: Configuring Simulink to run in External Mode for our model.

4. Change the configuration of the ROS Simulink blocks to connect to the correct IP addresses for the laptop and the Raspberry Pi.
    a. Double click on one of the ROS Subscriber or Publisher blocks.
    b. Click on “Configure network addresses” in the new window.
    c. For ROS Master, type in the IP address of the PC that is running Simulink, and for Node Host, type in the IP address for the Raspberry Pi that was used in Step 3.
    d. ROS Master Port should be 11311.
5. In the MATLAB command window, write the following lines to let Simulink and MATLAB know what IP address the ROS core system is running on.
    a. “setenv('ROS_MASTER_URI','http://XXX.XXX.XXX.XXX:11311')” *with the X’s replaced with the IP address of the* ***Raspberry Pi***
    b. “setenv('ROS_IP','XXX.XXX.XXX.XXX')” *with the X’s replaced with the address of the* ***PC***
6. Open a terminal on the Raspberry Pi. You can either do this directly on the Pi with a monitor and keyboard-mouse combo, or SSH into the device using a PC and Wi-Fi.
    a. SSH with “pi@XXX.XX.XXX.XX” (based on the IP address of the device) and a password of “daimtronics.”
7. In the terminal, change the directory to “home/pi/daimtronics/semi_catkin_ws/src/launch.”
8. Type the command: “roslaunch semi_truck_bags.launch” and press Enter.
    a. If you need to run different nodes than is in the launch file, modify lines to mimic the following format with your specific package and node in Figure 4

```
<node pkg="your_package_here" type="your_node_here" name="your_node_name_here" />
<node pkg="your_package_here" type="your_node_here" name="your_node_name_here" />
```

Figure 70: Adding additional nodes to the launch file.

If all of the previous steps worked, the ROS nodes on the Raspberry Pi will be running, and the PC should have its Simulink model ready to send and receive messages with the Pi.

Tuning the Simulink Model Parameters in Real-time

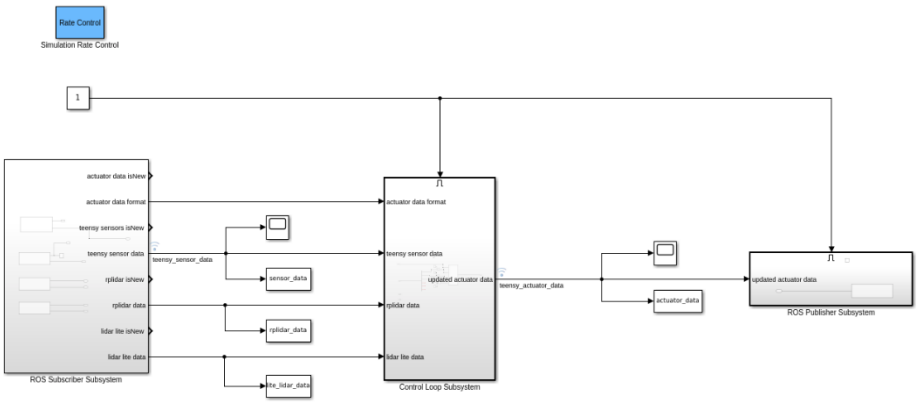


Figure 71: Simulink example

1. Follow the steps in “Uploading a Simulink Model to the Raspberry Pi”. This should get you to a state where the ROS nodes are running on the Pi. They are currently waiting for a Simulink model to read sensor data and output actuator data to the system.
2. In Simulink, make sure that the “Simulation Mode” is "Normal."

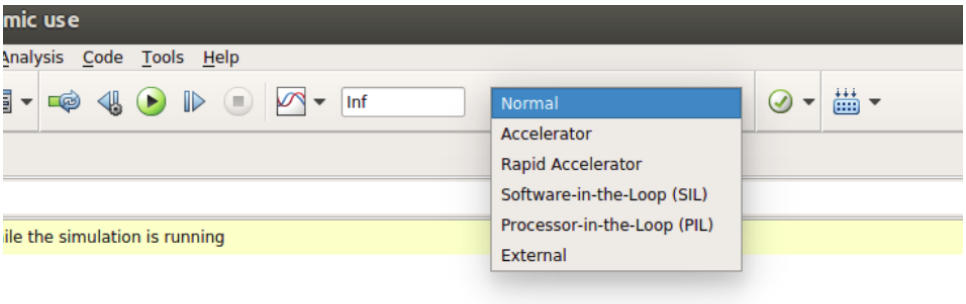


Figure 72: Simulink Simulation Mode

3. Run the model simulation on the PC in Simulink by clicking on the green “Play Button”.
    a. This will effectively use Simulink to create a ROS node that is running on the PC. It can communicate to the Raspberry Pi through the TCP/IP settings we configured earlier.
    b. Messages will be received, processed in the model, and sent back to the ROS network as frequently as is specified in the “Simulation Rate Control” block.
4. Open scopes connected to the actuator_data and sensor_data buses.
    a. This should show the current readings of the sensors connected to the Teensy.
5. Alter block values in the Simulink model, and observe the changes in the scope.
    a. You can only change some parameters in the model, such as outputs from “Constant” blocks. Other alterations in the model will be denied, and a warning message that the model cannot change while the simulation is running will pop up.

6. When you have finished with the Simulation, stop the model and view the workspace in Matlab.
    a. There should be simout data that contains arrays of all of the sensor and actuator values that were running during the simulation. You can access this data and analyze how the system responded.

Custom ROS Model

2. Open a terminal on the Raspberry Pi. You can either do this directly on the Pi with a monitor and keyboard mouse combo, or SSH into the device using a PC and Wi-Fi.
    a. SSH with "pi@XXX.XX.XXX.XX" (based on the IP address of the device) and a password of "daimtronics."
3. In the terminal, change the directory to "home/pi/daimtronics/semi_catkin_ws/src/semi_truck/src".
4. Create a new .cpp file in this folder. This .cpp file should contain all of the autonomous algorithm code that acts as the "brains" of the system.
    a. This file will need to subscribe to the ROS topics of the RPLidar and the Lite Lidar, and publish to the pi_comm_node. A **truck_template_node.cpp** file is in the "src" folder that gives some direction on what the file should look like.
5. Change the CMakeLists.txt in "home/pi/daimtronics/semi_catkin_ws/src/semi_truck" so that the new file is recognized when building the project.
    a. Add the following line around line 140 (there should be similar code as a reference):
        i. add_executable(your_file_name_node src/your_file_name.cpp)
    b. Add the following line around 157 (again there should be similar lines of code here):
        i. target_link_libraries(your_file_name_node ${catkin_LIBRARIES})
        ii. **The first argument (eg your_file_name_node) to each line's function call must match exactly.**
6. Change the terminal directory to "home/pi/daimtronics/semi_catkin_ws".
7. Run the command: "catkin_make"
    a. This may take some time, especially if new packages have been integrated into the project. This is where all of the files get compiled into machine code.
8. Update semi_truck_bags.launch (home/pi/daimtronics/semi_catkin_ws/src/launch) to run the new file by adding the following line.
    a. <node pkg="semi_truck" type="your_file_name_node" name="your_file_name_node" />
    b. Make sure that the names here match up with what was added in Step 4.
9. Run the new launch file by typing "roslaunch semi_truck_bags.launch" at the terminal.
10. Once the session is done, and you have killed the program (by type Ctrl-C in the terminal that you launched the nodes) a ROSbag will be generated in /home/pi/daimtronics/semi)catkin_ws/rosbags.
    a. You can convert this data to csv's with entire sets of data for individual topics that were running during the launch. In the terminal type "./readBag.sh {filename}.bag"
    b. The previous command will run a script to make a new directory with the ".bag" extension removed. All of the topics converted to csv's will be placed here. Note that for longer simulations, this may take a long time to convert data.

Testing Procedure:

Warning: **Releasing the trigger on the RC controller will automatically stop the platform.**

In manual mode, if at any time an unsafe or unwanted action is about to be performed, or if at any time you need to stop the platform, simply release the trigger on the RC controller to send an automatic braking command to the platform. Alternatively, in either manual or automatic mode, release the dead man's switch on the RC controller, button on SW1.

Figure 73 Deadman's Switch is on SW1

1. Remove the body housing by unscrewing the four M3 screws attaching the body housing to the Tractor.
2. Mount the batteries in the battery mounts on the Tractor.
3. Connect the battery connectors for both batteries to the motherboard. Both batteries are identical, therefore it doesn't matter which one is plugged in to which port.
4. Reattach the body housing to the Tractor by screwing in the four M3 screws.
5. Turn on the RC Controller by turning the switch to the "ON" position.
6. Turn on the Tractor by turning the ON/OFF switch to the "ON" position.
7. Flip the "AUX Ch.2" switch on the RC Controller to switch to autonomous mode.
8. Hold the RC Controller trigger to begin running the autonomous algorithm currently uploaded. The algorithm will continue running as long as the trigger is pressed.
9. Let go of the trigger to bring the platform to a stop.

Adding Sensors:

1. Physical integration: The physical attachment of an additional sensor to the platform is entirely up to the operator and dependent on the sensor.
2. Electrical integration: Wires supplying power to the sensor and sending outputs must be interfaced with the motherboard.
    a. All wires must have female 0.1" pitch connectors.

b. Available supply voltages are 3.3V and 5V, each either on the motor-side of the power circuit (noisier) or the computing-side (quieter).
c. SPI, I2C, and serial ports are available if the sensor communicates via these protocols.
d. 3.3V and 5V analog input ports are available, as well as general digital GPIO pins.
e. The Raspberry Pi has USB ports available, but check that the current draw of the sensor combined with the other devices connected to the Pi's USB ports does not exceed the maximum draw across all of the ports, 1200mA. If this sensor will put the Pi over the limit, cut the supply voltage and ground wires inside the USB cable and reroute it to either the noisy/motor or quiet/computing 5V supply rail on the motherboard.

3. Software integration: Ideally, there will be a driver online that has already been written to control the sensor. If not, you will need to write the code to interface with the device.
    a. If the sensor is meant to be interfaced with the Teensy
        i. Create a .cpp task file in "teensy_chibios/src." (see Software Overview 1D in this document)
        ii. Add a thread for this task to the chSetup function in main.ino
    b. If the sensor is meant for the Raspberry Pi
        i. Find a package online for controlling the device (or write one yourself with help from ROS tutorials).
            1. If you write one yourself, the place to put it is in the semi_truck package under daimtronics/semi_catkin_ws/src/semi_truck/src.
        ii. Add the package and executable name to the semi_truck_bags.launch launch file.

Adding Peripherals:

A 'peripheral' is defined as any device that is not a sensor or preexisting actuator, such as an additional fan, lights, or another servo. It follows the same procedure as adding additional sensors, except that software integration will be up to the operator and is dependent on the nature of the device.

Adding Computing Modules:

Additional computing modules (such as another Pi, an Nvidia Jetson, or other small computation device) can potentially be added to the vehicle. Such a change will require the design of new mounting and integration with the current software suite. Communication can be established with the Pi and Teensy either through one of the serial ports on the Teensy or with one of the non-serial communication buses on the Pi, including USB if applicable.

An additional ROS node will have to be added to handle the communication, but all sensor, actuator, and state values for the system can potentially be shared with the new device. This new device can either provide additional inputs to the control system on the Pi, or the ROS network can be reworked to run the control system on the new hardware.

Manual Control:

The vehicle can be switched between manual and automatic control through the [INSERT SWITCH NUMBER] switch on the RC controller. When the switch is in [MANUAL SWITCH STATE], all actuator signals from the RC controller are sent directly to the motor and the steering servo. The trigger on the RC controller is the throttle and the wheel on the side of the controller is the steering input.

Motherboard Overview:

1. Powering Motherboard
    a. The motherboard is powered by two separate LiPo batteries.
    b. The connector for the motor-side LiPo battery is spliced such that there are two XT-90 connectors and one set of bare wires. The bare wires go into the terminal of the motherboard. One XT-90 connector goes to the battery and the other goes to the ESC.
    c. The connector for the computing-side LiPo battery is an XT-90 connector on one end and bare wires on the other that go to the computing-side terminal on the motherboard.
2. Powering Devices
    a. All supply pins are 0.1”-pitch male headers on the motherboard, which require mating wires to have 0.1”-pitch female connectors.
    b. Both the motor-side and computing-side power circuits have 3.3V and 5V supply rails available. The motor-side pins are labelled “noisy” and the computing-side pins are labelled “quiet”.
    c. An additional adjustable power line for peripherals is provided off of the motor battery that is default set to 12V. The voltage can be changed by swapping out the lower feedback resistor with a value of 59.6*(Vout - .596) kOhms. For 12V, this comes out to

5.23 kOhms. Swapping this resistor requires soldering.

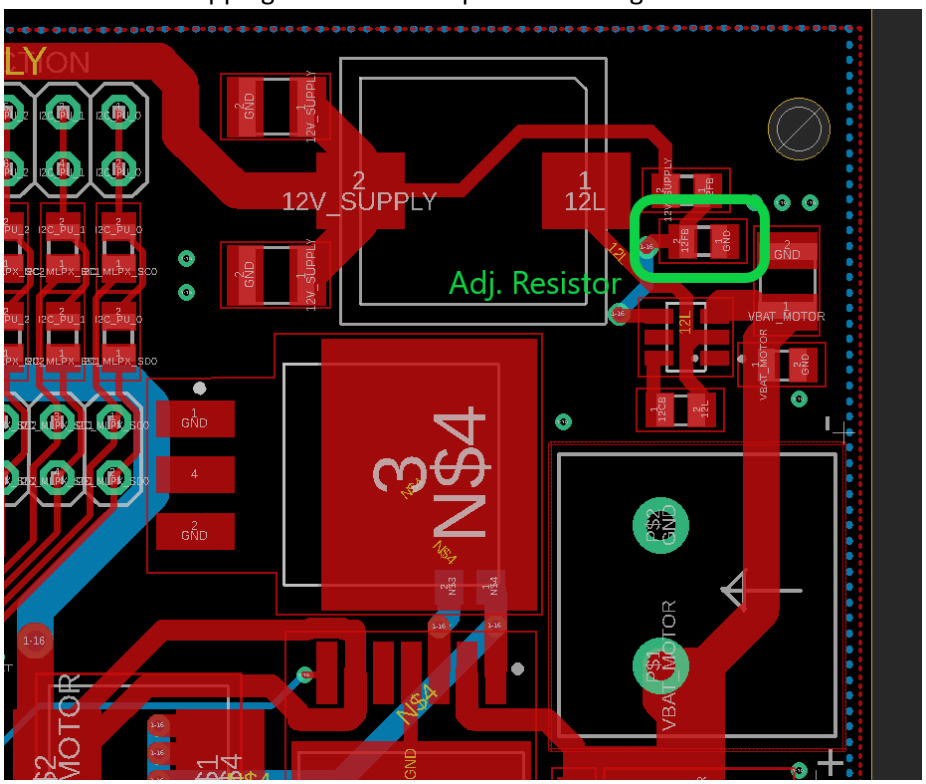


Figure 8: Adjustable power supply resistor location

   d. The noisy power lines should be the default, reserving the quiet supply lines for devices that require less noise on their voltage supply, such as computing modules and sensors whose performance degrades at the level of noise present on the motor/noisy side.

3. Input/Output Ports
   a. All I/O pins are 0.1”-pitch male headers on the motherboard, which require mating wires to have 0.1”-pitch female connectors.
   b. There are four different I/O ports: GPIO, analog-input, PWM-output, and analog-input/PWM-output.
   c. The most restrictive ports should be used first. An analog input should be routed to the analog-input port if there are any available slots prior to being routed to an analog/PWM pin. Likewise, pins used just as a digital switches should first be routed to the GPIO port before using up any of the special-use (analog and/or PWM) pins.
   d. [ADD TABLE OF PORTS AND CORRESPONDING PINS]
4. Communication Ports
   a. All communication port pins are 0.1”-pitch male headers on the motherboard, which require mating wires to have 0.1”-pitch female connectors.
   b. Each communication port has multiple channels. The channel’s designation corresponds to its designation in the Teensy documentation. The order in which channels should be utilized is given on the motherboard, with higher-priority channels starting at the left of each port. The priority is such that there is a minimal loss of alternate, special pin functionality if channels are used in order. [ADD PICTURE]

c. Communication ports can, if all other pins are utilized, be used as GPIO pins. Some pins in communication ports also contain PWM-output or analog-input functionality if all other special-use pins are already utilized. These can be seen on the Teensy's pinout diagram. This should be the operator's last resort when trying to add numerous additional components.
d. [ADD TABLE OF CHANNELS AND CORRESPONDING PINS]

5. Other
    a. The fan cooling the casing of the computing module is a 50mm x 50mm 5V brushed DC centrifugal blower. It should be powered off the 5V noisy supply rail. There is an opening in the casing specifically for its supply wires. The fan has an additional blue control wire that is not needed and should be left unconnected but physically secured.

Software Overview:

Teensy / Arduino / ChibiOS

1. Arduino IDE
    a. The Teensy is programmed using the Arduino IDE, which is where programmers write the code to be deployed on the Teensy. Find and download the Windows, Mac or Linux distribution of the software at https://www.arduino.cc/. When working on a Sketch (Arduino program), there are two main features of the IDE that are used: Verify (upper left check mark) and Upload (upper left arrow).
        i. Verify will compile the code that has been written and check for any errors that the compiler finds. This is useful when adding additional code and checking for any lapses in syntax. This option does not need a Teensy to connect to.
        ii. Upload requires the Teensy to be connected to the computer via a micro-USB. It both compiles the code (similar to Verify) and then uploads the resulting file from compilation to the Teensy. When successfully uploaded, the program will automatically run on the Teensy. Since the compiler needs to know specifics about the Teensy architecture, the Teensy 3.6 board needs to be selected under the "Tools" tab. More info about this is given in the Teensy Loader section.

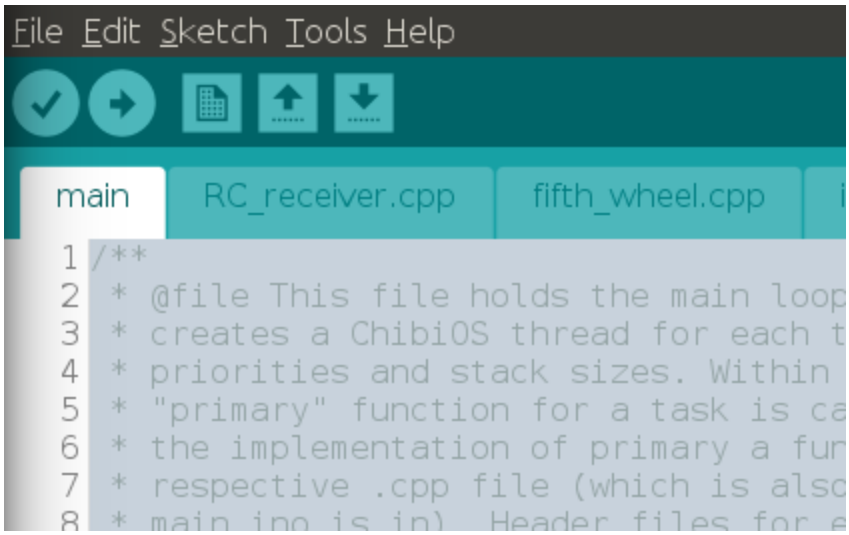


Figure 9: ChibiOS Main

b. Our Project makes use of a few different Software packages that are associated with the Arduino IDE. General information on how to download additional libraries should be readily available on online.
    i. BNO055: This is a driver that controls the IMU for angle data. Our code uses the Adafruit Unified Sensor and Adafruit BNO055 libraries. To install these, in the IDE go to "Tools" -> "Manage Libraries" and type in the library that you want to add. Once done downloaded, restart the IDE so that the libraries are recognized.
    ii. VL530X: This is a driver that controls the TOF sensors. You can download this library in the same way as the BNO055.
    iii. Servo: This library is default installed in each distribution of the Arduino IDE.
c. All of the files for running on the Teensy are found in the "teensy_chibios" folder within the "daimtronics" repository. The code written is found within "teensy_chibios/src" and there is currently a folder "main" that holds the Arduino Sketch to be run on the Teensy. There is a "main.ino" within this folder, and one can open this file in the Arduino IDE to Upload it to the Teensy.
d. The Teensy project is organized so that there is a single main.ino file within the "main" folder. Also in the folder is a collection of .cpp files. Each .cpp file holds the function for running the code for a single task. **The .cpp files need to be in the same directory as main.ino in order for the IDE to link the code between main.ino and respective .cpp files.** Also within the "main" folder is an "include" folder that holds all of the header files for the .cpp files. You will find at the top of "main.ino" that all of the files "include" are #included for successful compilation, otherwise you will get an undefined reference error.

2. ChibiOS
    a. ChibiOS is another package that is necessary to download for the project to run. There is an official ChibiOS release, but this release does not have support for the Teensy architecture that we need. Download the Teensy compatible version from https://github.com/greiman/ChRt. Once downloaded, copy this library to the

Arduino/libraries folder. To verify that you are copying it to the right folder, you should see the Adafruit BNO055 library folder that was acquired through the “Manage Libraries” option within the Arduino IDE.

3. Teensy Loader
    a. The Teensy Loader is required for being able to deploy a program onto the Teensy. The Arduino IDE does not come with support for the Teensy by default. There are tutorials for how to do this at https://www.pjrc.com/teensy/tutorial.html. The tutorials should get you to a point where you can upload a simple “Blink” example sketch to the Teensy and watch the LED blink.

Raspberry Pi / ROS

1. Basics behind ROS
    a. It is recommended that anyone who is reading this has a basic understanding of how ROS works, including the concepts of ROS nodes, topics, publishers, subscribers and messages. In addition, the basic file structure behind ROS should be familiar to the user: how different "packages" are organized within a project, how to build the source files through the "catkin_make" command, and how to actually run the executables (or ROS nodes) either with "rosrun" or a launch file. All of these concepts are covered in ROS tutorials and going through the process of understanding the basics is essential for a smooth transition into working with our project.
    b. http://wiki.ros.org/ROS/Tutorials is a good source of information on the ROS side.
2. Required Packages
    a. The only required packages for ROS are already included in the Daimtronics repository, (https://github.com/nfurbeyr/daimtronics), but incase you need to access these folders they can be found on github.
        i. https://github.com/cocasema/ros-lidar-lite/ for generating a node to control the 1D LIDAR
        ii. https://github.com/robopeak/rplidar_ros for generating a node to control the 360 spinning LIDAR
3. Launch files
    a. ROS has a feature of Launch files that can start multiple nodes at once. This is useful when running the entire system with all the nodes. Topics will be generated automatically based on what nodes subscribe and publish to topics.
        i. The launch files can be found in /daimtronics semi_catkin_ws/src/, and can be launched by typing “roslaunch {filename}” in the Raspberry Pi terminal.
4. Launching individual nodes
    a. If you just want to run a node without using a launch file, you need to have the ROS master node running. Type “roscore” in the terminal of the Pi. This will take several seconds to finish building.
    b. To run a node, you can use the command: “rosrun <package_name> <executable_name>.
        i. An example of how to run the communication node is: “rosrun semi_truck pi_comm_node” where semi_truck is the package name and pi_comm_node is the executable (node) name.

Deploying Simulink Models

1. Required Simulink – RasPi packages
2. Creating a Simulink model on a PC
    a. Below is a diagram of a simple Simulink model that takes in sensor data (wheel speed, IMU angle, URF distance) from a ROS topic named teensy_sensor_data, increments all of these values by 1, and then converts those incremented values to actuator data (motor output, steer output, fifth wheel output). The new actuator data is published to a separate ROS topic (teensy_actuator_data) for other nodes in the system to process, and to ultimately be sent back to the Teensy for controlling the semi-truck. This is a useless Simulink model in practice, but it demonstrates the general flow that a Simulink model for our project might work with, including the necessary Subscribe and Publish Simulink blocks for communicating with other nodes. Also shown is the Bus Assignment block which takes a general message (eg teensy_actuator_data) and its data components (eg MotorOutput) and assigns incoming values that have just been incremented from the sensor values.
    b. (Note: the second Adding block ends up converting the IMU angle from a float to an int.)

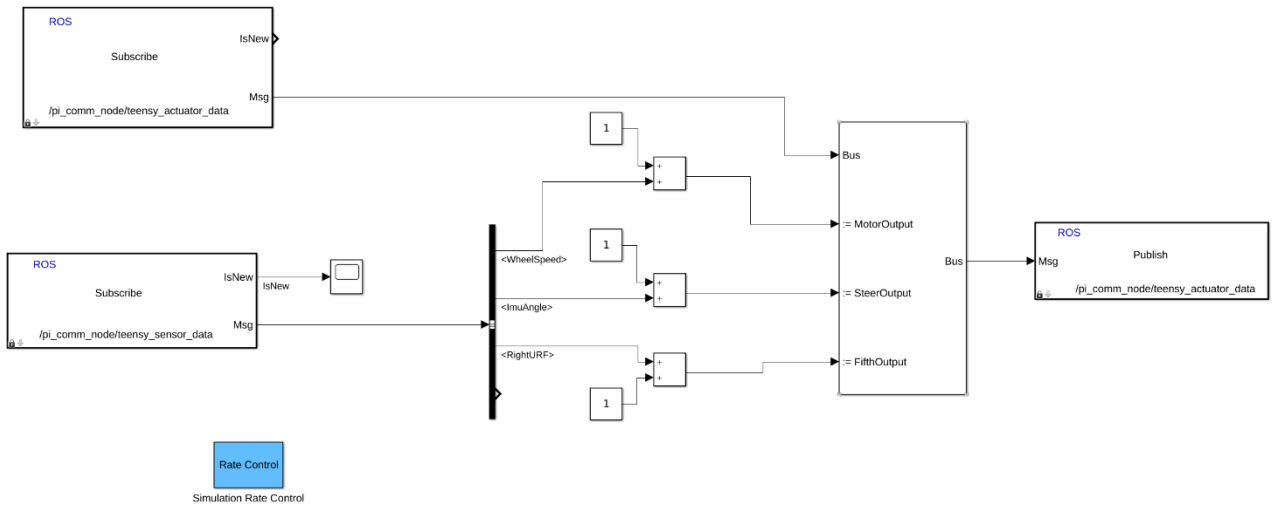


Figure 10: Simulink

3. ROS Messages
    a. The Simulink model must work with the same types of data that are passed between the other nodes in the ROS system. This means that the ROS messages that make up individual pieces of data in a ROS Topic (that runs on the Pi in our project) need to be recognized by Simulink.
    b. First add our semi-truck specific messages to the suite of ROS Custom Messages. The following article by MathWorks gives a good overview of how to do this. You will need to have the specified Simulink packages in Part 1 of the Uploading Algorithm Procedure.
        i. https://www.mathworks.com/help/robotics/ug/ros-custom-message-support.html

ii. If the a matlab_gen directory exists within the project directory, delete thhis matlab_gen directory that has the path: {PATH}/daimtronics/semi_catkin_ws/src/matlab_gen.
iii. To generate the new messages, in the MATLAB console type rosgenmsg("{PATH}/daimtronics/semi_catkin_ws/src") (Replacing {PATH} with the path to the daimtronics repository on your computer)
iv. Add "{PATH}/daimtronics/semi_catkin_ws/src/matlab_gen/jar/semi_truck-0.0.0.jar" to javaclasspath.txt in the MATLAB folder. (See article for more info on where javaclasspath.txt is)
v. Restart MATLAB for the changes to take place.

c. In a Simulink Model on the PC, add the Simulink blocks below (Figure 11) and double click on them to bring up a "Block Parameters Subscribe / Publish" window.
i. Under "Topic source" select "Specify your own."
ii. Under "Topic" type the name of the topic that the model will connect with when deployed on the Pi
iii. Press the "Select …" button and select "semi_truck/Teensy_Sensors" or "semi_truck/Teensy_Actuators"

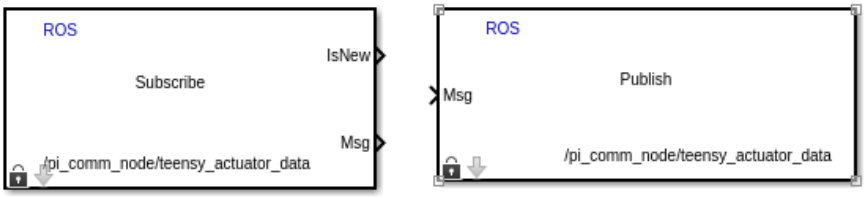


Figure 11: Subscribe and Publish Blocks within Simulink

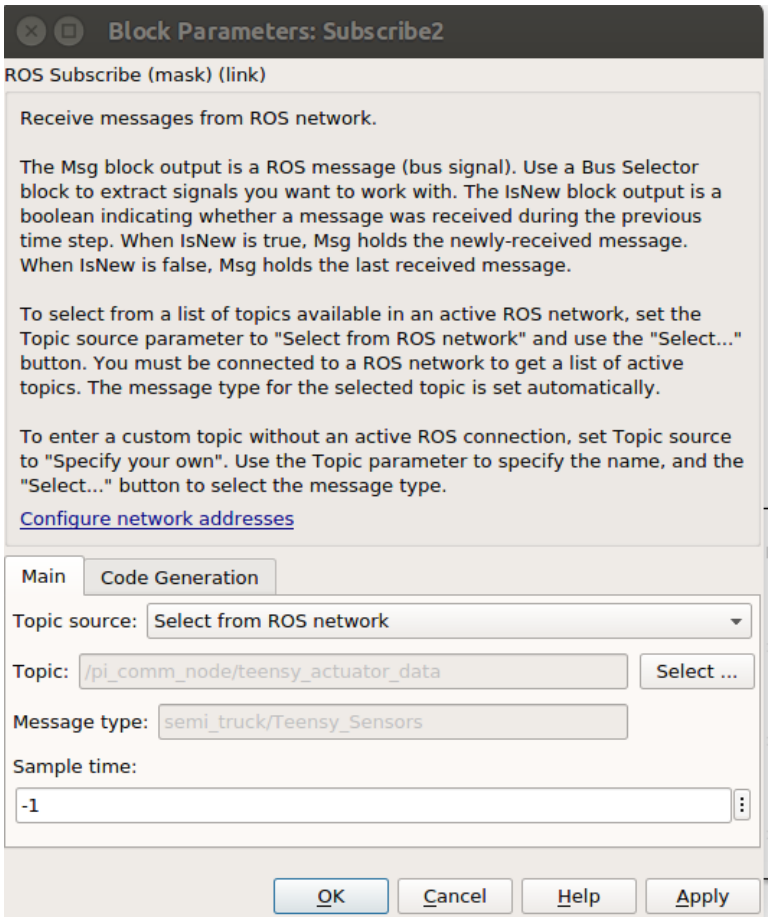


Figure 12: Subscriber / Publisher Block Parameters Window

d. An alternative method requires Simulink to access to a running ROS node on the Raspberry Pi that either subscribes or publishes to a topic with the desired message type to be integrated into the model. Bring up the Subscriber / Publisher Block Parameters Window. From here, configure the network address so that the ROS Master is hooked up to the IP address of the Raspberry Pi on port 11311. This IP address will change depending on what Wi-Fi network you are working on. Make sure the Raspberry Pi is up and running (and able to connect to the PC over Wi-Fi) and start a ROS node that works with a topic using the desired message type. In this case of Figure 13, the pi_comm_node on the Raspberry Pi was running which generated the teensy_actuator_data and teensy_sensor_data topics. In the Block Parameters Window, under “Topic source” select “Select from ROS network.” Press the select button. A dialogue box should open up that allows you to select which type of message the ROS subscriber / publisher is working with, as long as a proper connection to the Pi exists.

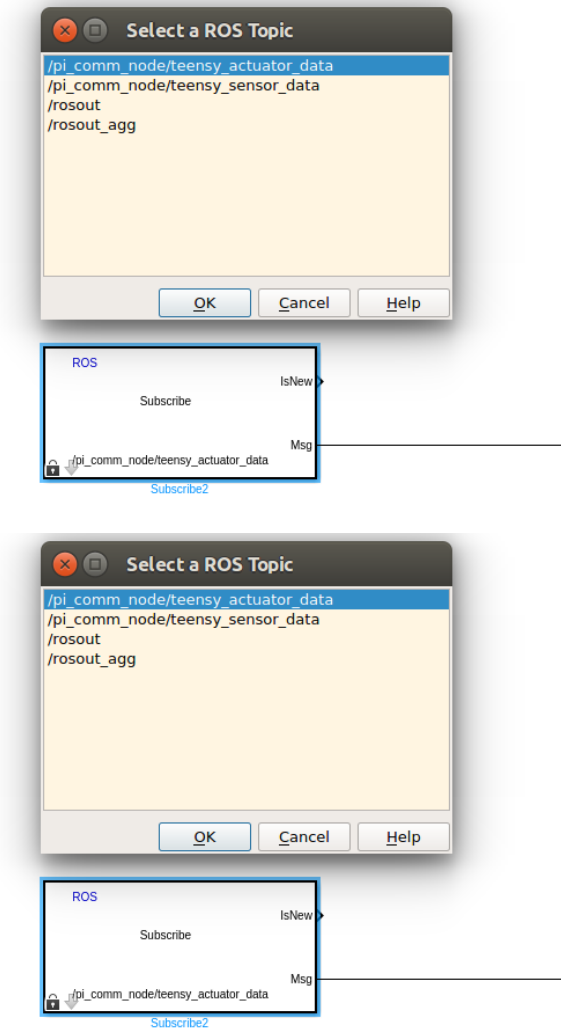


Figure 13: Selecting a ROS topic that is running on the Pi

e. If the above steps are followed correctly, you should have the necessary information to construct Simulink models to run within a ROS network. Everything else in the model besides the Publisher and Subscriber blocks should follow standard Simulink control loop procedures.

Appendix N: Doxygen Report for Software Documentation

Daimtronics

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 actuator_data_t Struct Reference

`#include <system_data.h>`

Public Attributes

- int16_t motor_output
- int16_t steer_output
- int16_t fifth_output

3.1.1 Member Data Documentation

3.1.1.1 int16_t actuator_data_t::fifth_output

3.1.1.2 int16_t actuator_data_t::motor_output

3.1.1.3 int16_t actuator_data_t::steer_output

The documentation for this struct was generated from the following file:

- daimtronics/teensy_chibios/src/main/include/system_data.h

3.2 sensor_data_t Struct Reference

`#include <system_data.h>`

Public Attributes

- int16_t imu_angle
- int16_t wheel_speed
- int16_t right_TOF
- int16_t left_TOF
- int16_t rear_TOF

3.2.1 Member Data Documentation

3.2.1.1 int16_t sensor_data_t::imu_angle

3.2.1.2 int16_t sensor_data_t::left_TOF

3.2.1.3 int16_t sensor_data_t::rear_TOF

3.2.1.4 int16_t sensor_data_t::right_TOF

3.2.1.5 int16_t sensor_data_t::wheel_speed

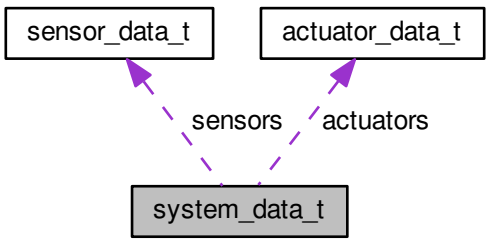
The documentation for this struct was generated from the following file:

- daimtronics/teensy_chibios/src/main/include/system_data.h

3.3 system_data_t Struct Reference

```
#include <system_data.h>
```

Collaboration diagram for system_data_t:



Public Attributes

- int16_t wheel_speed
- int16_t imu_angle
- uint16_t right_TOF
- uint16_t left_TOF
- uint16_t rear_TOF
- uint16_t drive_mode
- int16_t motor_output
- int16_t steer_output
- uint16_t fifth_output
- bool updated
- int16_t deadman
- int16_t drive_mode
- sensor_data_t sensors
- actuator_data_t actuators

3.3.1 Member Data Documentation

3.3.1.1 actuator_data_t system_data_t::actuators

3.3.1.2 int16_t system_data_t::deadman

3.3.1.3 int16_t system_data_t::drive_mode

3.3.1.4 uint16_t system_data_t::drive_mode

3.3.1.5 uint16_t system_data_t::fifth_output

3.3.1.6 int16_t system_data_t::imu_angle

3.3.1.7 uint16_t system_data_t::left_TOF

3.3.1.8 int16_t system_data_t::motor_output

3.3.1.9 uint16_t system_data_t::rear_TOF

3.3.1.10 uint16_t system_data_t::right_TOF

3.3.1.11 sensor_data_t system_data_t::sensors

3.3.1.12 int16_t system_data_t::steer_output

3.3.1.13 bool system_data_t::updated

3.3.1.14 int16_t system_data_t::wheel_speed

The documentation for this struct was generated from the following file:

- daimtronics/semi_catkin_ws/src/semi_truck/include/system_data.h

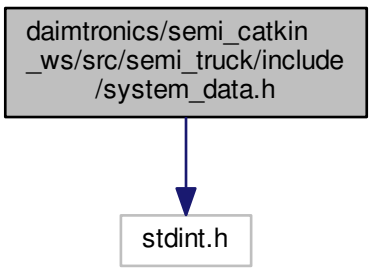
Chapter 4

File Documentation

4.1 daimtronics/semi_catkin_ws/src/semi_truck/include/system_data.h File Reference

```
#include <stdint.h>
```

Include dependency graph for system_data.h:



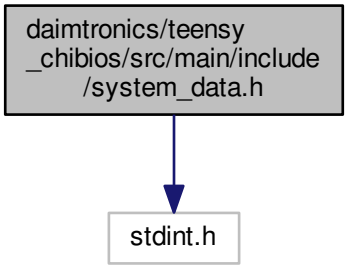
Classes

- struct system_data_t

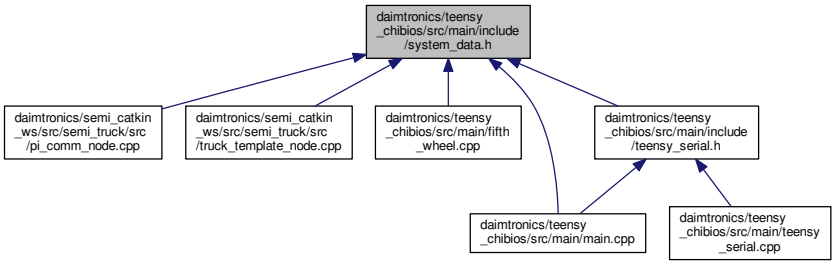
4.2 daimtronics/teensy_chibios/src/main/include/system_data.h File Reference

```
#include <stdint.h>
```

Include dependency graph for system_data.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct sensor_data_t
- struct actuator_data_t
- struct system_data_t

Typedefs

- typedef struct sensor_data_t sensor_data_t
- typedef struct actuator_data_t actuator_data_t
- typedef struct system_data_t system_data_t

4.2.1 Typedef Documentation

4.2.1.1 typedef struct actuator_data_t actuator_data_t

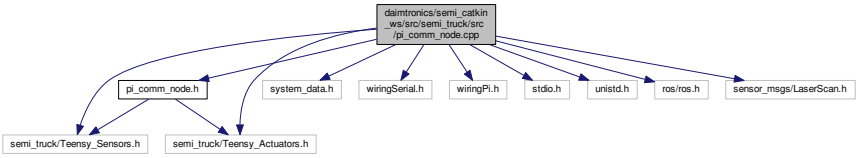
4.2.1.2 typedef struct sensor_data_t sensor_data_t

4.2.1.3 typedef struct system_data_t system_data_t

4.3 daimtronics/semi_catkin_ws/src/semi_truck/src/pi_comm_node.cpp File Reference

```
#include "pi_comm_node.h"
#include "system_data.h"
#include "semi_truck/Teensy_Sensors.h"
#include "semi_truck/Teensy_Actuators.h"
#include <wiringSerial.h>
#include <wiringPi.h>
#include <stdio.h>
#include <unistd.h>
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
```

Include dependency graph for pi_comm_node.cpp:



Macros

- #define SHORT_SIZE 2
- #define RELAY_PIN_1 7
- #define RELAY_PIN_2 0
- #define SYNC_VALUE -32000
- #define SENSOR_DATA_SIZE_W_SYNC 14
- #define SENSOR_DATA_SIZE 12
- #define UART "/dev/ttyS0"
- #define BAUDRATE 9600
- #define LOOP_FREQUENCY 20

Functions

- int main (int argc, char **argv)

  The main function for the ROS node to communicate with the Teensy over UART. It receives sensor data from the Teensy and publishes this data to the teensy_sensor_data topic. It also subscribes to the teensy_actuator_data topic and writes the values it gets to the Teensy over UART.
- void pi_sync ()

  Called before reading sensor data from the Teensy. It will read the buffer until it encounters the SYNC_VALUE that has been defined. Once this happens, the next set of bytes are the set of sensor values.
- short read_sensor_msg (int serial, char num_bytes)

  Reads a single value from the UART communication buffer.
- void read_from_teensy (int serial, semi_truck::Teensy_Sensors &sensors)

  Reads an entire set of Teensy sensor data by calling read_sensor_msg on each sensor data in the UART message.
- void write_actuator_msg (int serial, short actuator_val, char num_bytes)

  Writes a single actuator value to the Teensy.

- void write_to_teensy (int serial, const semi_truck::Teensy_Actuators &actuators)

  Writes an entire set of actuator data to the Teensy via UART.
- void print_sensors (const semi_truck::Teensy_Sensors &sensors)

  A function useful for debugging serial communication. Prints the entire set of sensor data to the console.
- void print_actuators (const semi_truck::Teensy_Actuators &actuators)

  A function useful for debugging serial communication. Prints the entire set of actuator data to the console.
- void actuator_cb (const semi_truck::Teensy_Actuators &msg)

  The callback function the the subscriber to the teensy_actuator_data topic. This function will run every time ROS::spinOnce is called. It reads the set of data from the topic and immediately writes these values to the Teensy via UART.

Variables

- static int serial

4.3.1 Macro Definition Documentation

4.3.1.1 #define BAUDRATE 9600

4.3.1.2 #define LOOP_FREQUENCY 20

4.3.1.3 #define RELAY_PIN_1 7

4.3.1.4 #define RELAY_PIN_2 0

4.3.1.5 #define SENSOR_DATA_SIZE 12

4.3.1.6 #define SENSOR_DATA_SIZE_W_SYNC 14

4.3.1.7 #define SHORT_SIZE 2

4.3.1.8 #define SYNC_VALUE -32000

4.3.1.9 #define UART "/dev/ttyS0"

4.3.2 Function Documentation

4.3.2.1 void actuator_cb (const semi_truck::Teensy_Actuators & *msg*)

The callback function the the subscriber to the teensy_actuator_data topic. This function will run every time ROS::spinOnce is called. It reads the set of data from the topic and immediately writes these values to the Teensy via UART.

Parameters

msg	A set of actuator data that has come from the actuator topic and needs to be written to the Teensy.

4.3.2.2 int main (int *argc,* char ** *argv*)

The main function for the ROS node to communicate with the Teensy over UART. It receives sensor data from the Teensy and publishes this data to the teensy_sensor_data topic. It also subscribes to the teensy_actuator_data topic and writes the values it gets to the Teensy over UART.

4.3.2.3 void pi_sync ( )

Called before reading sensor data from the Teensy. It will read the buffer until it encounters the SYNC_VALUE that has been defined. Once this happens, the next set of bytes are the set of sensor values.

4.3.2.4 void print_actuators (const semi_truck::Teensy_Actuators & *actuators*)

A function useful for debugging serial communication. Prints the entire set of actuator data to the console.

Parameters

actuators	The object that holds all of the actuator data

4.3.2.5 void print_sensors (const semi_truck::Teensy_Sensors & *sensors*)

A function useful for debugging serial communication. Prints the entire set of sensor data to the console.

Parameters

sensors	The object that holds all of the sensor data

4.3.2.6 void read_from_teensy (int *serial,* semi_truck::Teensy_Sensors & *sensors*)

Reads an entire set of Teensy sensor data by calling read_sensor_msg on each sensor data in the UART message.

4.3.2.7 short read_sensor_msg (int *serial,* char *num_bytes*)

Reads a single value from the UART communication buffer.

Parameters

serial	The serial file descriptor to read from
num_bytes	The number of bytes to read from

Returns

The value of the bytes that have been read

4.3.2.8 void write_actuator_msg (int *serial*, short *actuator_val*, char *num_bytes*)

Writes a single actuator value to the Teensy.

Parameters

serial	The serial file descriptor to write to
actuator_val	The value that is written through UART to the Teensy
num_bytes	Number of bytes to write to the UART

4.3.2.9 void write_to_teensy (int *serial*, const semi_truck::Teensy_Actuators & *actuators*)

Writes an entire set of actuator data to the Teensy via UART.

Parameters

serial	The serial file descriptor to write to
actuators	The object that holds all of the actuator data

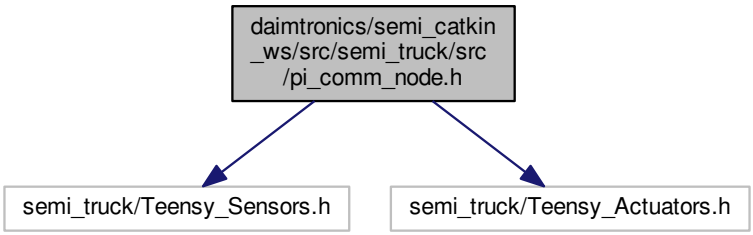
4.3.3 Variable Documentation

4.3.3.1 int serial [static]

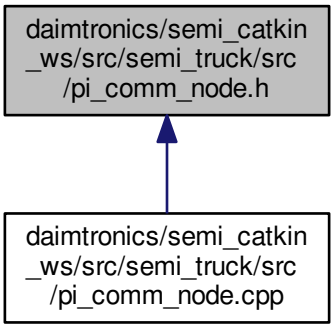
4.4 daimtronics/semi_catkin_ws/src/semi_truck/src/pi_comm_node.h File Reference

```
#include "semi_truck/Teensy_Sensors.h"
#include "semi_truck/Teensy_Actuators.h"
```

Include dependency graph for pi_comm_node.h:



This graph shows which files directly or indirectly include this file:



Functions

- short read_sensor_msg (int serial, char num_bytes)

  Reads a single value from the UART communication buffer.
- void read_from_teensy (int serial, semi_truck::Teensy_Sensors &sensors)

  Reads an entire set of Teensy sensor data by calling read_sensor_msg on each sensor data in the UART message.
- void write_sensor_msg (int serial, short sensor_val, char num_bytes)
- void write_to_teensy (int serial, const semi_truck::Teensy_Actuators &actuators)

  Writes an entire set of actuator data to the Teensy via UART.
- void update_sensors (semi_truck::Teensy_Sensors &sensors)
- void pi_sync ()

  Called before reading sensor data from the Teensy. It will read the buffer until it encounters the SYNC_VALUE that has been defined. Once this happens, the next set of bytes are the set of sensor values.
- void print_sensors (const semi_truck::Teensy_Sensors &sensors)

  A function useful for debugging serial communication. Prints the entire set of sensor data to the console.
- void print_actuators (const semi_truck::Teensy_Actuators &actuators)

  A function useful for debugging serial communication. Prints the entire set of actuator data to the console.
- void sensor_cb (const semi_truck::Teensy_Sensors &msg)
- void actuator_cb (const semi_truck::Teensy_Actuators &msg)

  The callback function the the subscriber to the teensy_actuator_data topic. This function will run every time ROS↩S::spinOnce is called. It reads the set of data from the topic and immediately writes these values to the Teensy via UART.

4.4.1 Function Documentation

4.4.1.1 void actuator_cb (const semi_truck::Teensy_Actuators & *msg*)

The callback function the the subscriber to the teensy_actuator_data topic. This function will run every time ROS↩::spinOnce is called. It reads the set of data from the topic and immediately writes these values to the Teensy via UART.

Parameters

msg	A set of actuator data that has come from the actuator topic and needs to be written to the Teensy.

4.4.1.2 void pi_sync ( )

Called before reading sensor data from the Teensy. It will read the buffer until it encounters the SYNC_VALUE that has been defined. Once this happens, the next set of bytes are the set of sensor values.

4.4.1.3 void print_actuators (const semi_truck::Teensy_Actuators & *actuators*)

A function useful for debugging serial communication. Prints the entire set of actuator data to the console.

Parameters

actuators	The object that holds all of the actuator data

4.4.1.4 void print_sensors (const semi_truck::Teensy_Sensors & *sensors*)

A function useful for debugging serial communication. Prints the entire set of sensor data to the console.

Parameters

sensors	The object that holds all of the sensor data

4.4.1.5 void read_from_teensy (int *serial,* semi_truck::Teensy_Sensors & *sensors*)

Reads an entire set of Teensy sensor data by calling read_sensor_msg on each sensor data in the UART message.

4.4.1.6 short read_sensor_msg (int *serial,* char *num_bytes*)

Reads a single value from the UART communication buffer.

Parameters

serial	The serial file descriptor to read from
num_bytes	The number of bytes to read from

Returns

The value of the bytes that have been read

4.4.1.7 void sensor_cb (const semi_truck::Teensy_Sensors & *msg*)

4.4.1.8 void update_sensors (semi_truck::Teensy_Sensors & *sensors*)

4.4.1.9 void write_sensor_msg (int *serial,* short *sensor_val,* char *num_bytes*)

4.4.1.10 void write_to_teensy (int *serial,* const semi_truck::Teensy_Actuators & *actuators*)

Writes an entire set of actuator data to the Teensy via UART.

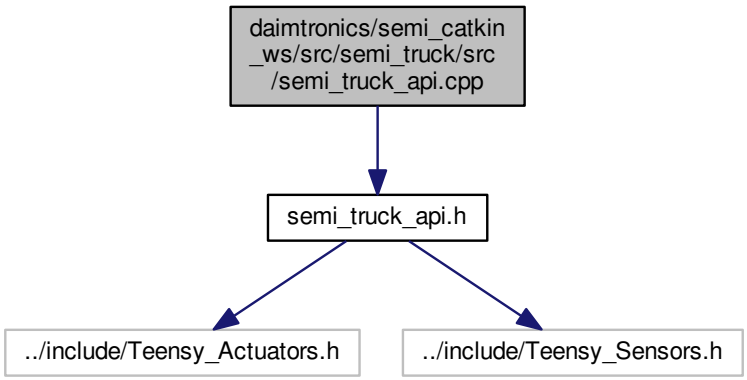
Parameters

serial	The serial file descriptor to write to
actuators	The object that holds all of the actuator data

4.5 daimtronics/semi_catkin_ws/src/semi_truck/src/semi_truck_api.cpp File Reference

```
#include "semi_truck_api.h"
```

Include dependency graph for semi_truck_api.cpp:



Functions

- void set_motor_output (semi_truck::Teensy_Actuators &actuators, int16_t motor_output)

  sets the motor output of the actuators object to the passed in value
- void set_steer_output (semi_truck::Teensy_Actuators &actuators, int16_t steer_output)

  sets the steer output of the actuators object to the passed in value
- void set_fifth_output (semi_truck::Teensy_Actuators &actuators, uint16_t fifth_output)

  sets the fifth wheel of the actuators object to the passed in value
- int16_t get_wheel_speed (semi_truck::Teensy_Sensors &sensors)

reads the wheel speed of a TeensySensors object

- int16_t get_imu_angle (semi_truck::Teensy_Sensors &sensors)

  reads the imu angle (degrees) of a TeensySensors object
- int16_t get_right_TOF (semi_truck::Teensy_Sensors &sensors)

  reads the right TOF distance (cm) of a TeensySensors object
- int16_t get_left_TOF (semi_truck::Teensy_Sensors &sensors)

  reads the left TOF distance (cm) of a TeensySensors object
- int16_t get_rear_TOF (semi_truck::Teensy_Sensors &sensors)

  reads the rear TOF distance (cm) of a TeensySensors object

4.5.1 Function Documentation

4.5.1.1 int16_t get_imu_angle (semi_truck::Teensy_Sensors & *sensors*)

reads the imu angle (degrees) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.5.1.2 int16_t get_left_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the left TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.5.1.3 int16_t get_rear_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the rear TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.5.1.4 int16_t get_right_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the right TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.5.1.5 int16_t get_wheel_speed (semi_truck::Teensy_Sensors & *sensors*)

reads the wheel speed of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.5.1.6 void set_fifth_output (semi_truck::Teensy_Actuators & *actuators,* uint16_t *value*)

sets the fifth wheel of the actuators object to the passed in value

Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. 0 for locked and 1 for unlocked.

4.5.1.7 void set_motor_output (semi_truck::Teensy_Actuators & *actuators,* int16_t *value*)

sets the motor output of the actuators object to the passed in value

Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. This value should range from 0 for full reverse power to 180 for full forwards power.

4.5.1.8 void set_steer_output (semi_truck::Teensy_Actuators & *actuators,* int16_t *value*)

sets the steer output of the actuators object to the passed in value

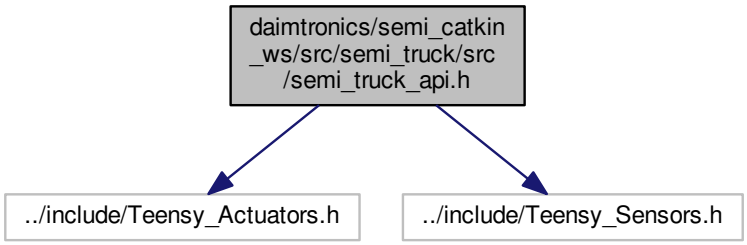
Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. This value should range from 0 for a 20 degree angle left to 180 for a 20 degree angle right.

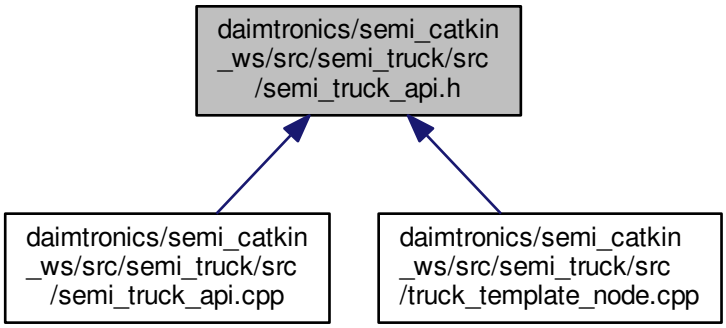
4.6 daimtronics/semi_catkin_ws/src/semi_truck/src/semi_truck_api.h File Reference

```
#include "../include/Teensy_Actuators.h"
#include "../include/Teensy_Sensors.h"
```

Include dependency graph for semi_truck_api.h:



This graph shows which files directly or indirectly include this file:



Functions

- void set_motor_output (semi_truck::Teensy_Actuators &actuators, int16_t value)
  sets the motor output of the actuators object to the passed in value
- void set_steer_output (semi_truck::Teensy_Actuators &actuators, int16_t value)
  sets the steer output of the actuators object to the passed in value
- void set_fifth_output (semi_truck::Teensy_Actuators &actuators, uint16_t value)
  sets the fifth wheel of the actuators object to the passed in value
- int16_t get_wheel_speed (semi_truck::Teensy_Sensors &sensors)
  reads the wheel speed of a TeensySensors object
- int16_t get_imu_angle (semi_truck::Teensy_Sensors &sensors)
  reads the imu angle (degrees) of a TeensySensors object
- int16_t get_right_TOF (semi_truck::Teensy_Sensors &sensors)
  reads the right TOF distance (cm) of a TeensySensors object
- int16_t get_left_TOF (semi_truck::Teensy_Sensors &sensors)
  reads the left TOF distance (cm) of a TeensySensors object
- int16_t get_rear_TOF (semi_truck::Teensy_Sensors &sensors)
  reads the rear TOF distance (cm) of a TeensySensors object

4.6.1 Function Documentation

4.6.1.1 int16_t get_imu_angle (semi_truck::Teensy_Sensors & *sensors*)

reads the imu angle (degrees) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.6.1.2 int16_t get_left_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the left TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.6.1.3 int16_t get_rear_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the rear TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.6.1.4 int16_t get_right_TOF (semi_truck::Teensy_Sensors & *sensors*)

reads the right TOF distance (cm) of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.6.1.5 int16_t get_wheel_speed (semi_truck::Teensy_Sensors & *sensors*)

reads the wheel speed of a TeensySensors object

Parameters

sensors	a reference to a TeensySensors object to read from.

4.6.1.6 void set_fifth_output (semi_truck::Teensy_Actuators & *actuators,* uint16_t *value*)

sets the fifth wheel of the actuators object to the passed in value

Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. 0 for locked and 1 for unlocked.

4.6.1.7 void set_motor_output (semi_truck::Teensy_Actuators & *actuators,* int16_t *value*)

sets the motor output of the actuators object to the passed in value

Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. This value should range from 0 for full reverse power to 180 for full forwards power.

4.6.1.8 void set_steer_output (semi_truck::Teensy_Actuators & *actuators,* int16_t *value*)

sets the steer output of the actuators object to the passed in value

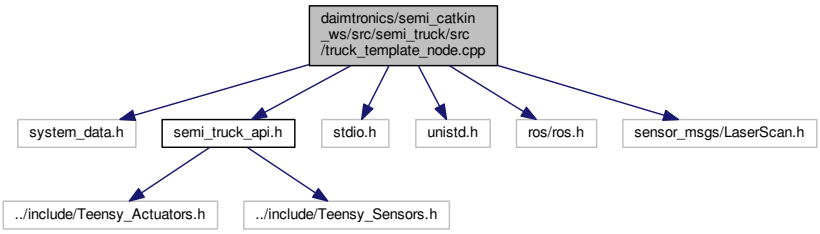
Parameters

actuators	a reference to a TeensyActuators object to alter
value	the value to update the actuators with. This value should range from 0 for a 20 degree angle left to 180 for a 20 degree angle right.

4.7 daimtronics/semi_catkin_ws/src/semi_truck/src/truck_template_node.cpp File Reference

```
#include "system_data.h"
#include "semi_truck_api.h"
#include <stdio.h>
#include <unistd.h>
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
```

Include dependency graph for truck_template_node.cpp:



Functions

- void rplidar_cb (const sensor_msgs::LaserScan &msg)
- void lidar_lite_cb (const sensor_msgs::LaserScan &msg)
- void teensy_sensors_cb (const semi_truck::Teensy_Sensors &msg)
- int main (int argc, char **argv)

4.7.1 Function Documentation

4.7.1.1 void lidar_lite_cb (const sensor_msgs::LaserScan & *msg*)

4.7.1.2 int main (int *argc,* char ** *argv*)

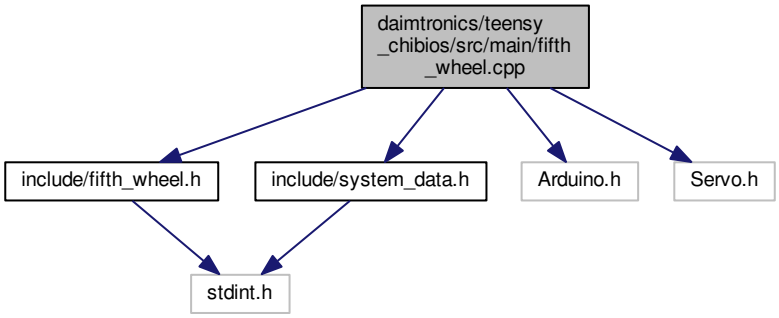
4.7.1.3 void rplidar_cb (const sensor_msgs::LaserScan & *msg*)

4.7.1.4 void teensy_sensors_cb (const semi_truck::Teensy_Sensors & *msg*)

4.8 daimtronics/teensy_chibios/src/main/fifth_wheel.cpp File Reference

```
#include "include/fifth_wheel.h"
#include "include/system_data.h"
#include <Arduino.h>
#include <Servo.h>
```

Include dependency graph for fifth_wheel.cpp:



Macros

- #define LOCKED_ANGLE 0
- #define UNLOCKED_ANGLE 180

Functions

- void fifth_wheel_loop_fn (int16_t fifth_output)
- void fifth_wheel_setup (short fifth_wheel_pin)

  Set up the fifth wheel task to write to the pin attached to the fifth wheel, and to be in the locked position.

Variables

- static Servo fifth_wheel_servo

  This is a Servo object to control the fifth wheel. It relies on code from Servo.h which is a prebuilt library for the Arduino IDE.

4.8.1 Macro Definition Documentation

4.8.1.1 #define LOCKED_ANGLE 0

4.8.1.2 #define UNLOCKED_ANGLE 180

4.8.2 Function Documentation

4.8.2.1 void fifth_wheel_loop_fn (int16_t *fifth_output*)

The is the primary function controlling the fifth wheel. It reads the fifth_output value form the system data and writes to the Servo for actuating between the two different angles.

Parameters

fifth_output	the output to the fifth wheel, which will be one of two values, either locked or unlocked

4.8.2.2 void fifth_wheel_setup (short *fifth_wheel_pin*)

Set up the fifth wheel task to write to the pin attached to the fifth wheel, and to be in the locked position.

Parameters

fifth_wheel_pin	The pin that signals a PWM to the fifth wheel servo.

4.8.3 Variable Documentation

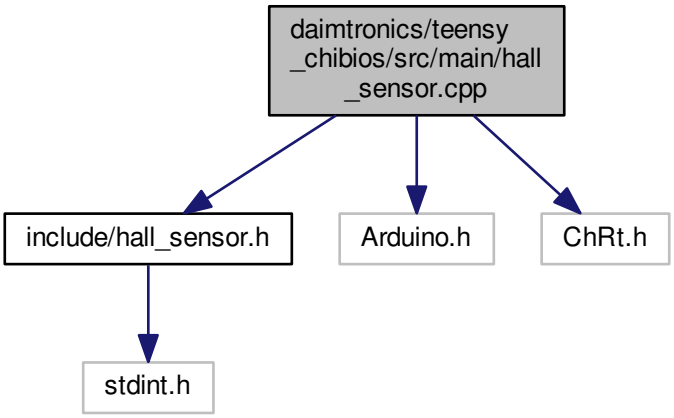
4.8.3.1 Servo fifth_wheel_servo [static]

This is a Servo object to control the fifth wheel. It relies on code from Servo.h which is a prebuilt library for the Arduino IDE.

4.9 daimtronics/teensy_chibios/src/main/hall_sensor.cpp File Reference

```
#include "include/hall_sensor.h"
#include <Arduino.h>
#include <ChRt.h>
```

Include dependency graph for hall_sensor.cpp:



Functions

- int16_t hall_sensor_loop_fn (short PhaseB_pin, short PhaseC_pin)

  This is the primary function controlling the wheel speed sensor. It reads a Hall sensor that is built into the Tekin R8 Motor. This is a three phased motor (PhaseA, PhaseB, PhaseC). Each phase is offset 120 degrees from each other. Ticks will be incremented if the motor rotates forwards one full revolution, and decremented if it rotates backwards one revolution.

- void hall_sensor_setup (short PhaseA_pin, short PhaseB_pin, short PhaseC_pin)

  Set up the wheel speed task to read high or low values from the three pins attached to the Hall sensor (one for each phase).

Variables

- int16_t ticks = 0

4.9.1 Function Documentation

4.9.1.1 int16_t hall_sensor_loop_fn (short *PhaseB_pin,* short *PhaseC_pin*)

This is the primary function controlling the wheel speed sensor. It reads a Hall sensor that is built into the Tekin R8 Motor. This is a three phased motor (PhaseA, PhaseB, PhaseC). Each phase is offset 120 degrees from each other. Ticks will be incremented if the motor rotates forwards one full revolution, and decremented if it rotates backwards one revolution.

Returns

the current number of ticks that the sensor reads.

4.9.1.2 void hall_sensor_setup (short *PhaseA_pin,* short *PhaseB_pin,* short *PhaseC_pin*)

Set up the wheel speed task to read high or low values from the three pins attached to the Hall sensor (one for each phase).

Parameters

PhaseA_pin	An interrupt is triggered every time the Teensy read a leading edge for this phase.
PhaseB_pin	If this pin is high when the interrupt is triggered, the truck is going in reverse.
PhaseC_pin	If this pin is high when the interrupt is triggered, the truck is going forwards.

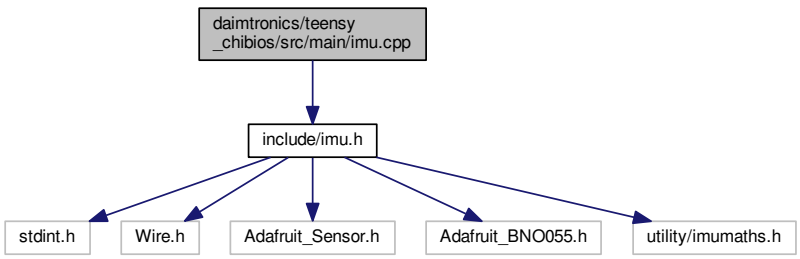
4.9.2 Variable Documentation

4.9.2.1 int16_t ticks = 0

4.10 daimtronics/teensy_chibios/src/main/imu.cpp File Reference

```
#include "include/imu.h"
```

Include dependency graph for imu.cpp:



Functions

- int16_t imu_loop_fn ()

  The primary function for the IMU that reads and returns heading data of the BNO055.The Adafruit_BNO055 library does most of the work here. This function simply sets up a sets up an "event" variable to hold the IMU data.
- void imu_setup ()

  Initializes the BNO055 sensor.
- void print_imu_data (sensors_event_t *event)

  A debugging function used to print out IMU orientation to the USB serial device (usually the Arduino Serial Monitor).

Variables

- Adafruit_BNO055 bno = Adafruit_BNO055(55)

  A global variable, for the only Adafruit_BNO055 object in the system.

4.10.1 Function Documentation

4.10.1.1 int16_t imu_loop_fn ( )

The primary function for the IMU that reads and returns heading data of the BNO055.The Adafruit_BNO055 library does most of the work here. This function simply sets up a sets up an "event" variable to hold the IMU data.

Returns

an integer representing the heading angle in degrees

4.10.1.2 void imu_setup ( )

Initializes the BNO055 sensor.

4.10.1.3 void print_imu_data (sensors_event_t * *event*)

A debugging function used to print out IMU orientation to the USB serial device (usually the Arduino Serial Monitor).

4.10.2 Variable Documentation

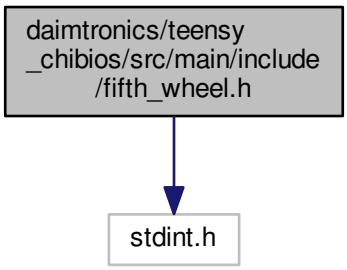
4.10.2.1 Adafruit_BNO055 bno = Adafruit_BNO055(55)

A global variable, for the only Adafruit_BNO055 object in the system.

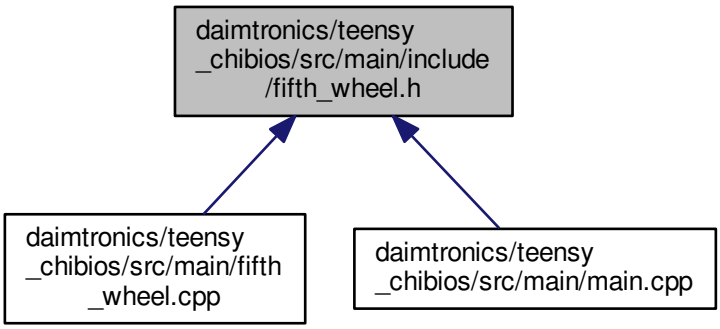
4.11 daimtronics/teensy_chibios/src/main/include/fifth_wheel.h File Reference

```
#include <stdint.h>
```

Include dependency graph for fifth_wheel.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define LOCKED 1
- #define UNLOCKED 0

Functions

- void fifth_wheel_loop_fn (int16_t fifth_output)
- void fifth_wheel_setup (short fifth_wheel_pin)

  Set up the fifth wheel task to write to the pin attached to the fifth wheel, and to be in the locked position.

4.11.1 Macro Definition Documentation

4.11.1.1 #define LOCKED 1

4.11.1.2 #define UNLOCKED 0

4.11.2 Function Documentation

4.11.2.1 void fifth_wheel_loop_fn (int16_t *fifth_output*)

The is the primary function controlling the fifth wheel. It reads the fifth_output value form the system data and writes to the Servo for actuating between the two different angles.

Parameters

fifth_output	the output to the fifth wheel, which will be one of two values, either locked or unlocked

4.11.2.2 void fifth_wheel_setup (short *fifth_wheel_pin*)

Set up the fifth wheel task to write to the pin attached to the fifth wheel, and to be in the locked position.

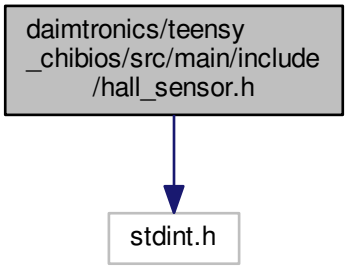
Parameters

fifth_wheel_pin	The pin that signals a PWM to the fifth wheel servo.

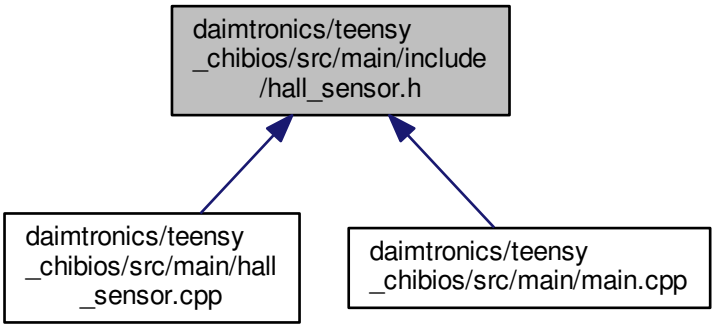
4.12 daimtronics/teensy_chibios/src/main/include/hall_sensor.h File Reference

```
#include <stdint.h>
```

Include dependency graph for hall_sensor.h:



This graph shows which files directly or indirectly include this file:



Functions

- int16_t hall_sensor_loop_fn (short PhaseB_pin, short PhaseC_pin)

  This is the primary function controlling the wheel speed sensor. It reads a Hall sensor that is built into the Tekin R8 Motor. This is a three phased motor (PhaseA, PhaseB, PhaseC). Each phase is offset 120 degrees from each other. Ticks will be incremented if the motor rotates forwards one full revolution, and decremented if it rotates backwards one revolution.
- void hall_sensor_setup (short PhaseA_pin, short PhaseB_pin, short PhaseC_pin)

  Set up the wheel speed task to read high or low values from the three pins attached to the Hall sensor (one for each phase).

4.12.1 Function Documentation

4.12.1.1 int16_t hall_sensor_loop_fn (short *PhaseB_pin,* short *PhaseC_pin*)

This is the primary function controlling the wheel speed sensor. It reads a Hall sensor that is built into the Tekin R8 Motor. This is a three phased motor (PhaseA, PhaseB, PhaseC). Each phase is offset 120 degrees from each other. Ticks will be incremented if the motor rotates forwards one full revolution, and decremented if it rotates backwards one revolution.

Returns

the current number of ticks that the sensor reads.

4.12.1.2 void hall_sensor_setup (short *PhaseA_pin,* short *PhaseB_pin,* short *PhaseC_pin*)

Set up the wheel speed task to read high or low values from the three pins attached to the Hall sensor (one for each phase).

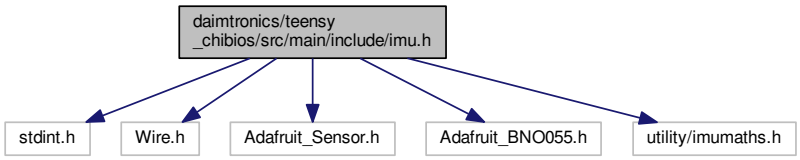
Parameters

PhaseA_pin	An interrupt is triggered every time the Teensy read a leading edge for this phase.
PhaseB_pin	If this pin is high when the interrupt is triggered, the truck is going in reverse.
PhaseC_pin	If this pin is high when the interrupt is triggered, the truck is going forwards.

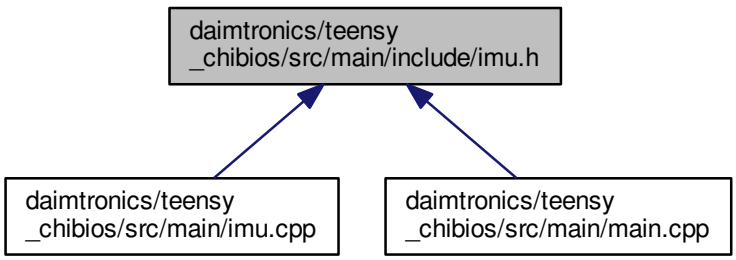
4.13 daimtronics/teensy_chibios/src/main/include/imu.h File Reference

```
#include <stdint.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
```

Include dependency graph for imu.h:



This graph shows which files directly or indirectly include this file:



Functions

- short imu_loop_fn ()

  The primary function for the IMU that reads and returns heading data of the BNO055.The Adafruit_BNO055 library does most of the work here. This function simply sets up a sets up an "event" variable to hold the IMU data.
- void imu_setup ()

  Initializes the BNO055 sensor.
- void print_imu_data (sensors_event_t ∗event)

  A debugging function used to print out IMU orientation to the USB serial device (usually the Arduino Serial Monitor).

4.13.1 Function Documentation

4.13.1.1 short imu_loop_fn ( )

The primary function for the IMU that reads and returns heading data of the BNO055.The Adafruit_BNO055 library does most of the work here. This function simply sets up a sets up an "event" variable to hold the IMU data.

Returns

an integer representing the heading angle in degrees

4.13.1.2 void imu_setup ( )

Initializes the BNO055 sensor.

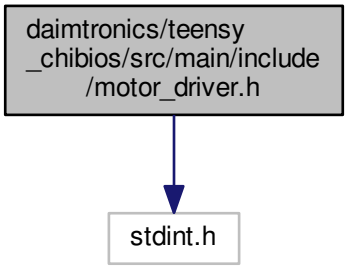
4.13.1.3 void print_imu_data (sensors_event_t * *event*)

A debugging function used to print out IMU orientation to the USB serial device (usually the Arduino Serial Monitor).

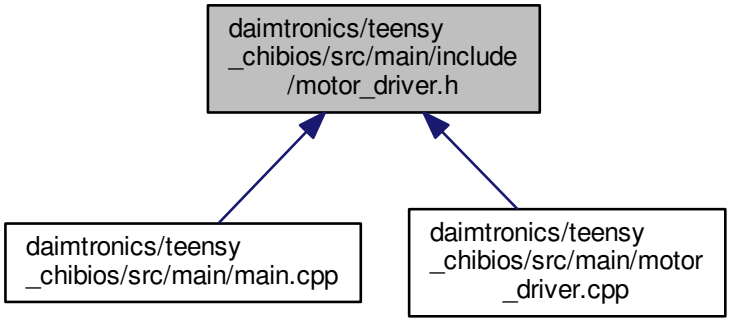
4.14 daimtronics/teensy_chibios/src/main/include/motor_driver.h File Reference

```
#include <stdint.h>
```

Include dependency graph for motor_driver.h:



This graph shows which files directly or indirectly include this file:



Functions

- void motor_driver_loop_fn (int16_t motor_output)

  This is the primary function controlling the motor. It reads the motor output value from the system data and controls the motor with this value.
- int16_t scale_output (int16_t motor_output)

  Scales a value coming from the pi between -100 and 100 to between 0 and 180 for what the Servo library requires.
- void motor_driver_setup (short motor_pin)

  Set up the motor driver task to write to the pin attached to the motor, and to be in the locked position. It also outputs a value corresponding to zero torque.
- int16_t stop_motor (int16_t wheel_speed, int16_t time_step)

  Runs a control loop to stop the motor based on the reported wheel speed, and returns a value to be output to the motor.

4.14.1 Function Documentation

4.14.1.1 void motor_driver_loop_fn (int16_t *motor_output*)

This is the primary function controlling the motor. It reads the motor output value from the system data and controls the motor with this value.

Parameters

motor_output	the output to the motor

4.14.1.2 void motor_driver_setup (short *motor_pin*)

Set up the motor driver task to write to the pin attached to the motor, and to be in the locked position. It also outputs a value corresponding to zero torque.

Parameters

motor_pin	The pin sends a PWM signal to the motor.

4.14.1.3 int16_t scale_output (int16_t *motor_output*)

Scales a value coming from the pi between -100 and 100 to between 0 and 180 for what the Servo library requires.

Parameters

motor_output	An input value, ideally between -100 and 100 (although it will be limited to -100 or 100 if outside this range

Returns

a value to output to the motor between 0 and 180

4.14.1.4 int16_t stop_motor (int16_t *wheel_speed,* int16_t *time_step*)

Runs a control loop to stop the motor based on the reported wheel speed, and returns a value to be output to the motor.

Parameters

wheel_speed	speed of the truck read by the wheel speed sensor
time_step	number of millis since the last time this task ran; used in integral control

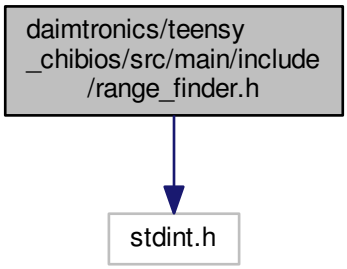
Returns

the output to the motor

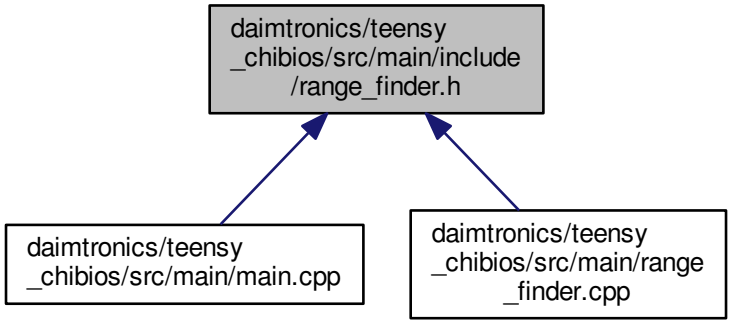
4.15 daimtronics/teensy_chibios/src/main/include/range_finder.h File Reference

```
#include <stdint.h>
```

Include dependency graph for range_finder.h:



This graph shows which files directly or indirectly include this file:



Functions

- long range_finder_loop_fn (short urf_echo_pin)
- void range_finder_ping (short urf_trig_pin)
- void range_finder_setup (short urf_trig_pin)

4.15.1 Function Documentation

4.15.1.1 long range_finder_loop_fn (short *urf_echo_pin*)

This is the primary function controlling the URFs. It reads a value representing a distance to an object from the URF sensor and returns that value

Returns

distance to object

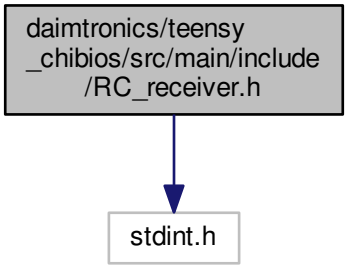
4.15.1.2 void range_finder_ping (short *urf_trig_pin*)

4.15.1.3 void range_finder_setup (short *urf_trig_pin*)

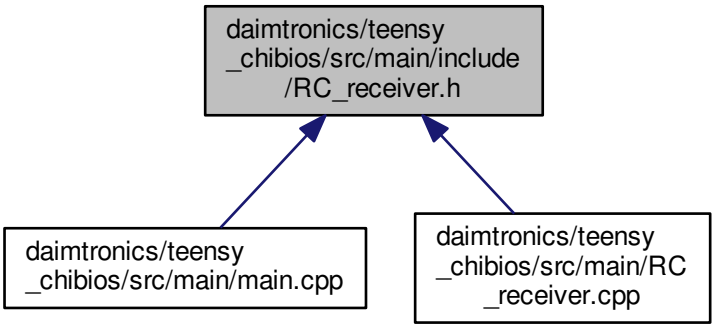
4.16 daimtronics/teensy_chibios/src/main/include/RC_receiver.h File Reference

```
#include <stdint.h>
```

Include dependency graph for RC_receiver.h:



This graph shows which files directly or indirectly include this file:



Functions

- int16_t RC_receiver_SW1_fn (short PWM_PIN)

  This is the primary function reading Switch 1 on the receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a deadman mode (either deadman switch pressed or not pressed) is selected to send to the rest of the platform.
- int16_t RC_receiver_SW2_fn (short PWM_PIN)

- int16_t RC_receiver_SW3_fn (short PWM_PIN)

  This is the primary function controlling the RC receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a drive mode (either manual or one of the autonomous algorithms on the Pi) is selected to control the vehicle.
- void RC_receiver_setup ()

4.16.1 Function Documentation

4.16.1.1 void RC_receiver_setup ( )

4.16.1.2 int16_t RC_receiver_SW1_fn (short *PWM_PIN*)

This is the primary function reading Switch 1 on the receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a deadman mode (either deadman switch pressed or not pressed) is selected to send to the rest of the platform.

Returns

the deadman mode of the semi-truck based on RC receiver signal

4.16.1.3 int16_t RC_receiver_SW2_fn (short *PWM_PIN*)

4.16.1.4 int16_t RC_receiver_SW3_fn (short *PWM_PIN*)

This is the primary function controlling the RC receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a drive mode (either manual or one of the autonomous algorithms on the Pi) is selected to control the vehicle.

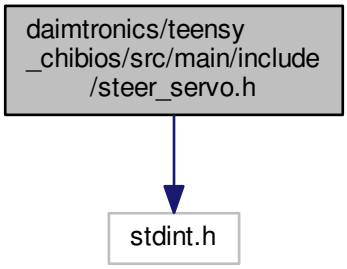
Returns

the driving mode of the semi-truck based on RC receiver signal

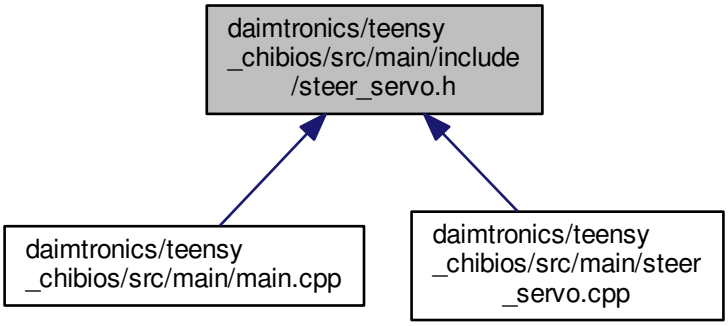
4.17 daimtronics/teensy_chibios/src/main/include/steer_servo.h File Reference

```
#include <stdint.h>
```

Include dependency graph for steer_servo.h:



This graph shows which files directly or indirectly include this file:



Functions

- void steer_servo_loop_fn (int16_t steer_output)
- void steer_servo_setup (short servo_pin)

  Set up the steer servo task to write to the pin attached to the servo controlling angle of the trucks's front axis.

4.17.1 Function Documentation

4.17.1.1 void steer_servo_loop_fn (int16_t *steer_output*)

The is the primary function controlling the steering servo wheel. It reads a value form the system data and controls the steering servo based on what the system data contains.

Parameters

steer_output	the output to the steering servo that controls angle

4.17.1.2 void steer_servo_setup (short *servo_pin*)

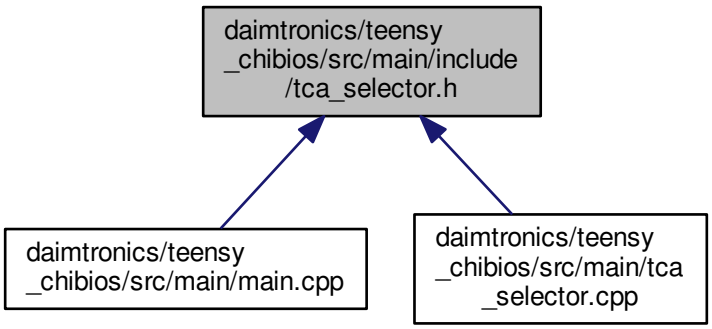
Set up the steer servo task to write to the pin attached to the servo controlling angle of the trucks's front axis.

Parameters

servo_pin	The pin that signals a PWM to the steering servo.

4.18 daimtronics/teensy_chibios/src/main/include/tca_selector.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void tcaselect (short i)

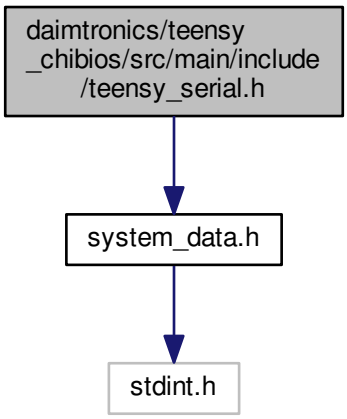
4.18.1 Function Documentation

4.18.1.1 void tcaselect (short *i*)

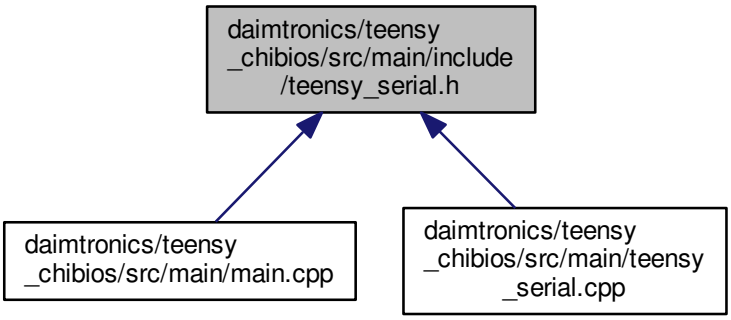
4.19 daimtronics/teensy_chibios/src/main/include/teensy_serial.h File Reference

```
#include "system_data.h"
```

Include dependency graph for teensy_serial.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define HWSERIAL Serial1

Functions

- void teensy_serial_loop_fn (system_data_t *system_data)

  The primary function for communicating between the Teensy and the Pi over the Serial UART port.
- void teensy_serial_setup ()

  Sets up the serial communication for the teensy to output data to both the Pi (through UART) and a PC console (through USB).
- void set_sensor_msg (int user_input, sensor_data_t *data_ptr)
- void teensy_sync ()

  Ensures that the data being read from the Pi is synced. It clears the serial buffer up until it reads the designated SYNC_VALUE.
- void clear_buffer ()

  Clears the serial buffer from all of its data. This is used primarily when the Teensy is restarted, and the Pi has been sending data that the Teensy does not need to process.
- void read_from_pi (actuator_data_t *actuators_ptr)
- void print_sensor_msg (sensor_data_t *sensors_ptr)
- void print_actuator_msg (actuator_data_t *actuators_ptr)

4.19.1 Macro Definition Documentation

4.19.1.1 #define HWSERIAL Serial1

4.19.2 Function Documentation

4.19.2.1 void clear_buffer ( )

Clears the serial buffer from all of its data. This is used primarily when the Teensy is restarted, and the Pi has been sending data that the Teensy does not need to process.

4.19.2.2 void print_actuator_msg (actuator_data_t * *actuators_ptr*)

4.19.2.3 void print_sensor_msg (sensor_data_t * *sensors_ptr*)

4.19.2.4 void read_from_pi (actuator_data_t * *actuators_ptr*)

4.19.2.5 void set_sensor_msg (int *user_input,* sensor_data_t * *data_ptr*)

4.19.2.6 void teensy_serial_loop_fn (system_data_t * *system_data*)

The primary function for communicating between the Teensy and the Pi over the Serial UART port.

Parameters

system_data	a pointer to the system data that is declared statically in main.ino.

4.19.2.7 void teensy_serial_setup ( )

Sets up the serial communication for the teensy to output data to both the Pi (through UART) and a PC console (through USB).

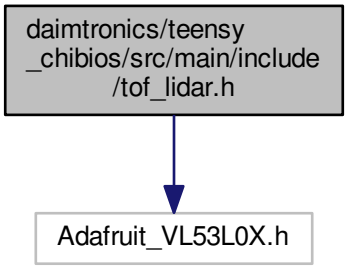
4.19.2.8 void teensy_sync ( )

Ensures that the data being read from the Pi is synced. It clears the serial buffer up until it reads the designated SYNC_VALUE.

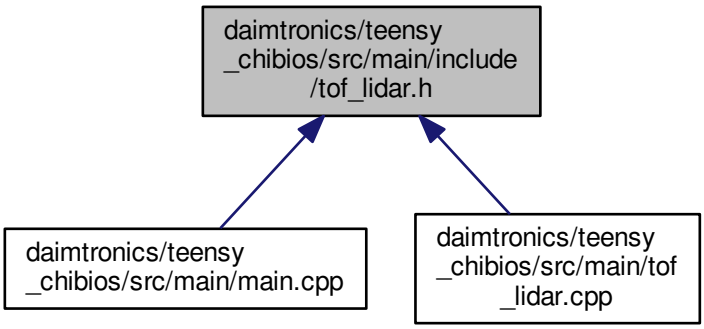
4.20 daimtronics/teensy_chibios/src/main/include/tof_lidar.h File Reference

```
#include <Adafruit_VL53L0X.h>
```

Include dependency graph for tof_lidar.h:



This graph shows which files directly or indirectly include this file:



Functions

- int16_t tof_loop_fn ()

  This is the primary function controlling the ToF Lidar for reading distance measurements on the sensors.

- void tof_lidar_setup ()

  Initializes the VL53L0X sensor.

4.20.1 Function Documentation

4.20.1.1 void tof_lidar_setup ( )

Initializes the VL53L0X sensor.

4.20.1.2 int16_t tof_loop_fn ( )

This is the primary function controlling the ToF Lidar for reading distance measurements on the sensors.

The Adafruit_VL53L0X library does most of the work and the function here calls the measure.RangeMilliMeter instruction and stores the distance here.

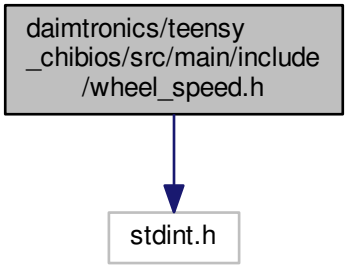
Returns

an integer representing distance the sensor detected in millimeters

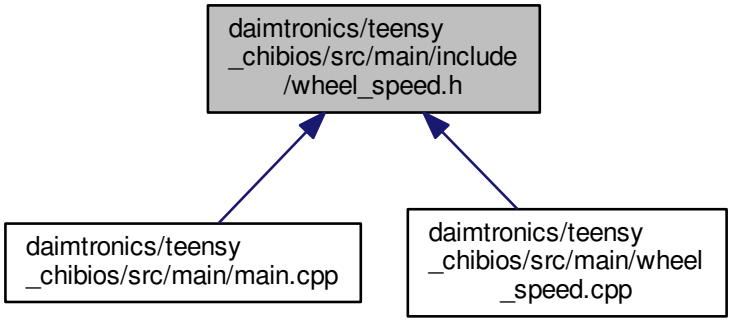
4.21 daimtronics/teensy_chibios/src/main/include/wheel_speed.h File Reference

```
#include <stdint.h>
```

Include dependency graph for wheel_speed.h:



This graph shows which files directly or indirectly include this file:



Functions

- int16_t wheel_speed_loop_fn (int16_t ticks)

  This function reads the motor ticks that have been determined by the hall sensor and converts this value into a speed for the truck.
- void wheel_speed_setup ()

4.21.1 Function Documentation

4.21.1.1 int16_t wheel_speed_loop_fn (int16_t *ticks*)

This function reads the motor ticks that have been determined by the hall sensor and converts this value into a speed for the truck.

Parameters

ticks	The number of ticks that is kept track of by the hall_sensor task.

Returns

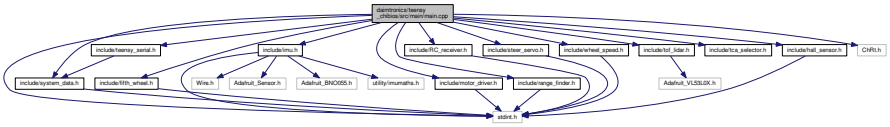
the speed of the truck

4.21.1.2 void wheel_speed_setup ( )

4.22 daimtronics/teensy_chibios/src/main/main.cpp File Reference

```
#include <stdint.h>
#include "include/system_data.h"
#include "include/fifth_wheel.h"
#include "include/imu.h"
#include "include/motor_driver.h"
#include "include/range_finder.h"
#include "include/RC_receiver.h"
#include "include/steer_servo.h"
#include "include/teensy_serial.h"
#include "include/wheel_speed.h"
#include "include/tof_lidar.h"
#include "include/tca_selector.h"
#include "include/hall_sensor.h"
#include <ChRt.h>
```

Include dependency graph for main.cpp:



Macros

- #define HWSERIAL Serial1
- #define FIFTH_WHEEL_PIN 6
- #define MOTOR_PIN 5
- #define STEER_SERVO_PIN 2
- #define IMU_SDA_PIN 18
- #define IMU_SCL_PIN 19
- #define TOF_LIDAR_SDA_PIN 18
- #define TOF_LIDAR_SCL_PIN 19
- #define RC_SW1_PIN 25
- #define RC_SW3_PIN 24
- #define HALL_PHASE_A_PIN 40
- #define HALL_PHASE_B_PIN 41
- #define HALL_PHASE_C_PIN 42

Functions

- MUTEX_DECL (sysMtx)
- static THD_WORKING_AREA (fifth_wheel_wa, 64)

  Fifth Wheel Thread: Reads desired state of the fifth wheel from the system_data and outputs a servo angle corresponding to locked or unlocked.
- static THD_FUNCTION (fifth_wheel_thread, arg)
- static THD_WORKING_AREA (imu_wa, 2048)

  IMU Thread: Reads euler angles from the BNO055 IMU and writes the data to the system_data after obtaining the system_data mutex.
- static THD_FUNCTION (imu_thread, arg)
- static THD_WORKING_AREA (motor_driver_wa, 64)

  Motor Driver Thread: Reads motor output levels from the system data and controls the motor based on this value.
- static THD_FUNCTION (motor_driver_thread, arg)
- static THD_WORKING_AREA (left_tof_wa, 512)

  Left Time of Flight Lidar Thread: Reads the distance in millimeters to the nearest object for the left Time of Flight sensor.
- static THD_FUNCTION (left_tof_thread, arg)
- static THD_WORKING_AREA (right_tof_wa, 512)

  Right Time of Flight Lidar Thread: Reads the distance in millimeters to the nearest object for the right Time of Flight sensor.
- static THD_FUNCTION (right_tof_thread, arg)
- CH_IRQ_HANDLER (RC_SW1_ISR_Fcn)
- static THD_WORKING_AREA (rc_sw1_isr_wa_thd, 128)

  RC Receiver Switch 1 Thread: Reads auxiliary signals from the RC receiver switch 1 for determining if the deadman switch is pressed.
- static THD_FUNCTION (rc_sw1_handler, arg)
- CH_IRQ_HANDLER (RC_SW3_ISR_Fcn)
- static THD_WORKING_AREA (rc_sw3_isr_wa_thd, 128)

  RC Receiver Switch 3 Thread: Reads auxiliary signals from the RC receiver switch 3 for determining what drive mode the semi-truck is in.
- static THD_FUNCTION (rc_sw3_handler, arg)
- static THD_WORKING_AREA (steer_servo_wa, 64)

  Steer Servo Thread: Controls the servo that dictates the driving angle of the semi truck.
- static THD_FUNCTION (steer_servo_thread, arg)
- static THD_WORKING_AREA (teensy_serial_wa, 2048)

  Teensy Serial Thread: Communicates over the serial (UART) port to relay system data between the Teensy and the Raspberry Pi.
- static THD_FUNCTION (teensy_serial_thread, arg)
- CH_IRQ_HANDLER (HALL_ISR_Fcn)
- static THD_WORKING_AREA (hall_sensor_wa, 5120)

  Hall Sensor Thread Controls the Hall sensor that is outputted from the main motor of the vehicle for determining Encoder Ticks.
- static THD_FUNCTION (hall_sensor_thread, arg)
- static THD_WORKING_AREA (speed_wa, 5120)

  Speed Thread: Controls the Hall sensor that is outputted from the main motor of the vehicle for determining Encoder Ticks.
- static THD_FUNCTION (speed_thread, arg)
- void chSetup ()

  Creates the threads to be run by assigning the thread function, working space, priority and any parameters that the thread needs.
- void setup ()

  Initializes the semi-truck system so that it is ready to run in an RTOS environment.
- void loop ()

Variables

- static system_data_t system_data = {0}

  The data for the entire system. Synchronization to access this resource is achieved through the use of ChibiOS's mutex library.
- static int16_t Encoder_ticks

  The number of ticks that have read based on the rotation of the three-phase motor. The ticks are determined from the Hall Sensor that is on the motor, and passed into the wheel_speed task to determine physical speed of the system.
- static thread_reference_t rc_sw1_isr_trp = NULL

  RC Receiver Switch 1 Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the RC receiver Switch 1 thread.
- static thread_reference_t rc_sw3_isr_trp = NULL

  RC Receiver Switch 3 Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the RC receiver Switch 3 thread.
- static thread_reference_t hall_isr_trp = NULL

  Hall Sensor Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the main Hall Sensor thread.

4.22.1 Macro Definition Documentation

4.22.1.1 #define FIFTH_WHEEL_PIN 6

4.22.1.2 #define HALL_PHASE_A_PIN 40

4.22.1.3 #define HALL_PHASE_B_PIN 41

4.22.1.4 #define HALL_PHASE_C_PIN 42

4.22.1.5 #define HWSERIAL Serial1

4.22.1.6 #define IMU_SCL_PIN 19

4.22.1.7 #define IMU_SDA_PIN 18

4.22.1.8 #define MOTOR_PIN 5

4.22.1.9 #define RC_SW1_PIN 25

4.22.1.10 #define RC_SW3_PIN 24

4.22.1.11 #define STEER_SERVO_PIN 2

4.22.1.12 #define TOF_LIDAR_SCL_PIN 19

4.22.1.13 #define TOF_LIDAR_SDA_PIN 18

4.22.2 Function Documentation

4.22.2.1 CH_IRQ_HANDLER (RC_SW1_ISR_Fcn)

4.22.2.2 CH_IRQ_HANDLER (RC_SW3_ISR_Fcn)

4.22.2.3 CH_IRQ_HANDLER (HALL_ISR_Fcn)

4.22.2.4 void chSetup ( )

Creates the threads to be run by assigning the thread function, working space, priority and any parameters that the thread needs.

While the static thread definitions are written before this function, none of them are used until chThdCreateStatic(...) is called.

4.22.2.5 void loop ( )

4.22.2.6 MUTEX_DECL (sysMtx)

4.22.2.7 void setup ( )

Initializes the semi-truck system so that it is ready to run in an RTOS environment.

The setup function is default to Arduino sketches, and holds all of the code that must be run before the main loop can be run. In this project, each task will have a setup function in it's .cpp file that is called here. Finally, ChibiOS setup is called to start initialize threads and start the thread scheduling that is built in to ChibiOS.

4.22.2.8 static THD_FUNCTION (fifth_wheel_thread , arg) [static]

4.22.2.9 static THD_FUNCTION (imu_thread , arg) [static]

4.22.2.10 static THD_FUNCTION (motor_driver_thread , arg) [static]

4.22.2.11 static THD_FUNCTION (left_tof_thread , arg) [static]

4.22.2.12 static THD_FUNCTION (right_tof_thread , arg) [static]

4.22.2.13 static THD_FUNCTION (rc_sw1_handler , arg) [static]

4.22.2.14 static THD_FUNCTION (rc_sw3_handler , arg) [static]

4.22.2.15 static THD_FUNCTION (steer_servo_thread , arg) [static]

4.22.2.16 static THD_FUNCTION (teensy_serial_thread , arg) [static]

4.22.2.17 static THD_FUNCTION (hall_sensor_thread , arg) [static]

4.22.2.18 static THD_FUNCTION (speed_thread , arg) [static]

4.22.2.19 static THD_WORKING_AREA (fifth_wheel_wa , 64) [static]

Fifth Wheel Thread: Reads desired state of the fifth wheel from the system_data and outputs a servo angle corresponding to locked or unlocked.

This thread calls fifth_wheel_loop_fn() which is the primary function for the fifth wheel and whose implementation is found in fifth_wheel.cpp.

4.22.2.20 static THD_WORKING_AREA (imu_wa , 2048) [static]

IMU Thread: Reads euler angles from the BNO055 IMU and writes the data to the system_data after obtaining the system_data mutex.

This thread calls imu_loop_fn() which is the primary function for the IMU and whose implementation is found in imu.cpp.

4.22.2.21 static THD_WORKING_AREA (motor_driver_wa , 64) [static]

Motor Driver Thread: Reads motor output levels from the system data and controls the motor based on this value.

This thread calls motor_driver_loop_fn which is the primary function for the motor and whose implementation is found in motor_driver.cpp.

4.22.2.22 static THD_WORKING_AREA (left_tof_wa , 512) [static]

Left Time of Flight Lidar Thread: Reads the distance in millimeters to the nearest object for the left Time of Flight sensor.

This thread calls tof_loop_fn which is the primary function for the motor and whose implementation is found in tof_lidar.cpp.

4.22.2.23 static THD_WORKING_AREA (right_tof_wa , 512) [static]

Right Time of Flight Lidar Thread: Reads the distance in millimeters to the nearest object for the right Time of Flight sensor.

This thread calls tof_loop_fn which is the primary function for the motor and whose implementation is found in tof_lidar.cpp.

4.22.2.24 static THD_WORKING_AREA (rc_sw1_isr_wa_thd , 128) [static]

RC Receiver Switch 1 Thread: Reads auxiliary signals from the RC receiver switch 1 for determining if the deadman switch is pressed.

This thread calls RC_receiver_SW1_fn which is the primary function for the RC receiver switch 1 and whose implementation is found in RC_receiver.cpp

4.22.2.25 static THD_WORKING_AREA (rc_sw3_isr_wa_thd , 128) [static]

RC Receiver Switch 3 Thread: Reads auxiliary signals from the RC receiver switch 3 for determining what drive mode the semi-truck is in.

This thread calls RC_receiver_SW3_fn which is the primary function for the RC receiver switch 3 and whose implementation is found in RC_receiver.cpp

4.22.2.26 static THD_WORKING_AREA (steer_servo_wa , 64) [static]

Steer Servo Thread: Controls the servo that dictates the driving angle of the semi truck.

This thread calls steer_servo_loop_fn which is the primary function for the steering servo and whose implementation is found in steer_servo.cpp

4.22.2.27 static THD_WORKING_AREA (teensy_serial_wa , 2048) [static]

Teensy Serial Thread: Communicates over the serial (UART) port to relay system data between the Teensy and the Raspberry Pi.

This thread calls serial_loop_fn() which is the primary function for the serial communication and whose implementation is found in teensy_serial.cpp.

4.22.2.28 static THD_WORKING_AREA (hall_sensor_wa , 5120) [static]

Hall Sensor Thread Controls the Hall sensor that is outputted from the main motor of the vehicle for determining Encoder Ticks.

This thread calls hall_sensor_loop_fn() which is the primary function for the Hall sensor and whose implementation is found in hall_sensor.cpp.

4.22.2.29 static THD_WORKING_AREA (speed_wa , 5120) [static]

Speed Thread: Controls the Hall sensor that is outputted from the main motor of the vehicle for determining Encoder Ticks.

This thread calls hall_sensor_loop_fn() which is the primary function for the Hall sensor and whose implementation is found in hall_sensor.cpp.

4.22.3 Variable Documentation

4.22.3.1 int16_t Encoder_ticks [static]

The number of ticks that have read based on the rotation of the three-phase motor. The ticks are determined from the Hall Sensor that is on the motor, and passed into the wheel_speed task to determine physical speed of the system.

4.22.3.2 thread_reference_t hall_isr_trp = NULL [static]

Hall Sensor Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the main Hall Sensor thread.

4.22.3.3 thread_reference_t rc_sw1_isr_trp = NULL [static]

RC Receiver Switch 1 Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the RC receiver Switch 1 thread.

4.22.3.4 thread_reference_t rc_sw3_isr_trp = NULL [static]

RC Receiver Switch 3 Interrupt Handler: Runs preemptive Chibios Interrupt code and awakens the RC receiver Switch 3 thread.

4.22.3.5 system_data_t system_data = {0} [static]

The data for the entire system. Synchronization to access this resource is achieved through the use of ChibiOS's mutex library.

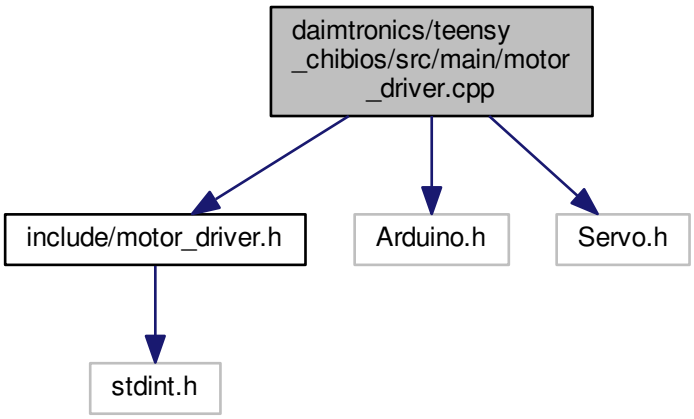
Tasks that control a sensor will call the primary function to read that sensor, obtain the mutex to for the system data, write the sensor data to the system_data, and then release the mutex on the system_data.

Tasks that control actuators will read from the system_data (no mutex needed as the system_data is effectively read-only for actuator tasks) to receive the specific data that corresponds to their task and then run their primary functions to control the actuators.

4.23 daimtronics/teensy_chibios/src/main/motor_driver.cpp File Reference

```
#include "include/motor_driver.h"
#include <Arduino.h>
#include <Servo.h>
```

Include dependency graph for motor_driver.cpp:



Macros

- #define WHEEL_SPEED_STOP 0
- #define MOTOR_STOP 90
- #define FORWARDS 120
- #define INIT_VALUE 68
- #define KP 1
- #define KI 0.05f
- #define SAT_ERROR 1000
- #define MAX_TIME_STEP 500
- #define WHEEL_SPEED_RANGE 1000
- #define MOTOR_RANGE 180
- #define FULL_REVERSE 1087
- #define FULL_FORWARD 1660

Functions

- void motor_driver_loop_fn (int16_t motor_output)

  This is the primary function controlling the motor. It reads the motor output value from the system data and controls the motor with this value.
- int16_t scale_output (int16_t motor_output)

  Scales a value coming from the pi between -100 and 100 to between 0 and 180 for what the Servo library requires.
- void motor_driver_setup (short motor_pin)

  Set up the motor driver task to write to the pin attached to the motor, and to be in the locked position. It also outputs a value corresponding to zero torque.
- int16_t stop_motor (int16_t wheel_speed, int16_t time_step)

  Runs a control loop to stop the motor based on the reported wheel speed, and returns a value to be output to the motor.

Variables

- static Servo motor

  This is a Servo object to control the motor. It relies on code from Servo.h which is built into the Arduino IDE, and is able to control motors as well as servos.
- static int16_t error_sum = 0

  The accumulated error for integral control of the control loop that stops the motor when the dead man's switch is not pressed.

4.23.1 Macro Definition Documentation

4.23.1.1 #define FORWARDS 120

4.23.1.2 #define FULL_FORWARD 1660

4.23.1.3 #define FULL_REVERSE 1087

4.23.1.4 #define INIT_VALUE 68

4.23.1.5 #define KI 0.05f

4.23.1.6 #define KP 1

4.23.1.7 #define MAX_TIME_STEP 500

4.23.1.8 #define MOTOR_RANGE 180

4.23.1.9 #define MOTOR_STOP 90

4.23.1.10 #define SAT_ERROR 1000

4.23.1.11 #define WHEEL_SPEED_RANGE 1000

4.23.1.12 #define WHEEL_SPEED_STOP 0

4.23.2 Function Documentation

4.23.2.1 void motor_driver_loop_fn (int16_t *motor_output*)

This is the primary function controlling the motor. It reads the motor output value from the system data and controls the motor with this value.

Parameters

motor_output	the output to the motor

4.23.2.2 void motor_driver_setup (short *motor_pin*)

Set up the motor driver task to write to the pin attached to the motor, and to be in the locked position. It also outputs a value corresponding to zero torque.

Parameters

motor_pin	The pin sends a PWM signal to the motor.

4.23.2.3 int16_t scale_output (int16_t *motor_output*)

Scales a value coming from the pi between -100 and 100 to between 0 and 180 for what the Servo library requires.

Parameters

motor_output	An input value, ideally between -100 and 100 (although it will be limited to -100 or 100 if outside this range

Returns

a value to output to the motor between 0 and 180

4.23.2.4 int16_t stop_motor (int16_t *wheel_speed,* int16_t *time_step*)

Runs a control loop to stop the motor based on the reported wheel speed, and returns a value to be output to the motor.

Parameters

wheel_speed	speed of the truck read by the wheel speed sensor
time_step	number of millis since the last time this task ran; used in integral control

Returns

the output to the motor

4.23.3 Variable Documentation

4.23.3.1 int16_t error_sum = 0 [static]

The accumulated error for integral control of the control loop that stops the motor when the dead man's switch is not pressed.

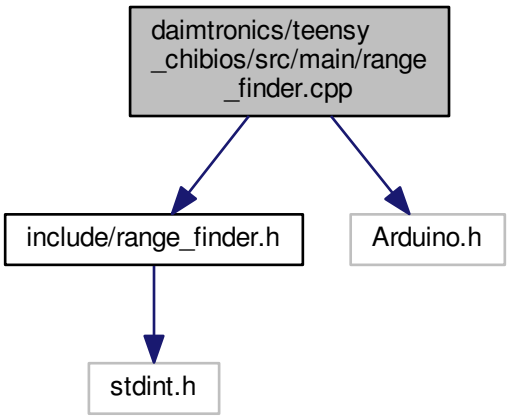
4.23.3.2 Servo motor [static]

This is a Servo object to control the motor. It relies on code from Servo.h which is built into the Arduino IDE, and is able to control motors as well as servos.

4.24 daimtronics/teensy_chibios/src/main/range_finder.cpp File Reference

```
#include "include/range_finder.h"
#include "Arduino.h"
```

Include dependency graph for range_finder.cpp:



Functions

- long range_finder_loop_fn (short urf_echo_pin)
- void range_finder_ping (short urf_trig_pin)
- void range_finder_setup (short urf_trig_pin)

Variables

- int val = 0
- unsigned long high_time = 0
- long distance = 0
- unsigned long time = 0

4.24.1 Function Documentation

4.24.1.1 long range_finder_loop_fn (short *urf_echo_pin*)

This is the primary function controlling the URFs. It reads a value representing a distance to an object from the URF sensor and returns that value

Returns

distance to object

4.24.1.2 void range_finder_ping (short *urf_trig_pin*)

4.24.1.3 void range_finder_setup (short *urf_trig_pin*)

4.24.2 Variable Documentation

4.24.2.1 long distance = 0

4.24.2.2 unsigned long high_time = 0

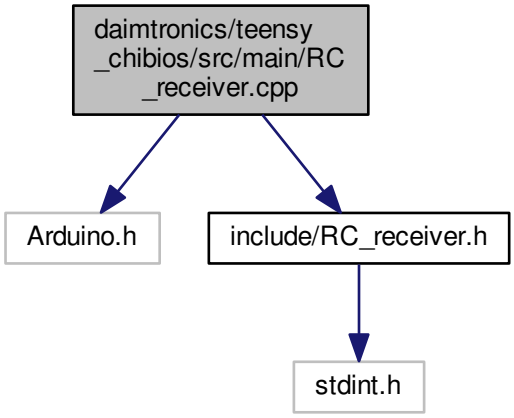
4.24.2.3 unsigned long time = 0

4.24.2.4 int val = 0

4.25 daimtronics/teensy_chibios/src/main/RC_receiver.cpp File Reference

```
#include <Arduino.h>
#include "include/RC_receiver.h"
```

Include dependency graph for RC_receiver.cpp:



Functions

- int16_t RC_receiver_SW1_fn (short PWM_PIN)

  This is the primary function reading Switch 1 on the receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a deadman mode (either deadman switch pressed or not pressed) is selected to send to the rest of the platform.
- int16_t RC_receiver_SW3_fn (short PWM_PIN)

  This is the primary function controlling the RC receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a drive mode (either manual or one of the autonomous algorithms on the Pi) is selected to control the vehicle.
- void RC_receiver_setup ()

Variables

- float SW1_high_time = 0
- volatile unsigned long SW1_time = 0
- short SW1_mode = 0
- float SW3_high_time = 0
- volatile unsigned long SW3_time = 0
- short SW3_mode = 0

4.25.1 Function Documentation

4.25.1.1 void RC_receiver_setup ( )

4.25.1.2 int16_t RC_receiver_SW1_fn (short *PWM_PIN*)

This is the primary function reading Switch 1 on the receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a deadman mode (either deadman switch pressed or not pressed) is selected to send to the rest of the platform.

Returns

the deadman mode of the semi-truck based on RC receiver signal

4.25.1.3 int16_t RC_receiver_SW3_fn (short *PWM_PIN*)

This is the primary function controlling the RC receiver. It reads a PWM signal that the RC receiver receives from the RC controller. Based on the specific timing of the PWM, a drive mode (either manual or one of the autonomous algorithms on the Pi) is selected to control the vehicle.

Returns

the driving mode of the semi-truck based on RC receiver signal

4.25.2 Variable Documentation

4.25.2.1 float SW1_high_time = 0

4.25.2.2 short SW1_mode = 0

4.25.2.3 volatile unsigned long SW1_time = 0

4.25.2.4 float SW3_high_time = 0

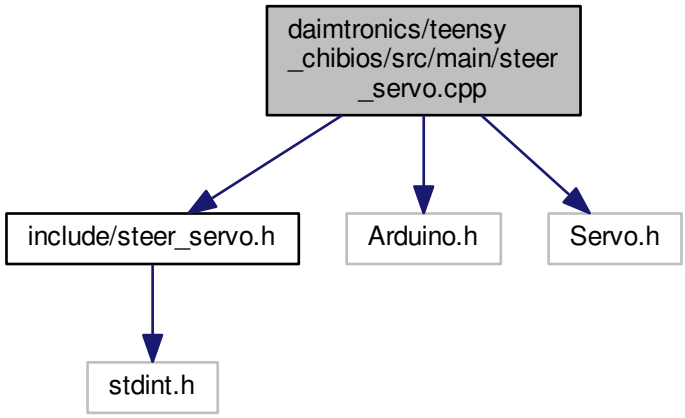
4.25.2.5 short SW3_mode = 0

4.25.2.6 volatile unsigned long SW3_time = 0

4.26 daimtronics/teensy_chibios/src/main/steer_servo.cpp File Reference

```
#include "include/steer_servo.h"
#include <Arduino.h>
#include <Servo.h>
```

Include dependency graph for steer_servo.cpp:



Macros

- #define STRAIGHT 90
- #define MIN_ANGLE 1400
- #define MAX_ANGLE 1800

Functions

- void steer_servo_loop_fn (int16_t steer_output)
- void steer_servo_setup (short servo_pin)

  Set up the steer servo task to write to the pin attached to the servo controlling angle of the trucks's front axis.

Variables

- static Servo steer_servo

4.26.1 Macro Definition Documentation

4.26.1.1 #define MAX_ANGLE 1800

4.26.1.2 #define MIN_ANGLE 1400

4.26.1.3 #define STRAIGHT 90

4.26.2 Function Documentation

4.26.2.1 void steer_servo_loop_fn (int16_t *steer_output*)

The is the primary function controlling the steering servo wheel. It reads a value form the system data and controls the steering servo based on what the system data contains.

Parameters

steer_output	the output to the steering servo that controls angle

4.26.2.2 void steer_servo_setup (short *servo_pin*)

Set up the steer servo task to write to the pin attached to the servo controlling angle of the trucks's front axis.

Parameters

servo_pin	The pin that signals a PWM to the steering servo.

4.26.3 Variable Documentation

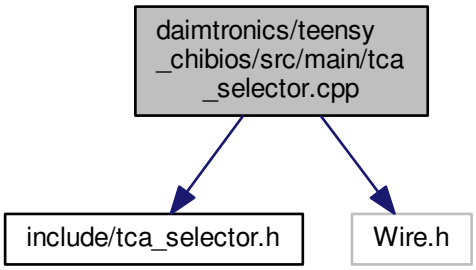
4.26.3.1 Servo steer_servo [static]

This is a Servo object to control the steering servo. It relies on code from Servo.h which is built into the Arduino IDE.

4.27 daimtronics/teensy_chibios/src/main/tca_selector.cpp File Reference

```
#include "include/tca_selector.h"
#include <Wire.h>
```

Include dependency graph for tca_selector.cpp:



Macros

- #define TCAADDR 0x70

Functions

- void tcaselect (short i)

4.27.1 Macro Definition Documentation

4.27.1.1 #define TCAADDR 0x70

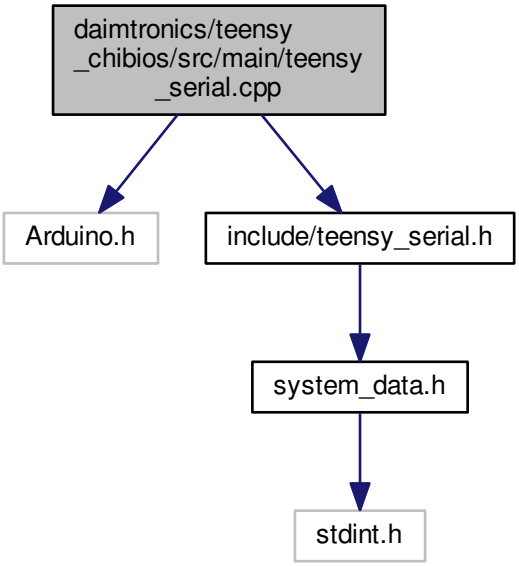
4.27.2 Function Documentation

4.27.2.1 void tcaselect (short *i*)

4.28 daimtronics/teensy_chibios/src/main/teensy_serial.cpp File Reference

```
#include <Arduino.h>
#include "include/teensy_serial.h"
```

Include dependency graph for teensy_serial.cpp:



Macros

- #define SHORT_SIZE 2
- #define ACT_DATA_SIZE_W_SYNC 8
- #define ACT_DATA_SIZE 6
- #define READ_CYCLES 2
- #define SYNC_VALUE -32000

Functions

- void teensy_serial_loop_fn (system_data_t *system_data)

  The primary function for communicating between the Teensy and the Pi over the Serial UART port.
- void teensy_serial_setup ()

  Sets up the serial communication for the teensy to output data to both the Pi (through UART) and a PC console (through USB).
- void clear_buffer ()

  Clears the serial buffer from all of its data. This is used primarily when the Teensy is restarted, and the Pi has been sending data that the Teensy does not need to process.
- void teensy_sync ()

  Ensures that the data being read from the Pi is synced. It clears the serial buffer up until it reads the designated SYNC_VALUE.
- void read_from_pi (actuator_data_t *actuators_ptr)
- void print_sensor_msg (sensor_data_t *sensors_ptr)
- void print_actuator_msg (actuator_data_t *actuators_ptr)

4.28.1 Macro Definition Documentation

4.28.1.1 #define ACT_DATA_SIZE 6

4.28.1.2 #define ACT_DATA_SIZE_W_SYNC 8

4.28.1.3 #define READ_CYCLES 2

4.28.1.4 #define SHORT_SIZE 2

4.28.1.5 #define SYNC_VALUE -32000

4.28.2 Function Documentation

4.28.2.1 void clear_buffer ( )

Clears the serial buffer from all of its data. This is used primarily when the Teensy is restarted, and the Pi has been sending data that the Teensy does not need to process.

4.28.2.2 void print_actuator_msg (actuator_data_t * *actuators_ptr*)

4.28.2.3 void print_sensor_msg (sensor_data_t * *sensors_ptr*)

4.28.2.4 void read_from_pi (actuator_data_t * *actuators_ptr*)

4.28.2.5 void teensy_serial_loop_fn (system_data_t * *system_data*)

The primary function for communicating between the Teensy and the Pi over the Serial UART port.

Parameters

system_data	a pointer to the system data that is declared statically in main.ino.

4.28.2.6 void teensy_serial_setup ( )

Sets up the serial communication for the teensy to output data to both the Pi (through UART) and a PC console (through USB).

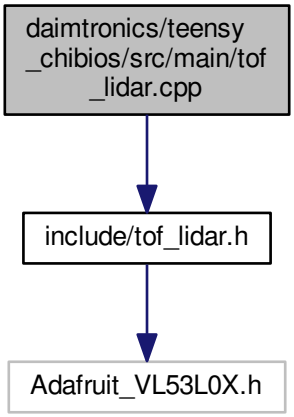
4.28.2.7 void teensy_sync ( )

Ensures that the data being read from the Pi is synced. It clears the serial buffer up until it reads the designated SYNC_VALUE.

4.29 daimtronics/teensy_chibios/src/main/tof_lidar.cpp File Reference

```
#include "include/tof_lidar.h"
```

Include dependency graph for tof_lidar.cpp:



Macros

- #define TCAADDR 0x70

Functions

- void tcaselect (uint8_t i)
- int16_t tof_loop_fn ()

  This is the primary function controlling the ToF Lidar for reading distance measurements on the sensors.
- void tof_lidar_setup ()

  Initializes the VL53L0X sensor.

Variables

- static bool initialized = false
- Adafruit_VL53L0X sensor1 = Adafruit_VL53L0X()

  A global variable that sets up the sensors to be used here.
- Adafruit_VL53L0X sensor2 = Adafruit_VL53L0X()
- Adafruit_VL53L0X sensor3 = Adafruit_VL53L0X()

4.29.1 Macro Definition Documentation

4.29.1.1 #define TCAADDR 0x70

4.29.2 Function Documentation

4.29.2.1 void tcaselect (uint8_t *i*)

4.29.2.2 void tof_lidar_setup ( )

Initializes the VL53L0X sensor.

4.29.2.3 int16_t tof_loop_fn ( )

This is the primary function controlling the ToF Lidar for reading distance measurements on the sensors.

The Adafruit_VL53L0X library does most of the work and the function here calls the measure.RangeMilliMeter instruction and stores the distance here.

Returns

an integer representing distance the sensor detected in millimeters

4.29.3 Variable Documentation

4.29.3.1 bool initialized = false [static]

4.29.3.2 Adafruit_VL53L0X sensor1 = Adafruit_VL53L0X()

A global variable that sets up the sensors to be used here.

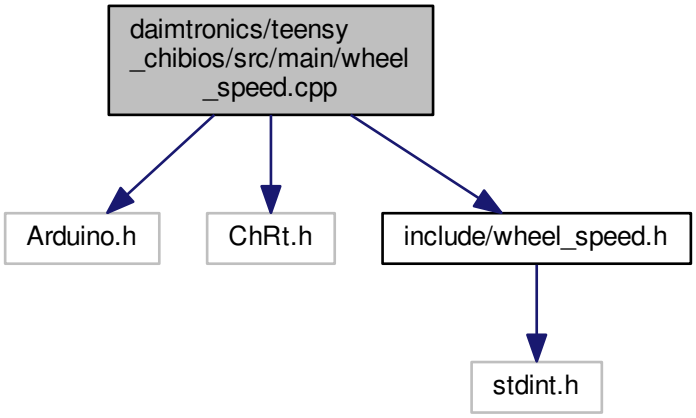
4.29.3.3 Adafruit_VL53L0X sensor2 = Adafruit_VL53L0X()

4.29.3.4 Adafruit_VL53L0X sensor3 = Adafruit_VL53L0X()

4.30 daimtronics/teensy_chibios/src/main/wheel_speed.cpp File Reference

```
#include <Arduino.h>
#include <ChRt.h>
#include "include/wheel_speed.h"
```

Include dependency graph for wheel_speed.cpp:



Macros

- #define SCALE 1
- #define MAX_CHANGE 32768

Functions

- int16_t wheel_speed_loop_fn (int16_t ticks)

  This function reads the motor ticks that have been determined by the hall sensor and converts this value into a speed for the truck.
- void wheel_speed_setup ()

Variables

- int16_t speed
- int16_t prev_ticks =0
- uint16_t prev_time = chVTGetSystemTime()

4.30.1 Macro Definition Documentation

4.30.1.1 #define MAX_CHANGE 32768

4.30.1.2 #define SCALE 1

4.30.2 Function Documentation

4.30.2.1 int16_t wheel_speed_loop_fn (int16_t *ticks*)

This function reads the motor ticks that have been determined by the hall sensor and converts this value into a speed for the truck.

Parameters

ticks	The number of ticks that is kept track of by the hall_sensor task.

Returns

the speed of the truck

4.30.2.2 void wheel_speed_setup ( )

4.30.3 Variable Documentation

4.30.3.1 int16_t prev_ticks =0

4.30.3.2 uint16_t prev_time = chVTGetSystemTime()

4.30.3.3 int16_t speed