Critical Design Review Report

IGVC Team Joyride

Oklahoma State University

Spring 2023

Contents

1 Project Overview

The overarching objective of the Intelligent Ground Vehicle Competition (IGVC) capstone project is to develop a self-driving vehicle capable of competing in the annual IGVC Self-Drive challenge. In order to do so, the team must develop a fully-functional drive-by-wire (DBW) package on the vehicle and a complex software suite to enable autonomous operation. The modifications are being made to a GEM e2 small electric car, and have been in progress for three semesters - Spring 22, Fall 22, Spring 23.

1.1 Intelligent Ground Vehicle Competition

The Intelligent Ground Vehicle Competition (IGVC) is an annual autonomous robotics competition hosted at Oakland University in Rochester, Michigan. Team Joyride aims to compete in the 30th annual competition from June 2nd through June 5th, 2023.

IGVC is comprised of several challenges, notably AutoNav and Self-Drive. AutoNav involves a small gorund vehicle navigating an obstacle course, while Self-Drive encompasses the construction of a self driving vehicle to travel in an urban street environment. This involves lane marker detection and following, lane changing, traffic sign detection, pedestrian and object detection, waypoint navigation, and more.

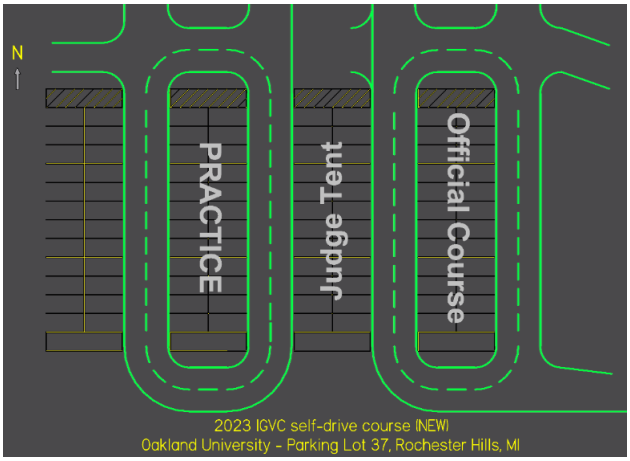


Figure 1: IGVC 2023 course layout from official IGVC rules document

1.1.1 Scoring

The competition is scored in a number of ways. First, a vehicle must qualify for the competition by passing a series of *safety and qualification tests*, noted in Appendix A.

Points are awarded in two categories: *Functional Tests* and *Main Course Performance*. The Functional Tests are worth 54% of the score, and are comprised of several discrete unit tests demonstrating autonomous performance; details are found in Appendix B. The main course and practice course are shown in Figure 1. The main course is a series of challenges that the vehicle must navigate through continuously; details are found in Appendix C.

1.1.2 Miscellaneous

Readers may find it helpful to know the following:

- The competition requires $\$1,000,000$ in general liability insurance to compete. The university will be handling this.
- Traditional machine vision techniques must be used for most purposes. Machine-learning approaches can be utilized for traffic sign detection.
- The rules specify a 75 inch height limit. The team contacted IGVC organizers, who clarified sensors can reasonably exceed this requirement while respecting the spirit of the competition.

1.2 Team Introduction

The interdisciplinary team is composed of electrical, computer, and mechanical engineering students. Team makeup for the Spring 2023 semester is shown in Table 1.

Table 1: Team Composition

Name	Major	Role
Max DeSantis	Electrical, Computer	Team Lead; Software Lead
Brenden Wickman	Electrical Technology	Electrical Lead; Purchasing Agent
Isabell Cook	Computer	Software - Perception
Kelvin Tran	Electrical, Computer	Software - Perception
Jason Aquino	Electrical	Software - Controls; Planner
Isaac Castilleja	Mechanical	Software - Interface
Nathan Wilson	Mechanical	Software - Controls
Jack Funderburgh	Mechanical	Mechanical Lead
Bailey DeSpain	Mechanical	
Troy Willoughby	Mechanical	
Kale Downing	Mechanical	
Christopher Shropshire	Mechanical	
Zeke Lappe	Mechanical	

1.3 Design Objectives

The overall objective is to **build a self driving car**. Objectives roughly break down into four categories: drive-by-wire, perception, navigation, and mounting.

1.3.1 Drive-by-Wire

Previous semesters prepared a functional drive-by-wire (DBW) system that allows computer control of throttle and steering angle.

This semester, the primary goals were the following:

1. Construct a robust and reliable autonomous braking system
2. Simplify custom PCBs to allow for direct computer to PCB CAN bus communication
3. Introduce additional safety features and feedback from the PCBs and vehicle to the onboard computer

1.3.2 Perception

The competition requires using computer vision to detect certain objects so that they can be navigated around. Per competition rules lane, pedestrian, and tire detection must be detected using traditional computer vision while the rest of the objects may be detected using other computer vision approaches, such as neural networks.

Objects include:

1. Lane Lines
2. Pedestrian
3. Traffic Barrels
4. Potholes
5. Traffic Signs
6. Tires

1.3.3 Navigation

The competition requires navigating between waypoints, avoiding obstacles, and lane following. Waypoint navigation forms the basis of all other movement related tasks, as a behavior tree will build waypoint splines to achieve other objectives like obstacle avoidance.

These tasks include:

1. Waypoint following
2. Lane following
3. Obstacle avoidance (barrels, pedestrians, potholes)
4. Parking (parallel and pull-in)
5. Lane changing
6. Merging
7. Turning through intersections
8. Stopping at intersections

1.3.4 Mounting

Numerous sensors and other components need to be securely mounted to the vehicle. Many of these require significant back and forth with the software team to ensure ideal placement. Sensors need to be easily removed and remounted without changing position or orientation.

1. Forward Planar Lidar (Hokuyo 30m)
2. Rear Planar Lidar (Hokuyo 10m)
3. Forward 3D Lidar (Blickfeld Cube 1 Outdoor)
4. Lane Camera
5. Two side cameras
6. GPS and Antennas
7. Power Inverter
8. Computer
9. Touchscreen
10. Rear wheel encoders
11. Emergency Stop Buttons
12. Fire extinguisher and First Aid Kit

2 Software Design

There are numerous interconnected software components necessary to achieve the project's desired functionality. The software stack is oriented around the Robot Operating System (ROS). Specifically, we are using ROS2 Foxy Fitzroy[1] on Ubuntu 20.04. This was chosen due to its relative maturity compared to other ROS2 versions, but newer navigation features compared to ROS1.

The vehicle's software stack is hosted on the *osu-igvc* GitHub organization[2], which requires member-privileges to view. The software is broken down into multiple ROS packages under the *joyride-ros-main* colcon workspace repository. These packages include:

- joyride_core. Holds utility and launch files.
- joyride_interfaces. Holds ROS interfaces (message, action, service definitions).
- joyride_hmi. Holds any human-machine interface components (GUI).
- joyride_perception
- joyride_sensors. Holds custom and off-the-shelf sensor drivers.
- joyride_servers. Holds system-wide servers for health monitoring, static transforms, etc.
- joyride_ros2_socketcan. Holds implementation of ROS-CAN communication. Must remain private due to Allied Motion NDA, may be split apart later.
- Other miscellaneous packages. Package definitions are not final nor rigidly enforced to allow for easier development.

A simplified software overview can be seen in Figure 2, with additional progress information embedded.

[1] https://docs.ros.org/en/foxy/index.html
[2] https://github.com/osu-igvc

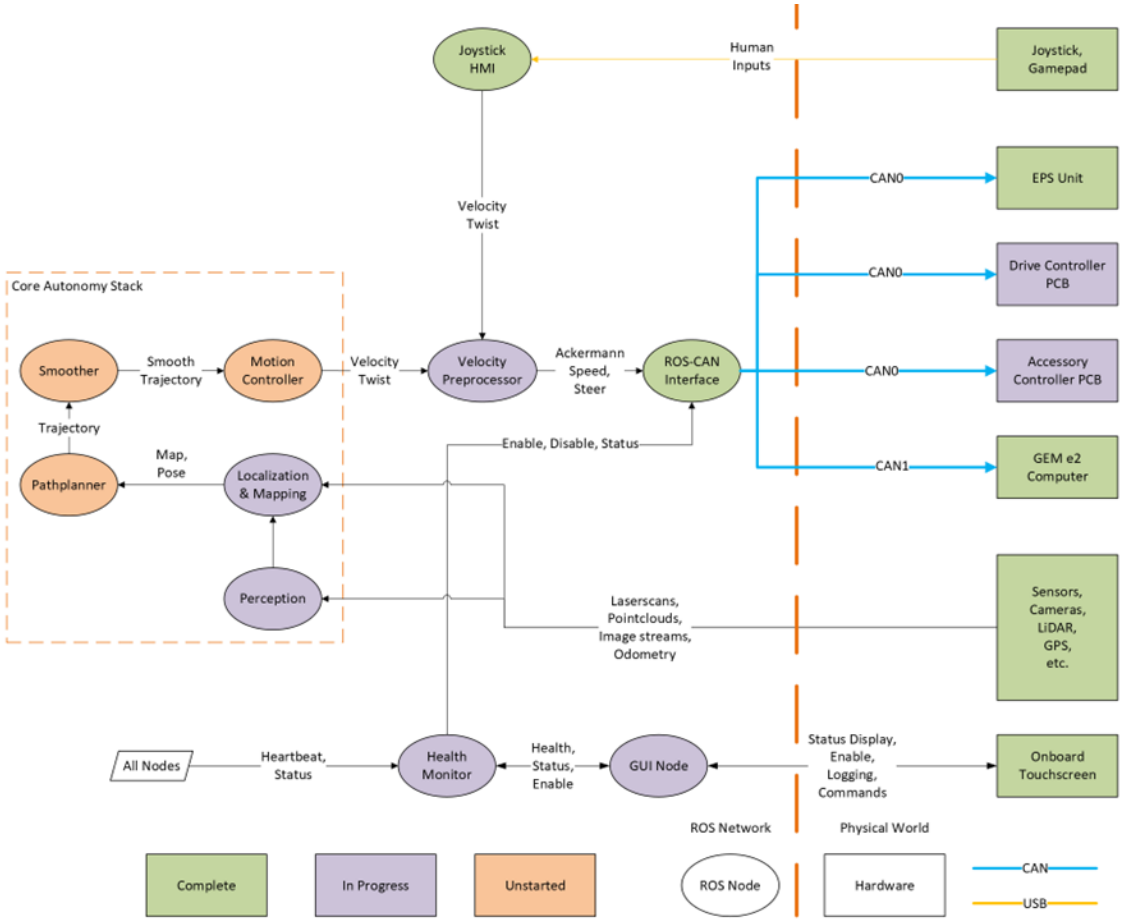


Figure 2: Simplified Software Overview

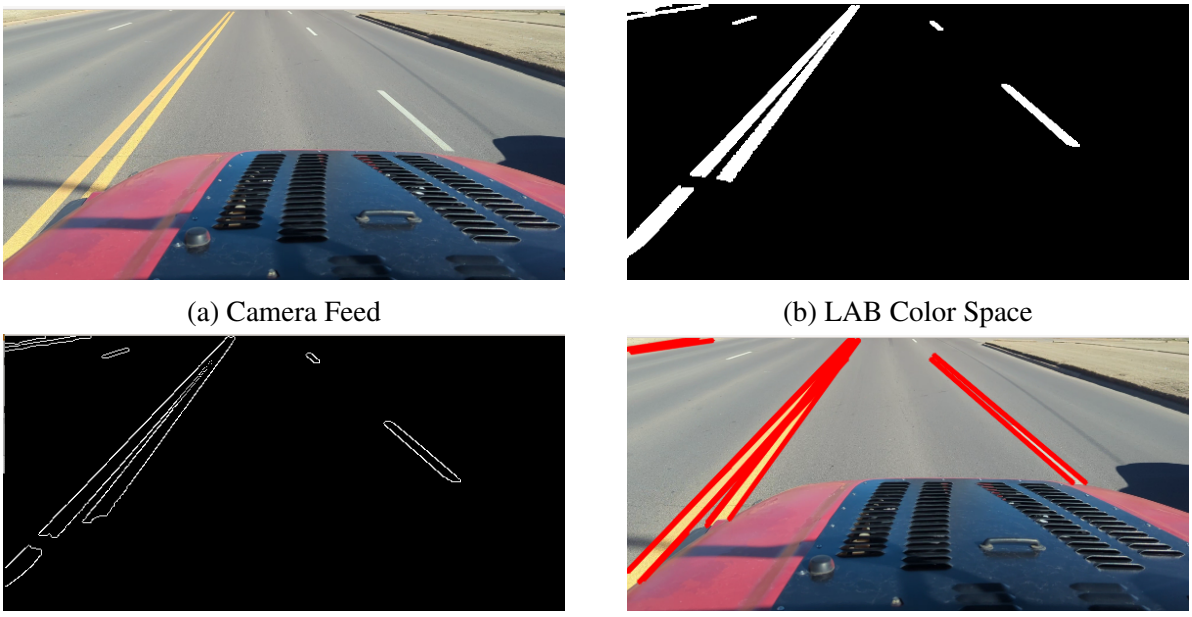
2.1 Perception

2.1.1 Lane Detection

For the Intelligent Ground Vehicle Competition (IGVC), lane detection is required to utilize traditional computer vision which encourages detection primarily based on shape and color. Furthermore, the lanes in the competition are comprised of a one-way, two-lane road that will be marked in three-inch white tape on asphalt. The lanes will be eight feet apart and contain a curve with a minimum twenty-foot radius. Taking these specifications into account the lanes can be detected by their light color on top of the dark asphalt.

Implementation of this involves getting the individual frames from the video stream and converting the color space of the frame from RGB, with axes representing red, green, and blue, to that of Lab, with axes representing lightness, red/green, and blue/yellow. The lightness channel is then extracted and a threshold is applied so that only pixels that are brighter than the threshold are collected. This set of light points is sent through Opencv's edge detector and then through a Hough Transform. The Hough Transform takes the edges and runs a line of best fit between the points collected by the edge detection. The lines with the highest probability of making up lines from the given edges are returned as a set of line segments. These segments are combined to form

distinct lane lines along which samples are going to be extracted and converted into 3D space for use in Navigation.



(a) Camera Feed

(b) LAB Color Space

(c) Edge Detection

(d) Hough Transform

Figure 3: Lane Detection Steps and Output

2.1.2 Object Detection

For the Intelligent Ground Vehicle Competition (IGVC), object detection is required to utilize traditional computer vision in order to detect and extract desired objects. Traditional computer vision methods exclude any implementation of a neural network to detect targets. There is a short list of objects to identify stated in the IGVC functionality test: pedestrians (mannequin wearing an orange high viz jacket), orange traffic barrels, and tires. Potholes are included in the desired detection list as well; however, these can be detected either with traditional or modern computer vision algorithms. For implementation, object detection is split into two categories: pedestrian detection and obstacle detection. Pedestrian detection serves to identify human-like figures in the frame while obstacle detection serves to identify objects in the pathway of the vehicle. For both pedestrian and obstacle detection programs, the OpenCV library assists with its real-time computer vision functions.[3] Additional applicable functions in this library include Kalman filters, edge detection algorithms, image processing functions, etc.

2.1.2.1 Pedestrian Detection

For pedestrian detection, the live video feed must be analyzed to capture and extract a pedestrian as an orange blob to display on the onboard GUI of the vehicle. The functional test for pedestrian detection includes a mannequin wearing an orange high-visibility jacket. Simply detecting the

[3] https://docs.opencv.org/4.7.0/

orange is not enough because there are many shades of orange in the potential testing location as a result of lighting conditions.

To solve this issue, a traditional computer vision algorithm is utilized to detect the general shape of a person. This algorithm is known as the Histogram of Oriented Gradients (HOG) along with a Support Vector Machine (SVM) to assist in pedestrian detection. The HOG-SVM algorithm is a feature extraction algorithm that vectorizes each frame of the video feed in order to create a histogram. The HOG algorithm divides a frame/image into small cells to compute the magnitude and orientation of the gradients within the cell. This essentially vectorizes an image in order to construct a histogram of gradient orientation. Afterward, the SVM classifier will distinguish if the frame contains the desired object using the vectorized image produce with the HOG algorithm. The SVM classifier is based on a trained model from a dataset of positive and negative images. Our implementation utilizes the OpenCV library's HOG function along with its SVM classifier. Once a pedestrian is identified, a bounding box will crop the target where an orange color filter is applied to produce the desired orange blob from the high viz jacket. After, the pedestrian detection program will send a flag/message within the ROS network to notify the system if a pedestrian is in the vehicle's lane of traffic.

However, this type of detection may lead to incorrect identification, where a person may not be continuously detected and tracked, if the system only relies on a traditional computer vision algorithm using a linear classifier. The classifier cannot consistently determine that a figure is a person from some point of view. To counteract this issue, a Kalman filter is implemented for a detected pedestrian. The Kalman filter is used for state estimation of an object based on its position, velocity, and orientation in space or time. Once a pedestrian is detected using the HOG-SVM algorithm, the implemented Kalman filter will ideally track that person until the target is out of view. This process essentially smooths any potential flickering issues which is the result of unusual angles for detection that the algorithm has difficulties identifying as a person or not.

Figure 4: Pedestrian Detection Output

2.1.2.2 Obstacle Detection

For obstacle detection, the live video feed must be analyzed to capture and extract the shape of a detected object and display the extracted shape on the onboard GUI of the vehicle. The obstacle detection functional test involves the placement of a tire in the lane of the vehicle. An implementation of a traditional computer vision algorithm must be able to detect an obstacle in the path of the vehicle and display it. This involves an edge detection process to find all contours of each frame of the live video feed while differentiating each object.

In our implementation of obstacle detection, the OpenCV library is extensively utilized to process the edge detection algorithm. OpenCV's Canny edge detection function is incorporated to find and display all edges within a specified area threshold. Defining a threshold in the edge detection function eliminates small object contours extracted within a frame.

First, the algorithm will crop each incoming frame to only perform edge detection in a Region of Interest (ROI). Since obstacle detection is used to detect objects in a pathway of the vehicle, the ROI can be visualized as a trapezoid or a triangle from the bottom of the frame up to the horizon line of the vehicle's perspective.

Then, the frame will be converted to a grayscale image, so the application of the Canny edge detection can extract the contours of all objects in the ROI. The grayscale conversion process is optional, but it removes some finer and unnecessary details and noise that may show as another tiny object. Furthermore, dilation can be applied to further remove small object detection and some additional noise.

Afterward, the Canny edge detection function is applied to the frame where contours of objects within a specified threshold appear within the cropped ROI. Now, the program will extract the object from the original frame based on complete contours within the ROI. A complete contour determines an object in the frame because it produces an outline of a shape. Incomplete contours can be seen as lines or incomplete shapes from the Canny edge detection function. Furthermore, the complete contour will provide the location to extract the image of the detected object in the vehicle's pathway.

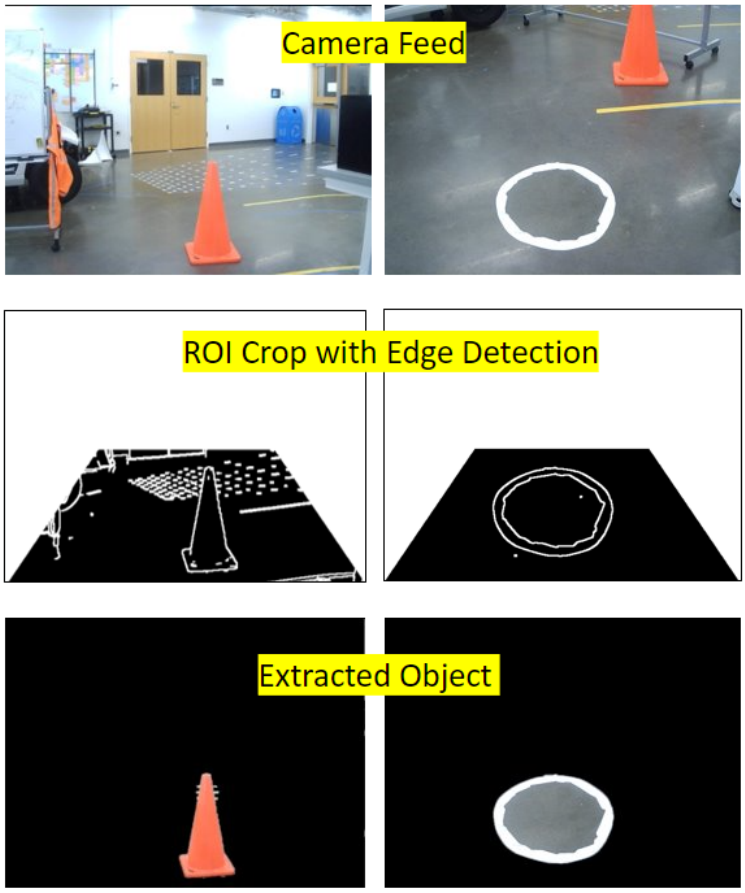


Figure 5: Obstacle Detection Output

2.1.3 Traffic Sign Detection

For the competition, most detection methods must use the traditional computer vision approach, which excludes neural networks. However, traffic sign detection and pothole detection may utilize neural networks to identify their targets. Traffic signs are detected with a machine-learning approach. A convolutional neural network uses video feed to quickly classify pedestrians, stop signs, vehicles, and more. Our implementation was produced using the YOLOv5 object classification architecture.[4] The YOLO algorithm is a single-shot detector that uses a convolutional neural network for real-time object detection. This implementation is selected due to its quick detection and high accuracy. The ROS node connecting to the neural network will provide pixel position of the object's center, classification type, and confidence level.

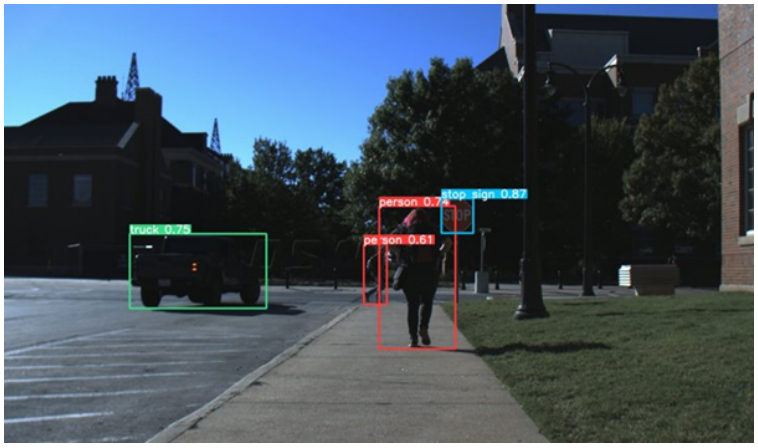


Figure 6: YOLO Object Classifier Output

2.2 Navigation

The team is utilizing ROS 2 Foxy's Navigation 2 (NAV2)[5] software stack to handle global planning and motion control. NAV2 is organized around behavior trees (BTs) that make requests of servers on the ROS network. For example, the *NavigateToPose* BT may request a new global path from the current position to a goal pose. This request is sent to the *planner* server, which computes and returns the requested path.

BTs can be created and combined in whatever manner necessary to achieve desired behaviors. The team will initially use stock NAV2 trees[6] (NavigateToPose, NavigateThroughPoses, etc.) to validate sensing, perception, and estimation algorithms. Later, custom BTs will be introduced.

2.2.1 Localization and State Estimation

State estimation is traditionally performed using Extended Kalman Filters (EKFs) that fuse a variety of sensor information to produce pose estimates. This may include XYZ position, roll-pitch-yaw orientation, and GPS location. After testing, the team chose not to use this approach due to its

[4] https://github.com/ultralytics/yolov5
[5] https://navigation.ros.org/
[6] https://navigation.ros.org/behavior_trees/index.html

time-intensive nature and reliance on somewhat well-behaved data. Instead, the team will be using the already-fused pose estimates provided by our Vectornav VN300 Inertial Navigation System (INS). The INS produces GPS latitude, longitude, and yaw estimates at 400Hz by fusing 5Hz GPS and 400Hz IMU.

These positions will be transformed into a world-fixed East-North-Up (ENU) coordinate system using a local tangent plane approximation. A static transform between the *map* and *odom* frames will be published with the INS data. This is irregular and does not entirely conform to ROS REP 105[7], the standard for mobile robot frame conventions. Regardless, this approach is advantageous due to the accuracy of the INS and less computational overhead necessary to perform traditional localization.

INS data was recorded on a long trip through OSU's campus. The data is shown in Figure 7. The VN300's INS picks up small changes in position and is accurate to within roughly a foot, which the team believes is acceptable.

Figure 7: GPS data from long trip through campus. Graphed using the collected rosbag and Mapviz

2.2.2 Environmental Representation and Mapping

The environment will be represented as a two-dimensional costmap, often referred to as an occupancy grid in ROS nomenclature. *Lethal* obstacles are those that the vehicle can collide with, such as barrels and pedestrians. *Nonlethal* obstacles should be avoided to optimize performance, but are not safety-critical; lane markers and potholes are examples of this type. Lethal obstacles will be inflated to a much higher cost than nonlethal obstacles so the vehicle crosses a lane rather than strikes an object.

[7]https://www.ros.org/reps/rep-0105.html#map

The map will be generated using a Simultaneous Localization and Mapping (SLAM) approach. The localization component will be provided by the INS, meaning SLAM is concerned only with building the map during operation. Obstacle information will come from two sources: cameras and LIDAR. LIDAR returns will be directly fed into SLAM. Camera frames will be interpreted for various obstacles and inserted into SLAM as pseudo-LIDAR. More information can be found in Section 2.1.

2.2.3 Global Planner

The global planner is responsible for computing global paths between the robot's current position and some goal pose. A path is represented as a series of intermediate goal poses. The team will be using the *SmacPlannerHybrid* planner, which uses the Hybrid A* algorithm to build minimum cost paths given an environment costmap.

The planner will be hosted on a *planner server*, spawned by NAV2.

2.2.4 Motion Controller

Once a path is generated, the vehicle must generate desired velocity twists (linear X and angular Z) to follow the path. The twists are generated by the motion controller, which can implement any number of algorithms. The team will be experimenting with both the *Timed-Elastic-Band* and *Regulated Pure Pursuit* controllers.

The controller will be hosted on a *controller server*, spawned by NAV2. The server will handle requests to generate new trajectories from the primary navigation behavior tree.

2.3 Interface

2.3.1 Graphical User Interface

The vehicle has a touchscreen monitor mounted to the dashboard that hosts a GUI built with JavaScript, HTML, and Bootstrap, a mobile-first CSS framework. Using JavaScript allows for the creation of a professional-looking interface that natively supports touchscreen functionality. JavaScript also has a larger community in comparison to other GUI development options like PyQt for Python which makes finding support and updated libraries easier. Additionally, the GUI can be easily supported across platforms by using Electron, a framework that embeds Chromium and Node.js for building desktop applications with JavaScript, HTML, and CSS.

The GUI is made up of five pages: The dashboard, commands, sensor data, system health, and waypoints page.

2.3.1.1 Dashboard

The Dashboard contains a video stream, vehicle metrics, and a toggle autonomy button. Currently, the video stream only displays the feed of the camera mounted to the vehicle, but further functionality will be added to allow the user to switch between what video stream is displayed. The vehicle metrics section shows the current speed, steering angle, battery level, and gear of the vehicle. CSS animations are used along with common icons that users can instantly recognize and

use to get feedback about the vehicle at a glance. When pressed, the toggle autonomy button sends a request to the system to enable autonomy and, if the system is healthy, an overlay will appear that will prevent the user from interacting with the GUI and a countdown until autonomy will be displayed, with a button that the user can press to cancel the process. After autonomy is entered or the process is canceled, the GUI returns to normal operation. Pressing the button while the vehicle is in autonomy will return the vehicle back to manual mode.

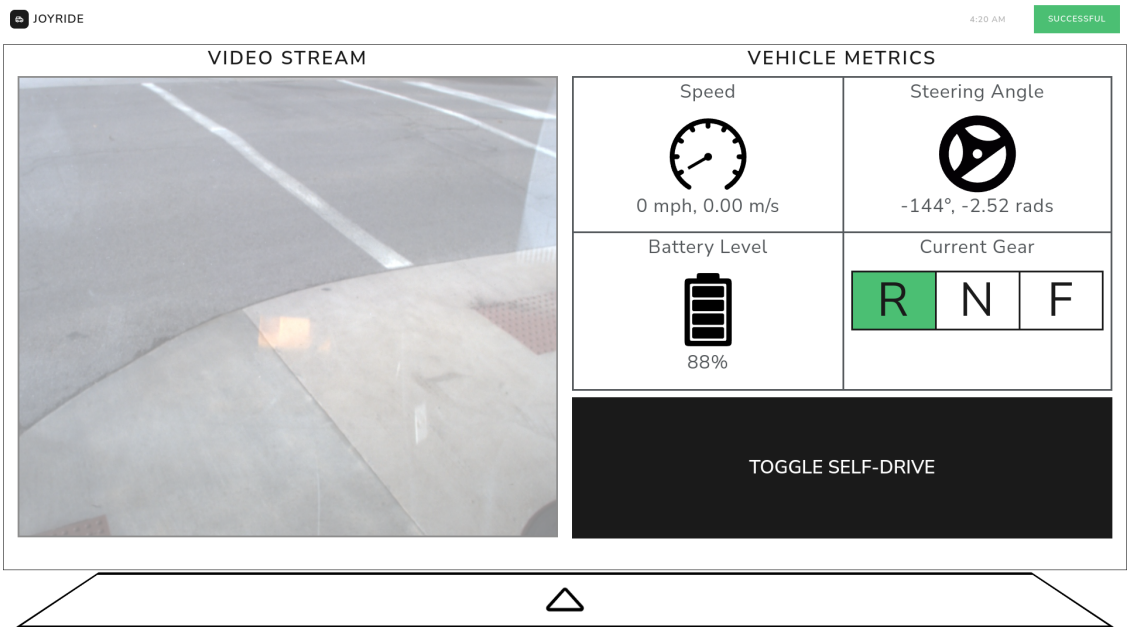


Figure 8: Current Dashboard Page

2.3.1.2 Commands

The Commands page is used for the unit testing section of the competition. Each of the unit tests must start with a single button push. Each of the unit tests will have a corresponding button that can be pressed to begin the test. A log will also be displayed that will detail the various steps as they are happening along with any additional information pertinent to a given test. This makes it simple for beginning tests, and for comparing step-by-step what the vehicle is doing versus what the vehicle is supposed to be doing. These unit tests are broken down in detail in appendices A and B.

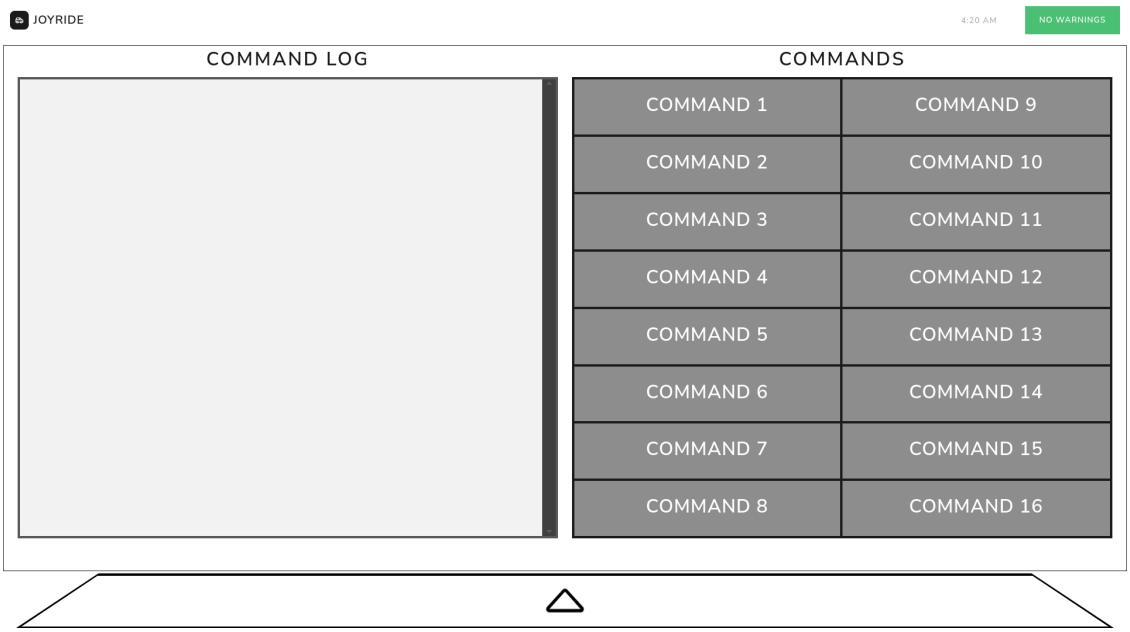


Figure 9: Current Commands Page

2.3.1.3 Sensor Data

The Sensor Data page is used to visualize the incoming data from the ROS network. The page contains a video stream and three graphs. The video stream will be similar, if not identical, to the video stream on the dashboard where it will have multiple streams that the user can switch between. This will let the user easily access any visual data such as lane tracking and object detection. The three graphs included on the page utilize touchscreen controls to make them modifiable. The user is able to drag and drop the data that they want to view. Furthermore, multiple data sets can be displayed on a single graph for quickly comparing data.

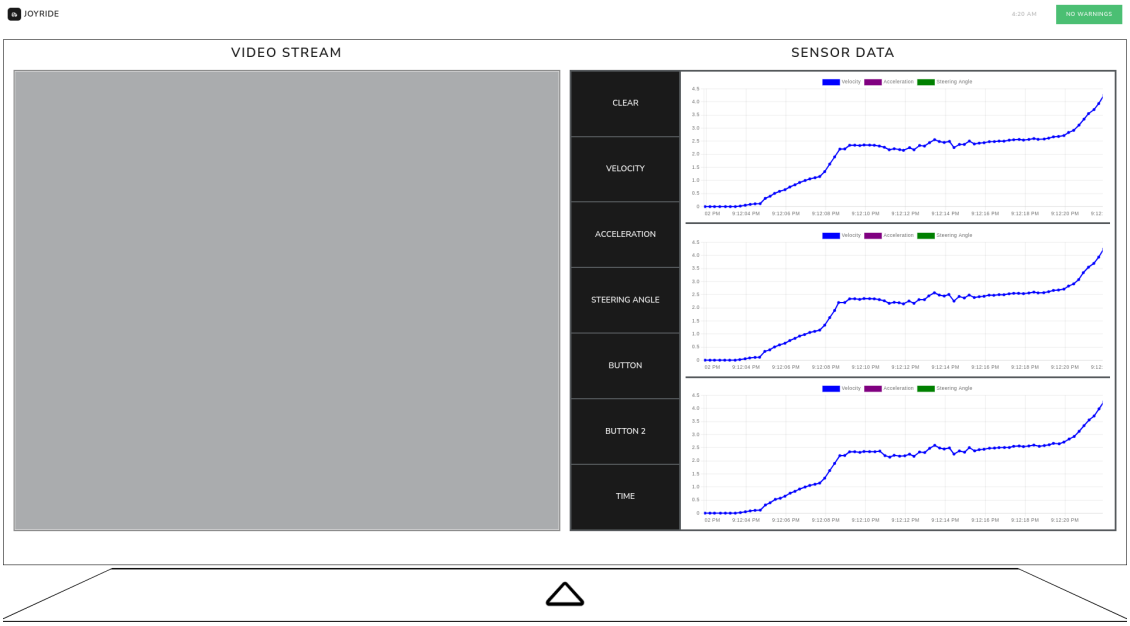


Figure 10: Current Sensor Data Page

2.3.1.4 System Health

The System Health page provides a comprehensive overview of the ROS network, displaying the status of every node on the system. Each node is listed and color-coded according to its status. A healthy node is highlighted in green, while a node with a warning is shown in yellow. A node with an error is flagged in red, providing an at-a-glance understanding of the system's health.

The Health Monitor, explained in the System Diagnostics section, feeds the GUI with the status of each node. Additionally, the page features a log that records messages received on the 'rosout' topic. For efficient log navigation, messages are timestamped and color-coded based on their severity level (info, debug, warning, or error).

The log can be scrolled using the touchscreen and has automatic start and stop scrolling features that activate when the user manually scrolls. Moreover, the log is written to a local file, ensuring log persistence throughout a session and enabling users to save and access previous log data.

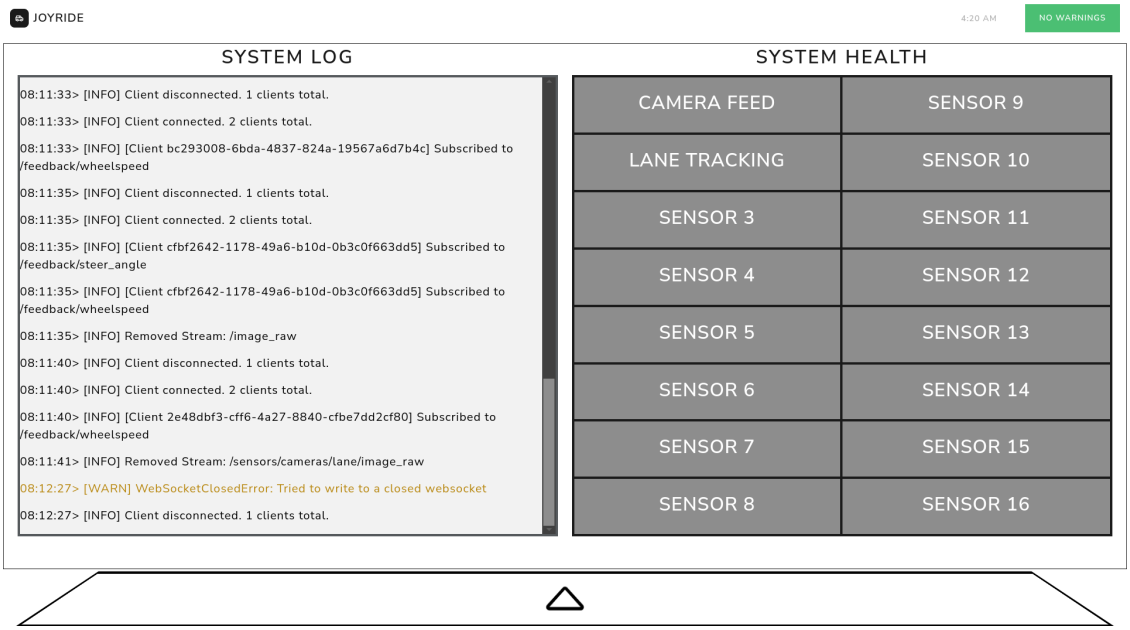


Figure 11: Current System Health Page

2.3.1.5 Waypoints

The Waypoints page has not yet been implemented but will be used for setting the destination that the vehicle will drive to. The ability of the vehicle to navigate to waypoints will be evaluated during the competition. The page will contain a map that the user can create waypoints on by pressing on the desired location on the map. The map is generated using leaflet[8], an open-source JavaScript library for mobile-friendly interactive maps. Importantly, leaflet has an offline fork so maps can be downloaded and used without the need to maintain an internet connection when using the vehicle. There will also be a text field to input the latitude and longitude coordinates of a desired destination for cases where precision is necessary. In addition to setting the destination, the user will be able to set a desired final orientation of the vehicle using a radial menu.

2.3.1.6 Navigation Bar

Each of the pages share a navigation bar at the top of the screen. This navigation bar is used to always display certain information regardless of what page is currently in view. Currently, the navigation bar shows the current time and an overall system status indicator. The overall system status indicator will change color and text depending on if the system is in a healthy, warning, or error state. When in a healthy state the indicator is green, yellow in a warning state, and red in an error state. Miscellaneous indicators will be added to the navigation bar to display additional information about the vehicle. For example, if the blinkers are on, an indicator on the navigation bar will show a recognizable arrow icon much like in a standard car.

[8]https://leafletjs.com/

2.3.2 System Diagnostics

Monitoring the status of the vehicle and all of its systems is paramount for discovering and diagnosing any failures in the vehicle and maintaining safety while it is in operation. This is done by creating a 'System Health' topic that each of the various ROS nodes will publish a custom 'Heartbeat' message to. A 'Health Monitor' node can then subscribe to the Health topic and use that information to determine whether the system is in a 'Healthy', 'Warning', or 'Error' state. The Health Monitor will then publish to a 'System Status' and 'Enable/Disable' topic with a message indicating the state of the system. The GUI subscribes to the System Status topic and displays the state of the system at all times at the top of the interface. The GUI publishes and subscribes to the Enable/Disable topic. If the user presses the button to enable autonomy, the GUI publishes to the topic to request the system to prepare to enter autonomy, if possible. If the Health Monitor publishes to the Enable/Disable topic that the system is not healthy, the GUI will disable the autonomy button to indicate that there is a problem that must be addressed before autonomy can be enabled and it will remain disabled until the Health Monitor publishes to the topic that the system is healthy. A CAN node within the ROS network also subscribes to the System Status topic and will hard disable autonomy in the event that the system is in an unhealthy state. This CAN node publishes to a 'Drive-By-Wire Feedback' topic that the Health Monitor subscribes to further monitor status.

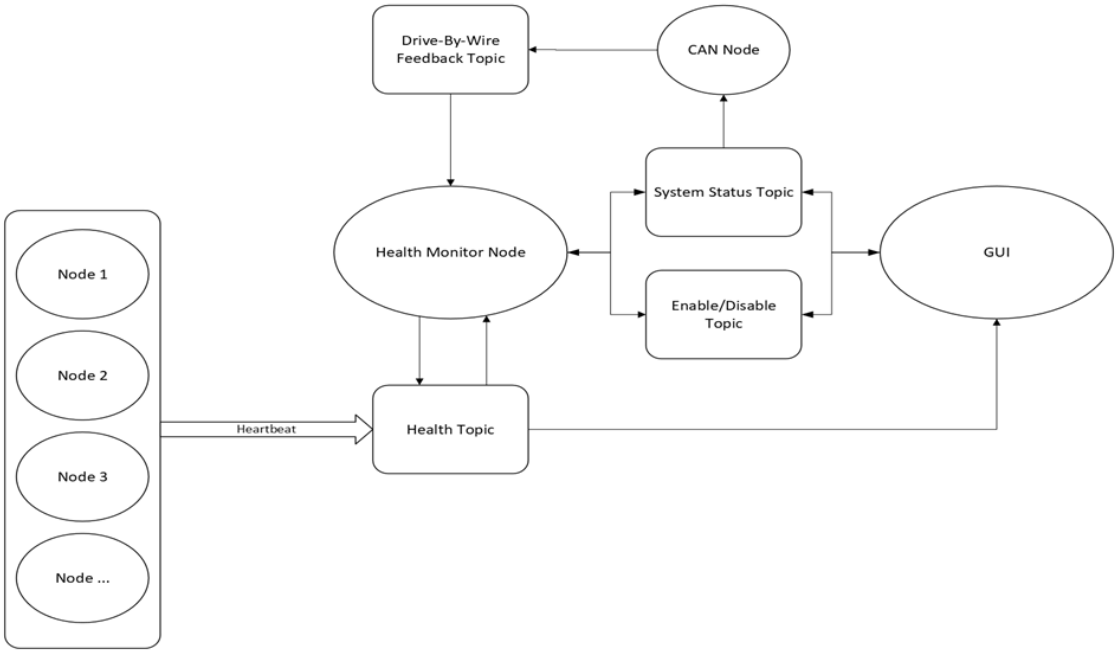


Figure 12: System status monitoring ROS topology

2.3.3 Local and Wireless Local Area Networks

The vehicle has an onboard local area network (LAN) that allows ethernet-based sensors and communication. These include the Blackfly cameras and LIDAR. A wireless local area network (WLAN) allows for remote system access through Wi-Fi. The WLAN is hosted with an onboard

access point[9] mounted on the roof of the vehicle cab. Each network-accessing device will have a statically-assigned IP address from a DHCP server hosted on the main PC. The PC connects to the network using a 4-port PCI Express power over Ethernet (POE) card.[10]

2.3.4 Controller Area Network Bus

Communication between the various onboard computers is handled using a controller area network (CAN) bus. Two CAN buses are present. CAN 0 carries messages to and from the main PC, electronic power steering, and custom PCBs. The other network, CAN 1, handles messaging between the main PC and vehicle's original network. The main PC has a two-port CAN interface PCI Express card[11] installed to facilitate message passing. The *python-can* package[12] allows easy interfacing with CAN from the computer. CAN ports are abstracted as network sockets, similar to traditional TCP/UDP socket-based networking.

The ROS network has a custom ROS-CAN interface node that translates between ROS nodes and CAN messages. Message types are defined in a configuration file using the YAML format. Each message has a ROS topic, whether it's publishing from CAN to ROS or from ROS to CAN, the relevant CAN ID(s) and the message name. This allows easy reconfiguration of topic names and CAN IDs in a single location.

The ROS-CAN interface reads from both CAN 0 and CAN 1 simultaneously and publishes to ROS indiscriminately. As for CAN publishing, the system will publish only to CAN0 as the GEM will not accept non-GEM messages on its network - nor is there a reason to publish there.

Due to its current structure, *the ROS-CAN node must remain closed-source* in order to respect the non-disclosure agreement between the students and Allied Motion (electronic power steering manufacturer). This could be remedied in the future.

2.4 Drive by Wire

2.4.1 Velocity Controller

2.4.1.1 System ID and Controller Design

A velocity controller is an essential component of an autonomous vehicle that is responsible for controlling the vehicle's speed or velocity. The purpose of the controller is to regulate the speed of the vehicle and maintain a desired velocity, regardless of external disturbances.

To design the velocity controller, it is necessary to analyze the system's behavior and develop a mathematical model that describes the input and output of the system. This model will then be used to design a controller that can adjust the system's behavior to match the desired output.

To develop a model for the brake system we have collected data from multiple unit step input tests at multiple input voltages using the onboard speedometer to measure our velocity.

[9] Ubiquiti UniFi 6 Long Range
[10] SIIG 4-Port Gigabit Ethernet PCIe Card with POE Intel 350
[11] Phytools PCAN-PCI Express Insert-card IPEH-003027
[12] https://python-can.readthedocs.io/en/stable/

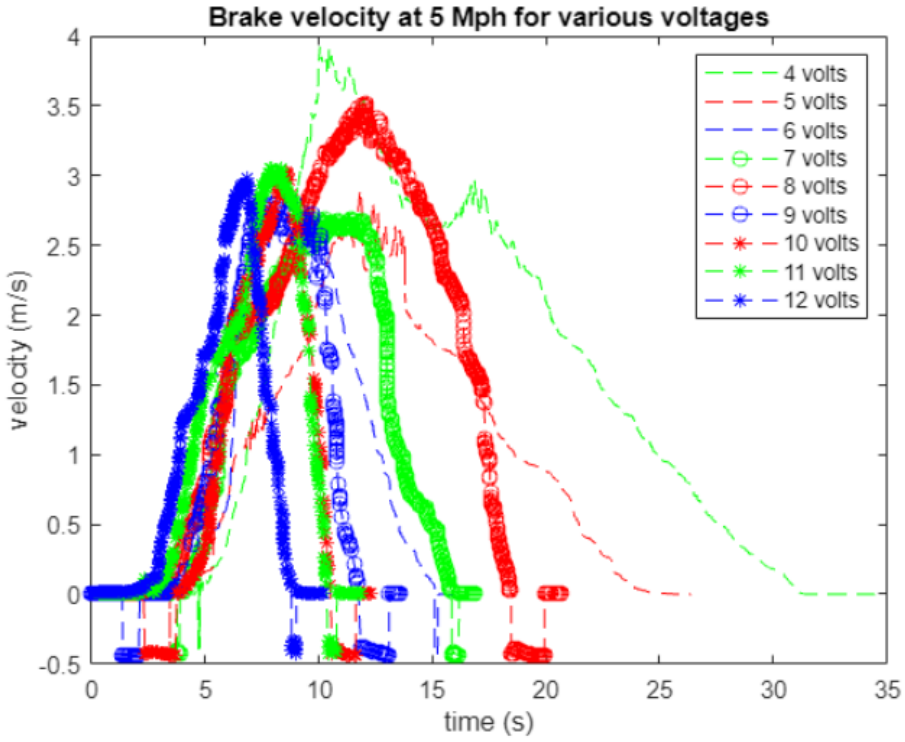


Figure 13: Velocity vs time data in response to a step input

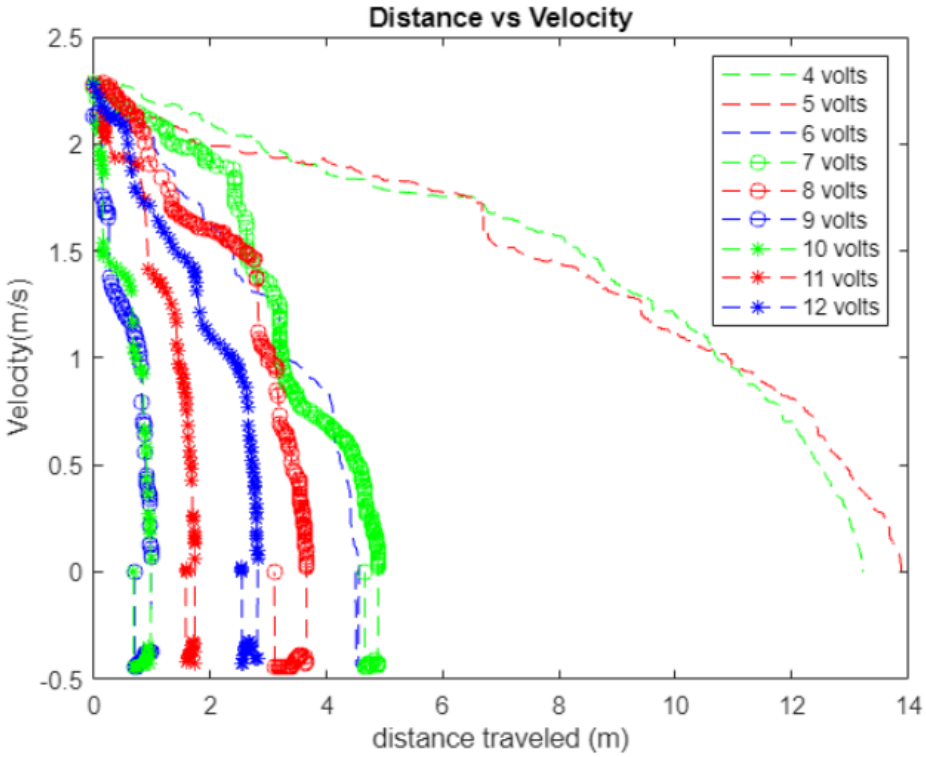


Figure 14: Velocity vs distance data in response to a step input

We then analyzed this data using MATLAB System ID Toolbox to find a first order transfer function model that best fits our gathered data. Using system ID Toolbox we were able to generate models that fit the data from particular step inputs, but could not accurately describe the data collected during step inputs of other magnitudes. We believe this is a consequence of using the onboard speedometer which does not output velocity data at a consistent rate. The speedometer also returned negative data values when coming to a stop. To improve our transfer function models using our currently available data we used interpolation to re-sample our data at consistently spaced intervals.

This improved our individual test model fits but did not create a model that could fit other test inputs. In the future we intend to improve our system transfer function models by collecting more consistent and accurate velocity data using wheel encoders. To create a model for the acceleration of the cart we plan on following a similar methodology as identifying the brake system model.

The velocity controller takes in the desired velocity V_{ref} as input and processes it using the transfer function equations that were derived from the brake and throttle tests. The output of the transfer functions will be used to adjust the throttle and brake commands to the vehicle's engine, so it can accelerate or decelerate to the desired speed. In order to maintain the desired velocity, the velocity controller uses a feedback loop to continuously monitor the vehicle's actual velocity and compare it to the desired velocity. Deviations from the desired velocity will result in an error signal, which will be used to adjust the throttle and brake commands to correct the vehicle's speed.

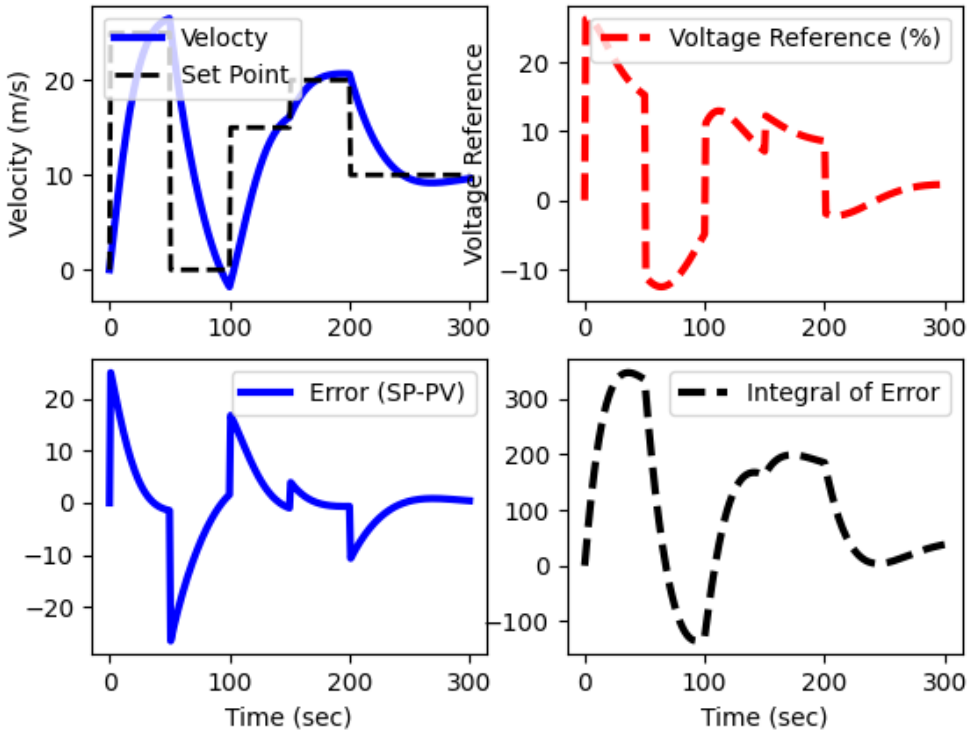


Figure 15: Simulation of controller with velocity set-points

2.4.1.2 Software Implementation

We are implementing a PID controller using feedback control in C++. C++ can handle complex mathematical calculations and real-time data processing efficiency as well as give low-level memory access. Also, the previous semester's team had programmed the Drive Controller in C++ so to preserve consistency across our software, C++ is an ideal choice.

The program first initializes the constants needed for the vehicle model, including the drag coefficient, air density, vehicle cross-sectional area, rolling resistance, and mass. The vehicle model is then defined as a function that takes in the current velocity, time, throttle input, and load as inputs, and returns the change in velocity over time using the vehicle dynamics equations. The main function of the program then runs a loop over a specified time period and a number of time steps. The loop computes the current error between the current velocity and the desired set point, integrates the error over time to obtain the integral term, and computes the control input (throttle) using the PI algorithm. The program limits the throttle to a maximum of 100% and a minimum of -50%, with corresponding adjustments made to the integral term if the limits are exceeded.

2.4.2 Drive Controller

A drive controller core software is the central component that controls the functionality of a drive system. It communicates with the host system to interpret commands and convert them into control signals for the drive hardware. This software includes the following features: Safety Manager, Steer Control, Velocity Control, and Gear Control.

2.4.2.1 Overview

All Drive Controller low-level software is written in C++. The Drive Controller Software is single-core. It contains the following functions:

- **Main**
- **Velocity Control**
- **Gear**
- **Steer**
- **Safety Manager**

Each function is communicated via CAN messages sent by an onboard computer through an interpreter. The architecture relies on a priority hierarchy to split up the time the OS runs each function.

Main: The main function in an autonomous drive controller software is the central component that controls the execution of the other functions. It acts as the interface between the OS and the sensor inputs. It determines how to distribute processing time to the various functions and manages the prioritization of functions based on their priority levels.

The main function performs the following tasks:

1. Receiving sensor input data
2. Allocating processing time to each function
3. Determining the delay time for functions that need to wait
4. Managing the prioritization of functions
5. Making real-time decisions about how to control the vehicle's motion

The main function is crucial in ensuring that the autonomous drive controller functions as intended, making real-time decisions about how to allocate resources and process sensor data to control the vehicle's motion.

Velocity Control: As discussed in the previous section. The velocity controller is an important part of the overall control strategy for the drive system and plays an important role in precise and efficient motion control.

Gear: The gear controller is responsible for detecting the current gear position of the vehicle and handling commands to switch gears as desired. Gears: *Neutral, Forward, Reverse, Reset*

Steer: The Steer Class interprets steering angles given as input into CAN commands and sends them to the electronic power steering (EPS) system, making it turn the desired number of degrees to the right or left with a speed cap of 0.75 revolutions per second. It monitors the current position of the steering wheel and reads user torque feedback. The three states of the Steer Class are: *Disabled, Idling, and Rotating*

Safety Manager: The Safety Manager is responsible for monitoring various pins and determining if the autonomous mode should be emergency stopped. Some of the key functions and responsibilities of the Safety Manager include:

- Monitoring Ignition Pin
- Emergency Stop (E-Stop)
- Electronic Brake (E-Brake)
- Manual Braking
- Charging
- Gear in Neutral
- Speed Cap *(IGVC rules: 5 mph max speed)*
- Steering Input:

In the event that any of these conditions are considered unsafe, or any other safety-critical events occur, the Safety Manager will immediately execute an emergency stop to bring the vehicle to a safe and controlled stop. This provides a fail-safe mechanism that can be triggered in emergency situations.

2.4.3 Accessory Controller

The accessory controller will function similarly to the drive controller software-wise. Several individual threads will run in parallel using the STM board's RTOS implementation. Each thread will communicate with the others through message queues, event flags, and thread flags. Because the accessory controller is not responsible for safety-critical operations, its code is allowed to be more flexible with timing.

Generally it will be writing digital output signals to activate lights. It will communicate on the same CAN bus as the drive controller and onboard PC. A steady heartbeat and frequent status reports will be provided as diagnostics to display on the GUI.

3 Electrical Design

The vehicle has many electrical components within that are used not only to enable autonomous control of the vehicle but to create enhanced safety features and data collection. Work on custom Printed Circuit Boards (PCB) started August of 2022 and the design for this semester will mark the third iteration of the current PCB architecture as seen in Figure 16.

The use of custom PCBs is an alternative to spending tens of thousands of dollars on a professionally designed and installed Drive by Wire system. Having a custom PCB not only ensures a greater educational outcome but an increased control of what types of data the custom Drive by Wire system has access to but also how the data is processed and passed along to other subsystems within the vehicle.

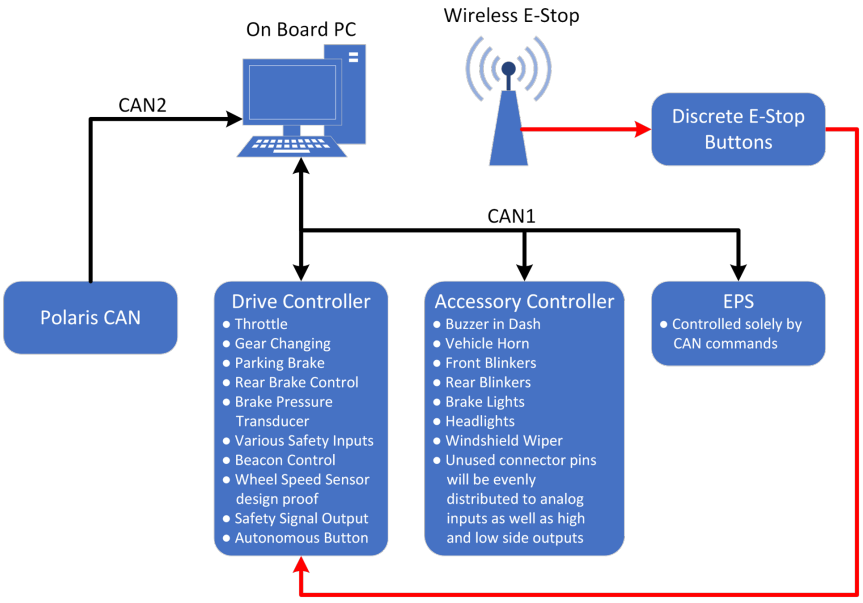


Figure 16: PCB Architecture

3.1 Drive Controller

The Drive Controller PCB is used to control functions that are directly related to the enabling of autonomy to the vehicle, as well as any sensor data that is considered important for safe vehicle operation. The Drive Controller will be enclosed in a large Armor IPX 48 pin enclosure as seen in Figure 17.

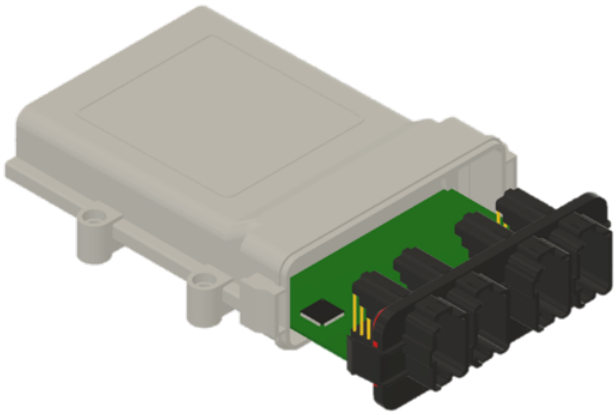


Figure 17: Drive Controller Form Factor

3.1.1 Driver Feedback Capabilities

A ring lit LED push button is installed into the dash that is used to enter autonomous mode and once the button has been pressed and the system has entered an autonomous state the ring light around the outside of the button will illuminate green, indicating to the users of the vehicle that an autonomous state has been entered. There is also a RGB LED indicator installed into the dash that is used to represent the activity of the PCB and act as a quick debug light that can present error codes in the form of light/color patterns.

3.1.2 Device Control Capabilities

The Drive Controller handles many types of signals and contains various hardware integrated logic sequences to ensure the system is always operating in a safe environment and has stable data being collected. The controller is capable of switching analog signal sources to turning on and off external relays.

3.1.2.1 Signal Spoofing

In order to control the throttle and gear control signals an analog switch is used to switch between the factory signals and the signals generated by the Drive Controller

3.1.2.2 Braking Systems

The Drive Controller houses three braking systems. The first system is for the parking brake which consists of an H-Bridge to engage the brake. The remaining two systems are for rear braking, first being a system to generate a 0-12V analog signal that can control a electric over hydraulic trailer brake system, and second being an additional H-Bridge to control a linear actuator that can be used to control a master cylinder in the rear of the vehicle.

There will also be two brake pressure transducer inputs, one for the front and one for the rear brake system. These sensors will be used to ensure the systems are seeing an increase of pressure when

the brakes are said to be applied.

3.1.2.3 Wheel Speed Sensors

Each rear wheel on the vehicle will be equipped with a hall effect quadrature encoder that will be used to determine the vehicles wheel speed and if it's moving in a forward or backwards direction. Having wheel speed data is also another data point that can be used to ensure the vehicle is coming to a stop when the brakes are said to be applied. This data can also be used to determine if a wheel is locked up or skidding on pavement when the brakes are applied and the vehicle is not stationary.

3.1.2.4 External Relays

The Drive by Wire system is powered from a relay that is turned on when the vehicles ignition is powered. In order for the On Board Computer to properly shutdown the computer must have a way of knowing the vehicle is off with out losing power.

To power the system after the vehicle is turned off an additional relay will be placed in parallel with the existing ignition relay. The secondary relay will be controlled by the Drive Controller and this gives the ability to the Drive Controller to keep itself powered once the vehicle is turned off without disturbing any factory power rails.

Now that the Drive Controller can keep itself powered it can now also keep the relay powering the 48V inverter turned on resulting in the on board computer staying powered after the vehicle is shut off. The Drive Controller can now send a shutdown signal to the on board computer and then properly shut down the entire Drive by Wire system.

3.1.3 Future Proofed Design

The Drive Controller has various unused features built in that can be used by future teams. These items are things such as front brake system pressure transducer input, and a second rear brake control system.

The front brake pressure transducer can be added to the vehicle to create a relationship between the front and rear braking system while the vehicle is in manual operation mode. This feature would be desired to improve the vehicles manual driving performance while coming to a stop since the system currently relies purely on front brakes. Creating a relationship between the two systems would allow for the Drive Controller to see the front brake systems pressure and match the rear brake system to that pressure to allow the user to have four wheel braking.

3.2 Accessory Controller

The Accessory Controller PCB is used to control functions that are not important to the vehicle's safe operation. These items mostly consist of lights and indicators or various other items which are required for the vehicle to operate in accordance with state and competition regulations. The Accessory Controller will be enclosed in small Armor IPX 24 pin enclosure as seen in Figure 18.

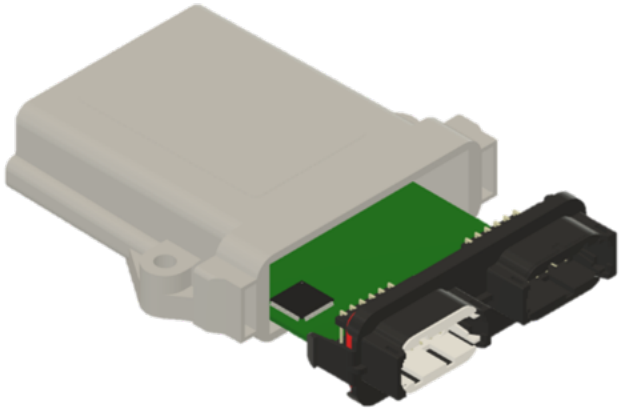


Figure 18: Accessory Controller Form Factor

3.2.1 Driver Feedback Capabilities

A piezoelectric buzzer is installed underneath the dash and the Accessory Controller will activate the buzzer to indicate when the vehicle enters and exits autonomous mode. There is also a RGB LED indicator installed into the dash that is used to represent the activity of the PCB and act as a quick debug light that can present error codes in the form of light/color patterns.

3.2.2 Device Control Capabilities

Various accessories that are required to stay in compliance with state and competition guidelines are listed below. All of them items are independently controlled by either a low side or high side self protected output. A self protected output is very important, this type of output means it can with stand many unwanted scenarios such as; over voltage, over current, over heating, and short to ground/12V.

- Headlights
- Blinkers
- Brake Lights
- Horn
- Windshield Wiper

3.2.3 Future Proofed Design

The Accessory Controller has unallocated pins that are designed to be used in the future if the need arises. The design proof is delivered through making the unallocated pins a mixture of input at outputs that can later be connected to any type of device or sensor.

Extra pins that are designated as inputs will have the ability to read an analog or digital input from 0V to 12V with a frequency of 12kHz. Outputs will consist of self protected high and low side outputs that can be used to power a 12V device or control a relay coil.

4 Testing and Verification

4.1 Overview

There is a few different categories of testing that will occur for our project. The most notable categories are split up into mechanical, software, and competition tests. As a baseline for each of these categories, we want to ensure that the manual operation of the car is not altered throughout the project. The goal is to have the car fully operational by an onboard operator, just as it was from the factory, as well as successfully navigate the course at the IGVC competition. Aside from keeping the manual operation consistent with its original state, there will be extensive testing over all of the aspects that the team has worked on this semester. The competition tests are in place to validate the preparedness for the IGVC course. Correspondingly, the mechanical and software tests are in place to validate the functionality of the individual projects our team has worked on throughout the semester. All of our structured car performance tests will be done at the DML (as of now). All supplies that are needed, stop signs, traffic cones, potholes, etc., are listed and will be purchased if they are not already in our possession.

4.2 Competition

The competition tests will be broken up into a couple of different categories. The first category will be safety/qualification tests. These are set up to replicate the minimum cut-off to compete in the competition. The first requirements that the car will have to meet are the general safety unit tests. These include the validation of the E-Stop(manual/wireless), speed limit, lane boundaries(left/right), object detection, and reverse. Similarly, the qualification tests include validating the E-Stop(manual/wireless), lane keeping, and turning(left/right). The tests our team will run will be accurate representations of what can be expected at the competition in June. Upon satisfying these requirements, we will move onto the functional tests. The functional tests are related to what can be expected for the overall course navigation. The seven categories of the functional tests are traditional machine vision, traffic sign, intersection, parking, VRU (vulnerable road user), curved road, and other. The details of all of these tests will be broken down in the appendix. We do not foresee any reason that we will not be able to perform all of these tests. The only exception is that we will not be able to replicate the course in its entirety due to space limitations at the DML, where we plan to conduct our tests.

4.3 Mechanical

Many different components have been designed already this semester by the mechanical team. With varying importance, each design will be tested to ensure it performs as needed. A few of the current designs with the highest priority are the brakes, wheel encoder, and the forward-facing cameras. The brakes were completely redone at the beginning of this semester and have proved to be problematic for past semesters. Therefore, to guarantee the continued success of our braking configuration, we will be closely monitoring the fluid reservoirs, line joints, banjo bolts, and hydraulic cylinders to ensure there are no leaks, air infiltrating the system, or loose connections.

The rear wheel encoder is mounted to a test bench as seen in the mechanical design section to validate the logic behind the design. Different design techniques have been heavily considered

so that the gear profile can be accurately read. For manufacturing purposes, the first gear design that we are testing is made out of acrylic and press-fit slots for metal round stock. This design allows for quick manufacturing for prototyping as needed. Due to the intricate nature of the part, the current conclusion is that the laser-cut acrylic will produce more accurate parts quickly than needing to repeatedly waterjet the parts out of metal. The testing setup was described in detail in the mechanical section and can be seen again below. Upon mounting the encoder to the hub, the necessary design alterations will be made as the reliability of the current components gets analyzed.

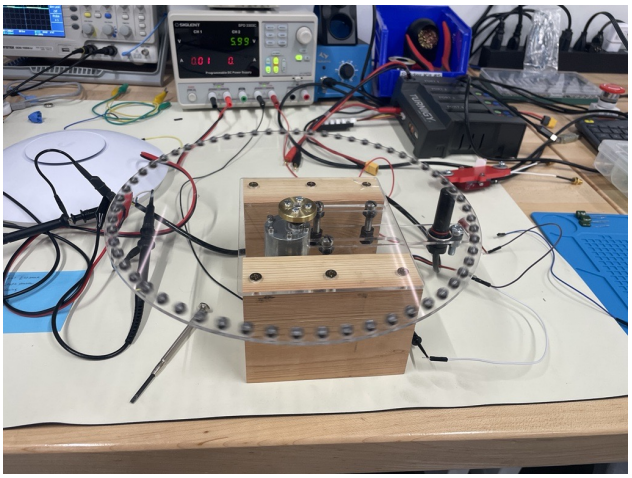


Figure 19: Fabricated Test Bench

The forward-facing cameras have undergone multiple iterations already, and will likely continue to do so. The majority of those changes have come from the tests conducted on them to optimize their effectiveness. A couple of the main issues at hand are the FOV and reflection off of the windshield. As previously shown, the different versions that have been made have slightly changed the angles to accommodate desired FOV alterations. Additionally, the reflection issues are currently being tested for. The biggest concern is that anything on the dash will get picked up by the cameras and potentially throw off the object and lane detection. To account for that issue, we plan to test the effectiveness of a polarizing filter, lens hood, and a non-reflective dash mat. We are confident that all three will be effective, however, we have yet to determine the best solution. Picture below is an example of what polarized sunglasses would do to the glare issue. Based on these results, it is safe to say the filter is worth testing. In addition to the reflective issues, the cameras will also be tested in different lighting and light rain conditions. The methods of conducting these tests will be operating at different times of the day and using a spray bottle to simulate the effects of different precipitation conditions.

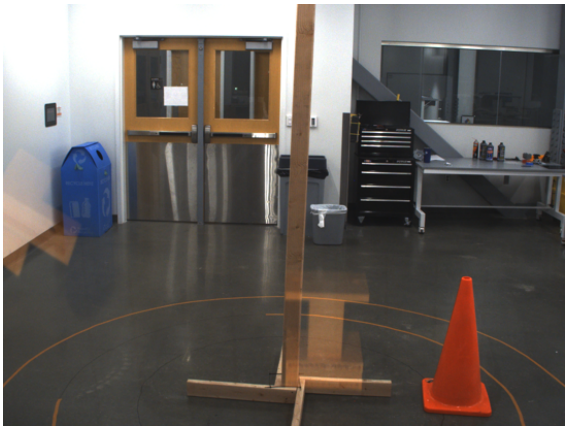


Figure 20: Without Polarizer Low Light

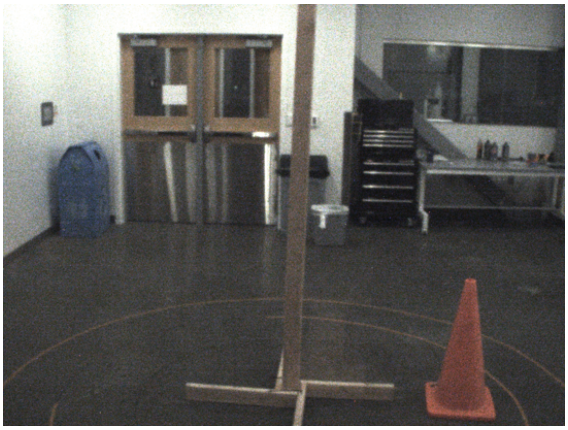


Figure 21: With Polarizer Low Light

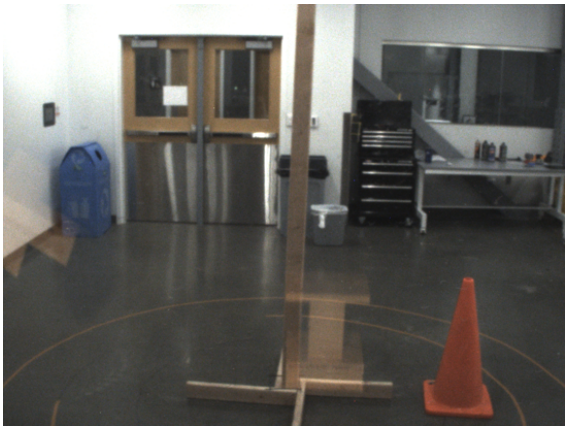


Figure 22: Without Polarizer High Light

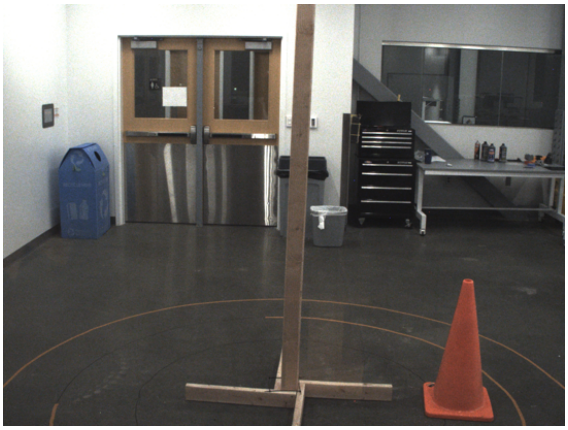


Figure 23: With Polarizer High Light

4.4 Software

The main testbench for testing and verifying the implemented software is through IGVC's functional testing displayed in its rules. In Appendix A and Appendix B, the software for our autonomous vehicle must complete these tests in order to participate in the main competition. Recreation of these testing conditions is created indoors for preliminary testing. Afterward, the final testing and verification process will take place outdoors to detect and fine-tune any issues and bugs for the autonomous vehicle before the competition. Upon validating each software component, the stress testing for the combination of all programs will proceed to confirm stable functionality.

5 Costs

The team has $10k available from senior design funds, and an additional $8k from a donor to spend on travel and sensors.

Due to previous semesters, this semester's spending is relatively light. The largest expenses are the 3D lidar (Blickfeld Cube 1 Outdoor purchased from ClearPath Robotics) and travel. Travel has yet to be fully ironed out, so the team allocated a generous $7k.

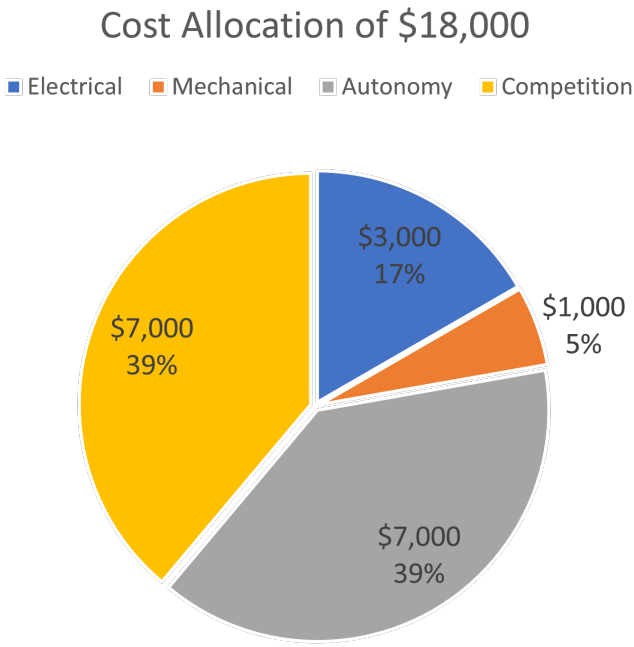


Figure 24: Cost breakdown

Category	Description	Estimate	Spent
Electrical	Two PCBA Revisions	$2000	
	Wire Harness Materials	$500	$379
	Sensors & Misc	$500	$490
Mechanical	Brake Repairs	$500	$95
	Brackets & Miscellaneous	$500	$131
Autonomy	3D Lidar	$5,000	$4,475
	Radar	$2,000	
Competition	Supplies, Transport & Hotels	$7,000	$500
Total Cost		$18,000	$6,070

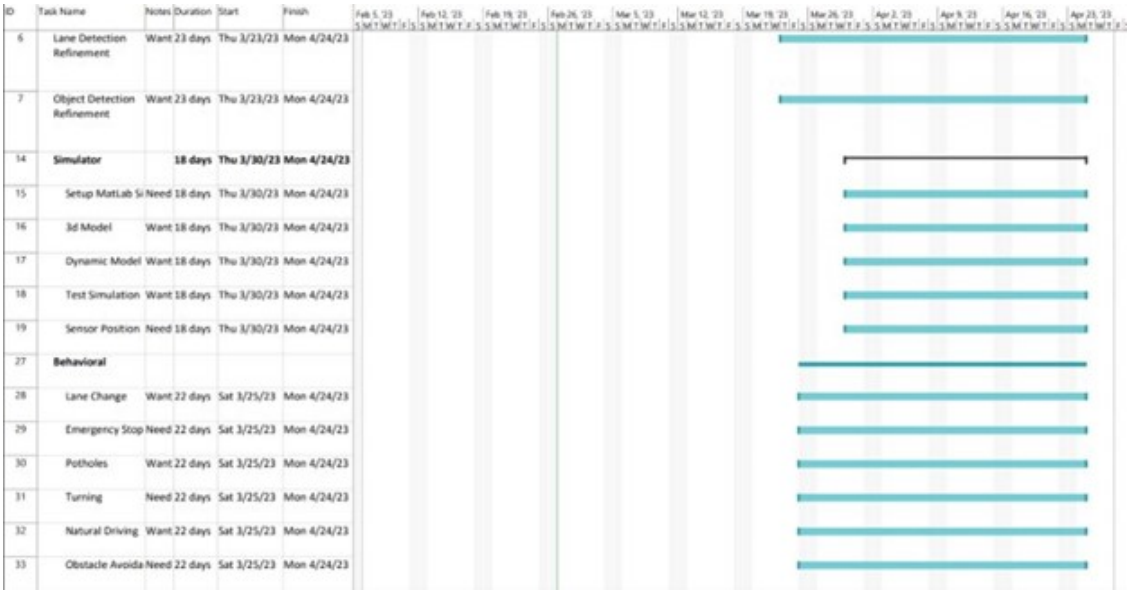
6 Project Plan

The team has found success in working backwards while generating plans. First, a testing day is identified. All potential subsystems available for testing on that day (e.g. brakes, throttle, cameras, etc.) will be scheduled to have functional prototypes completed before that date. Upon test completion, any data is analyzed and used to inform future test scheduling.

6.1 Software Plan

ID	Task Name	Notes	Duration	Start	Finish
1	**Perception**				**Fri 3/24/23**
2	Pedestrian Detecti	Need	25 days	Mon 2/6/23	Fri 3/10/23
4	Sign Detection	Need	25 days	Mon 2/6/23	Fri 3/10/23
5	Lane Detection	Need	24 days	Mon 2/20/23	Thu 3/23/23
9	**Basic Sensor Integration**				**Thu 3/23/23**
11	Cameras	Need	11 days	Mon 2/27/23	Mon 3/13/23
10	GPS/IMU	Need	15 days	Mon 2/27/23	Fri 3/17/23
13	EPS feedback	Need	16 days	Mon 2/27/23	Mon 3/20/23
12	Speedometer	Need	19 days	Mon 2/27/23	Thu 3/23/23
8	Object Detection	Need	19 days	Tue 2/28/23	Fri 3/24/23
20	**Motion Control**				**Fri 3/24/23**
22	Localization	Need	14 days	Wed 3/1/23	Mon 3/20/23
24	Local Controller	Need	14 days	Wed 3/1/23	Mon 3/20/23
23	SLAM	Need	15 days	Wed 3/1/23	Tue 3/21/23
25	Planner	Need	16 days	Wed 3/1/23	Wed 3/22/23
21	Basic Sensor integration	Need	24 days	Mon 2/20/23	Thu 3/23/23
26	Global Map	Need	18 days	Wed 3/1/23	Fri 3/24/23
3	Pedestrian Detection Refinement	Want	23 days	Thu 3/23/23	Mon 4/24/23

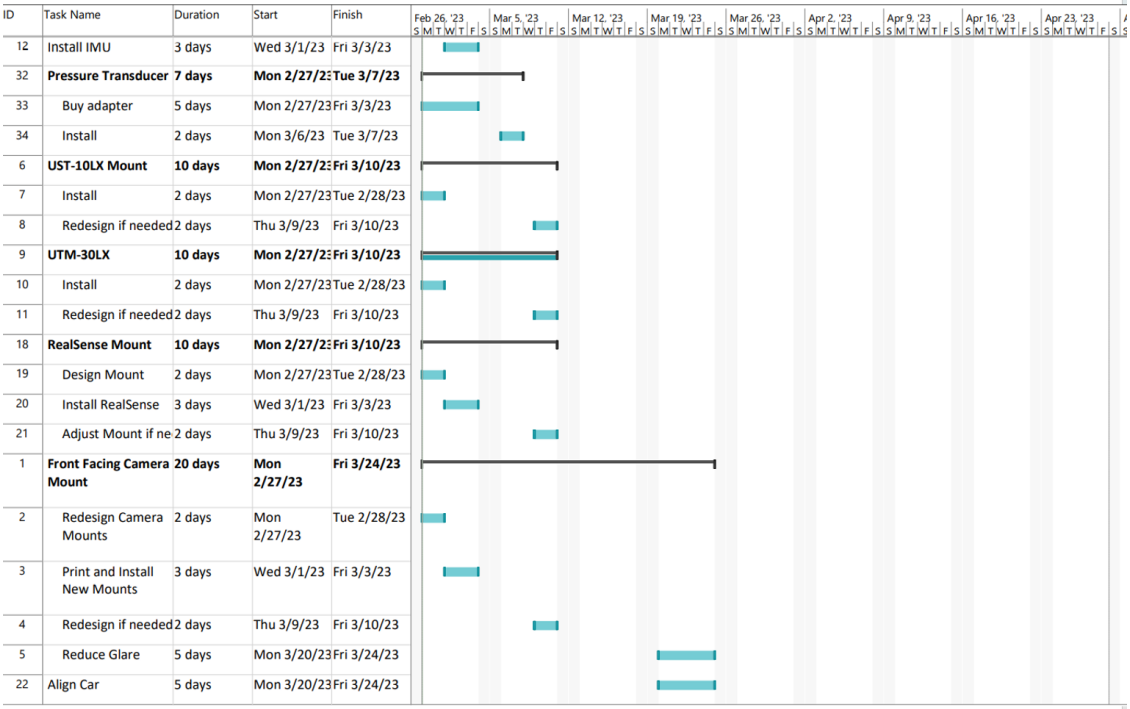
Figure 25: Software Gantt Chart



ID	Task Name	Notes	Duration	Start	Finish
6	Lane Detection Refinement	Want	23 days	Thu 3/23/23	Mon 4/24/23
7	Object Detection Refinement	Want	23 days	Thu 3/23/23	Mon 4/24/23
14	**Simulator**		**18 days**	**Thu 3/30/23**	**Mon 4/24/23**
15	Setup MatLab Si	Need	18 days	Thu 3/30/23	Mon 4/24/23
16	3d Model	Want	18 days	Thu 3/30/23	Mon 4/24/23
17	Dynamic Model	Want	18 days	Thu 3/30/23	Mon 4/24/23
18	Test Simulation	Want	18 days	Thu 3/30/23	Mon 4/24/23
19	Sensor Position	Need	18 days	Thu 3/30/23	Mon 4/24/23
27	**Behavioral**				
28	Lane Change	Want	22 days	Sat 3/25/23	Mon 4/24/23
29	Emergency Stop	Need	22 days	Sat 3/25/23	Mon 4/24/23
30	Potholes	Want	22 days	Sat 3/25/23	Mon 4/24/23
31	Turning	Need	22 days	Sat 3/25/23	Mon 4/24/23
32	Natural Driving	Want	22 days	Sat 3/25/23	Mon 4/24/23
33	Obstacle Avoida	Need	22 days	Sat 3/25/23	Mon 4/24/23

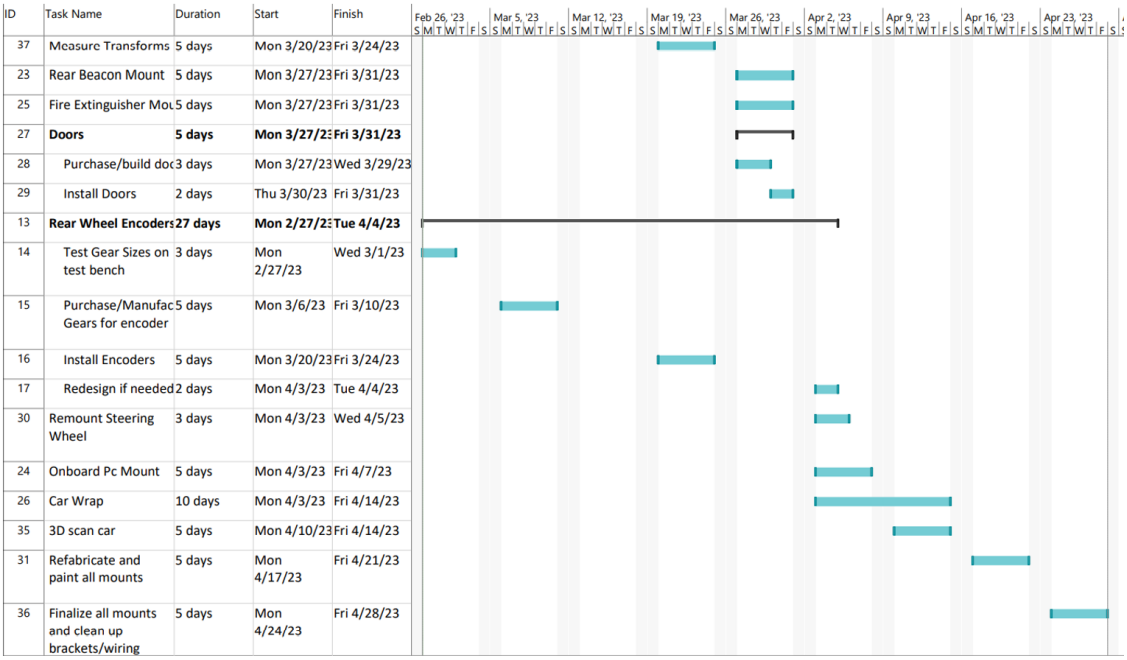
Figure 26: Software Gantt Chart Continued

6.2 Mechanical Plan



ID	Task Name	Duration	Start	Finish
12	Install IMU	3 days	Wed 3/1/23	Fri 3/3/23
32	**Pressure Transducer**	**7 days**	**Mon 2/27/23**	**Tue 3/7/23**
33	Buy adapter	5 days	Mon 2/27/23	Fri 3/3/23
34	Install	2 days	Mon 3/6/23	Tue 3/7/23
6	**UST-10LX Mount**	**10 days**	**Mon 2/27/23**	**Fri 3/10/23**
7	Install	2 days	Mon 2/27/23	Tue 2/28/23
8	Redesign if needed	2 days	Thu 3/9/23	Fri 3/10/23
9	**UTM-30LX**	**10 days**	**Mon 2/27/23**	**Fri 3/10/23**
10	Install	2 days	Mon 2/27/23	Tue 2/28/23
11	Redesign if needed	2 days	Thu 3/9/23	Fri 3/10/23
18	**RealSense Mount**	**10 days**	**Mon 2/27/23**	**Fri 3/10/23**
19	Design Mount	2 days	Mon 2/27/23	Tue 2/28/23
20	Install RealSense	3 days	Wed 3/1/23	Fri 3/3/23
21	Adjust Mount if ne	2 days	Thu 3/9/23	Fri 3/10/23
1	**Front Facing Camera Mount**	**20 days**	**Mon 2/27/23**	**Fri 3/24/23**
2	Redesign Camera Mounts	2 days	Mon 2/27/23	Tue 2/28/23
3	Print and Install New Mounts	3 days	Wed 3/1/23	Fri 3/3/23
4	Redesign if needed	2 days	Thu 3/9/23	Fri 3/10/23
5	Reduce Glare	5 days	Mon 3/20/23	Fri 3/24/23
22	Align Car	5 days	Mon 3/20/23	Fri 3/24/23

Figure 27: Mechanical Gantt Chart



ID	Task Name	Duration	Start	Finish
37	Measure Transforms	5 days	Mon 3/20/23	Fri 3/24/23
23	Rear Beacon Mount	5 days	Mon 3/27/23	Fri 3/31/23
25	Fire Extinguisher Mou	5 days	Mon 3/27/23	Fri 3/31/23
27	**Doors**	**5 days**	**Mon 3/27/23**	**Fri 3/31/23**
28	Purchase/build doo	3 days	Mon 3/27/23	Wed 3/29/23
29	Install Doors	2 days	Thu 3/30/23	Fri 3/31/23
13	**Rear Wheel Encoders**	**27 days**	**Mon 2/27/23**	**Tue 4/4/23**
14	Test Gear Sizes on test bench	3 days	Mon 2/27/23	Wed 3/1/23
15	Purchase/Manufac Gears for encoder	5 days	Mon 3/6/23	Fri 3/10/23
16	Install Encoders	5 days	Mon 3/20/23	Fri 3/24/23
17	Redesign if needed	2 days	Mon 4/3/23	Tue 4/4/23
30	Remount Steering Wheel	3 days	Mon 4/3/23	Wed 4/5/23
24	Onboard Pc Mount	5 days	Mon 4/3/23	Fri 4/7/23
26	Car Wrap	10 days	Mon 4/3/23	Fri 4/14/23
35	3D scan car	5 days	Mon 4/10/23	Fri 4/14/23
31	Refabricate and paint all mounts	5 days	Mon 4/17/23	Fri 4/21/23
36	Finalize all mounts and clean up brackets/wiring	5 days	Mon 4/24/23	Fri 4/28/23

Figure 28: Mechanical Gantt Chart Continued

6.3 Electrical Plan

ID	Task Name	Duration	Start	Finish
10	**Drive controller**	**6 days?**	**Mon 2/27/2**	**Mon 3/6/23**
12	Layout	5 days	Mon 2/27/2	Fri 3/3/23
13	Schemaic	5 days	Mon 2/27/2	Fri 3/3/23
11	PCBs Ordered	4 days	Wed 3/1/23	Mon 3/6/23
14	Accessory Controller	6 days	Mon 2/27/2	Mon 3/6/23
1	**DBW General tasks**	**11 days**	**Mon 3/6/23**	**Mon 3/20/23**
5	Install new Wireless ESTOP	2 days	Mon 3/6/23	Tue 3/7/23
2	Modify Wireless ESTOP	3 days	Wed 3/8/23	Fri 3/10/23
4	Add to DBW harness	7 days	Mon 3/6/23	Tue 3/14/23
3	Modify DBW Harness	4 days	Wed 3/15/2	Mon 3/20/23
6	**PCBs**	**20 days**	**Mon 3/6/23**	**Fri 3/31/23**
9	Build First Version of PCB	2 days	Fri 3/17/23	Mon 3/20/23
8	Software for new PCBs	15 days	Mon 3/6/23	Fri 3/24/23
7	PCBSoftware Revision(If necessary)	6 days	Sat 3/25/23	Fri 3/31/23

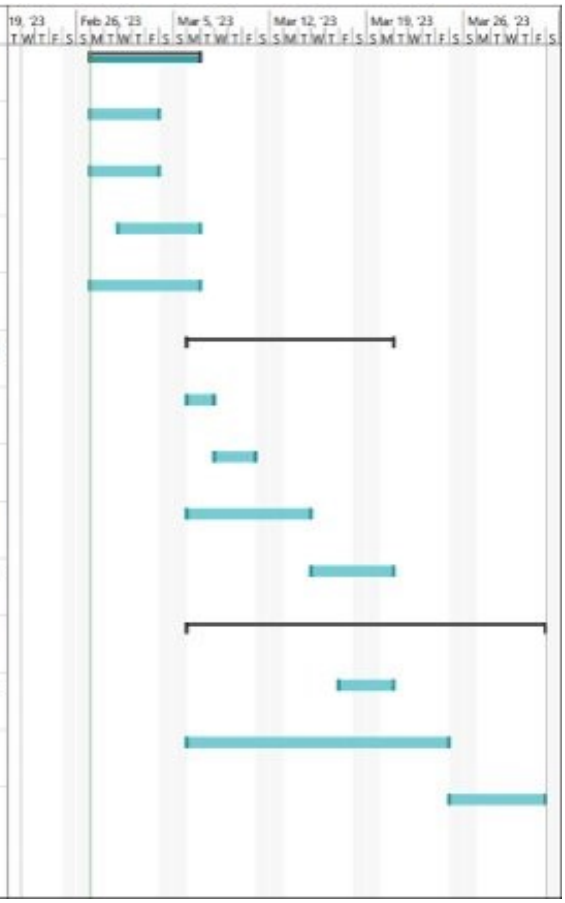


Figure 29: DBW Gantt Chart

Appendix A Safety and Qualification Tests

In order to compete in the functional and main course tests teams must complete several safety unit tests on the first day of the competition.

Test Type	Test ID	Name	# of Runs	Time	Penalty Points	Comments
Qualification	Q.1	E-Stop Manual				
Qualification	Q.2	E-Stop Wireless				
Qualification	Q.3	Lane Keeping (Go Straight)				
Qualification	Q.4	Left Turn				
Qualification	Q.5	Right Turn				

Figure 30: Qualification Test Data Sheet

A.1 Q.1: E-Stop Manual

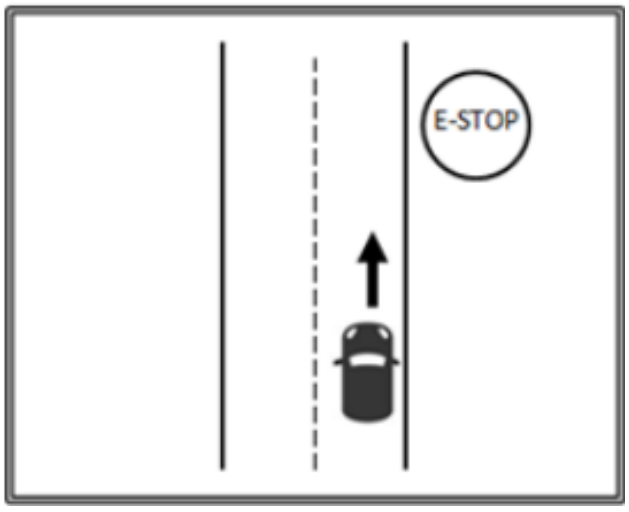


Figure 31: Qualification Testing, E-Stop Manual

For the manual E-stop tests teams are required to find the maximum distance for the vehicle to stop upon pressing the onboard E-stop. Upon stopping within a reasonable (undisclosed) distance, there will be a pass/fail consensus.

A.2 Q.2: E-Stop Wireless

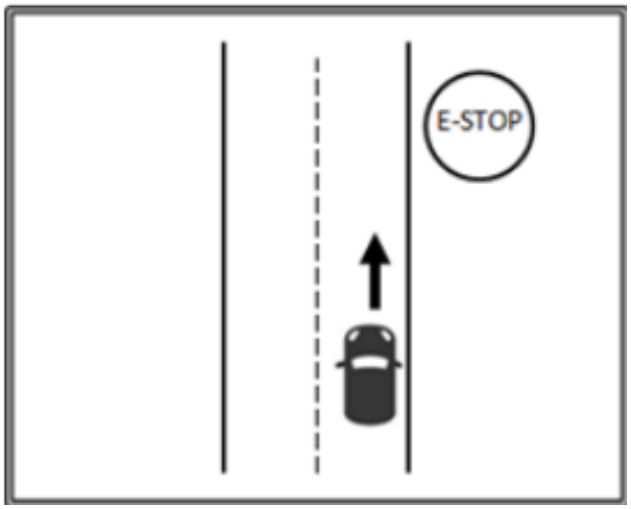


Figure 32: Qualification Testing, E-Stop Wireless

Similarly, for the wireless E-stop tests teams are required to find the maximum distance for the vehicle to stop upon pressing the wireless E-stop. Upon stopping within a reasonable (undisclosed) distance, there will be a pass/fail consensus.

A.3 Q.3: Lane Keeping (Go Straight)

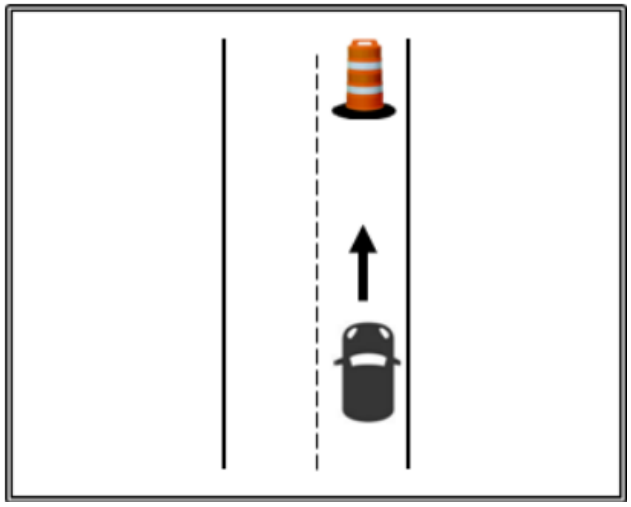


Figure 33: Qualification Testing, Lane Keeping, Go Straight

In the lane keeping test the vehicle will start at rest and accelerate towards a barrel 50 ft away. To pass this test the vehicle needs to stop within 3 feet of the barrel and keep the tires between the lane lines.

A.4 Q.4: Left Turn

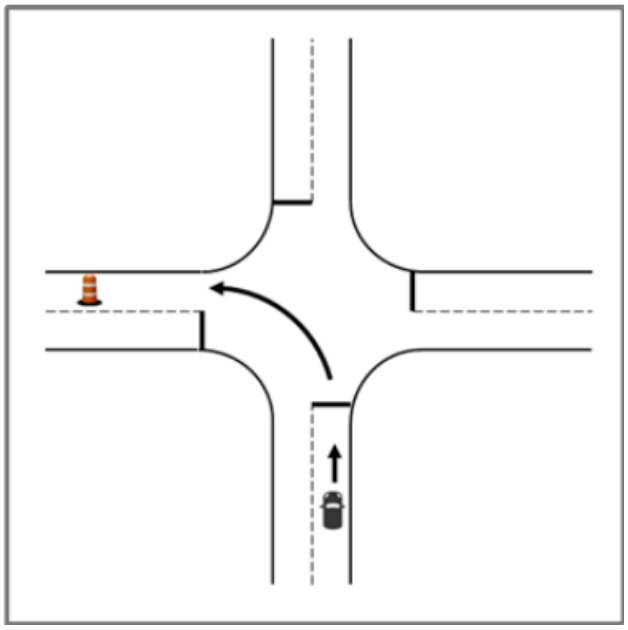


Figure 34: Qualification Testing, Left Turn

The turning tests are setup as intersection navigation. The vehicle will start stationary and need to make the necessary turn setup in the test. The car needs to navigate into the correct lane and stop within 3 ft of the barrel in the lane. This test is also pass/fail criteria.

A.5 Q.5: Right Turn

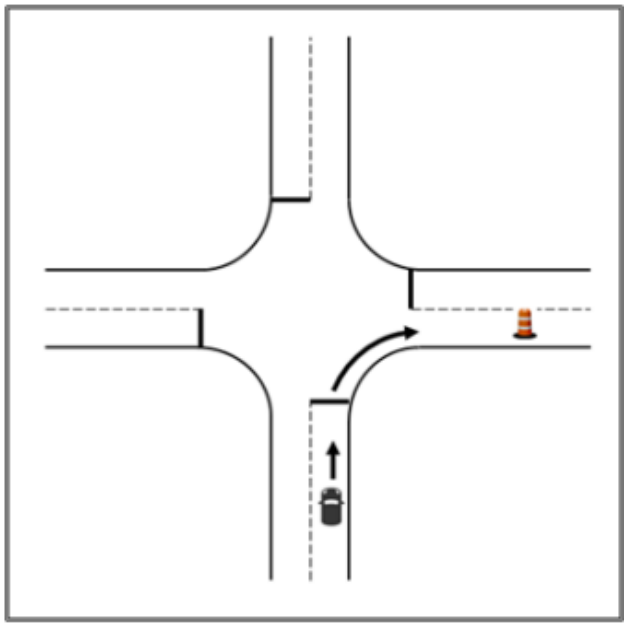


Figure 35: Qualification Testing, Right Turn

The right turn test will be an exact replica of the left turn test. The vehicle will simply need to turn the opposite direction.

Appendix B Functional Tests

Once qualification testing is completed teams will perform in a number of functionality tests. These tests are designed to assess each teams abilities in several important categories.

Test Type	Test ID	Name	# of Runs	Time	Penalty Points	Comments
I. Traditional Machine Vision Tests						
Function	I.1	White Lines Detection				
Function	I.2	Pedestrian Detection				
Function	I.3	Tire Detection				
II. Traffic Sign Tests						
Function	II.1	Stop Sign Detection				
III. Intersection Tests						
Function	III.1	Lane Keeping				
Function	III.2	Left Turn				
Function	III.3	Right Turn				
IV. Parking Tests						
Function	IV.1	Parking. Pull Out				
Function	IV.2	Parking. Pull In				
Function	IV.3	Parking. Parallel				
V. VRU (Vulnerable Road User) Tests						
Function	V.1	Unobstructed STATIC Pedestrian Detection				
Function	V.2	Obstructed DYNAMIC Pedestrian Detection				
Function	V.3	STATIC Pedestrian Detection. Lane Changing				
Function	V.4	Obstacle Detection. Lane Changing				
VI. Curved Road Evaluation Tests						
Function	VI.1	Curved Road Evaluation. Lane Keeping				
Function	VI.2	Curved Road Evaluation. Lane Changing				
VII. Other Tests						
Function	VII.1	Pothole Detection				
Function	VII.2	Merging				

Figure 36: Function Testing Scoring Sheet

B.1 F.1: Traditional Machine Vision

The first category of the functional tests is the traditional machine vision. This category consists of three tests: White Lines Detection, Pedestrian Detection, and Tire Detection. The white line detection is strictly to analyze the machine algorithms. No penalty points will be given as long as the white lines are present in the GUI interface at all times. Similarly, for the pedestrian detection, the GUI interface will need to show its ability to extract an orange blob when a mannequin is in its path. Finally, a tire will be placed in the lane to prove that the algorithms can extract the shape of a tire when it approaches one.

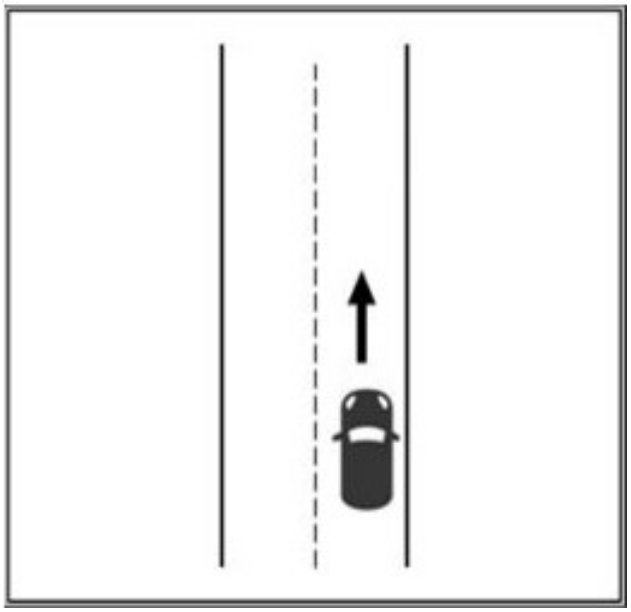


Figure 37: Machine Vision Tests, White Lines Detection

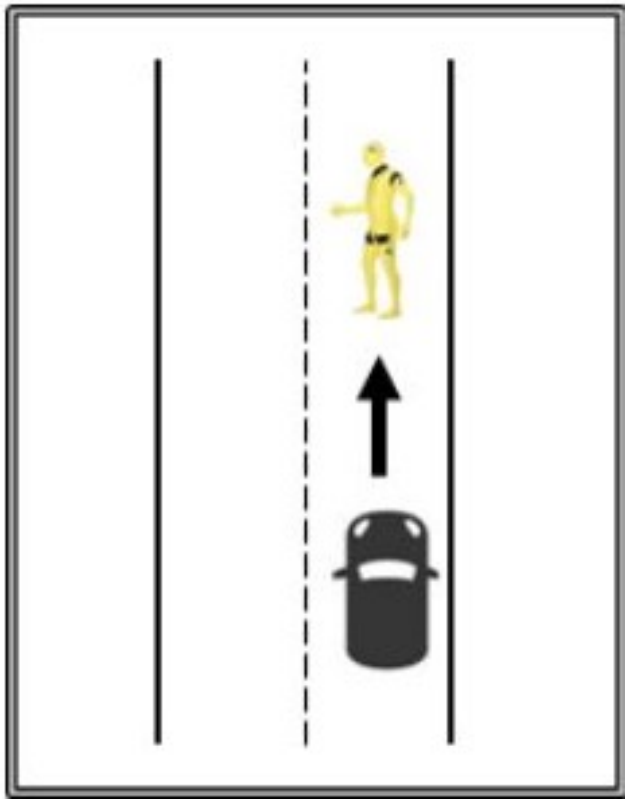


Figure 38: Machine Vision Tests, Static Pedestrian Detection

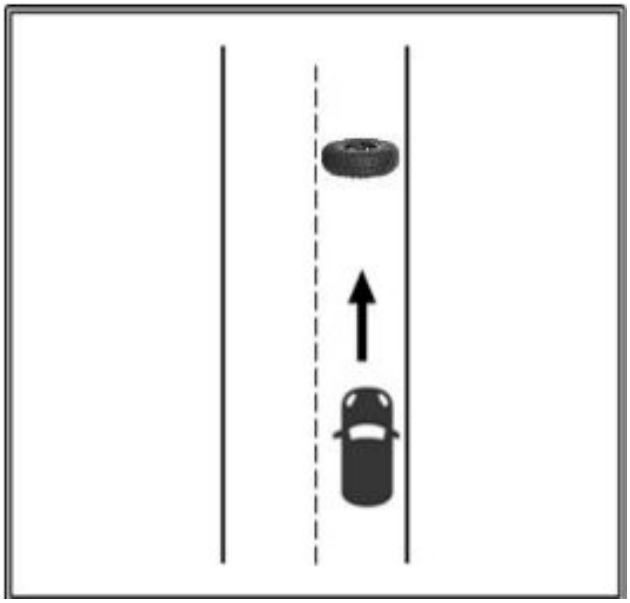


Figure 39: Machine Vision Tests, Tire Detection

B.2 F.2: Traffic Sign

This test will prove the vehicle's ability to detect a stop sign. The vehicle simply needs to prove it can categorize a stop even if there is different letters, or a small defect in the sign. Additionally, to succeed in the test, the vehicle needs to come to a stop within 2 ft from the sign.

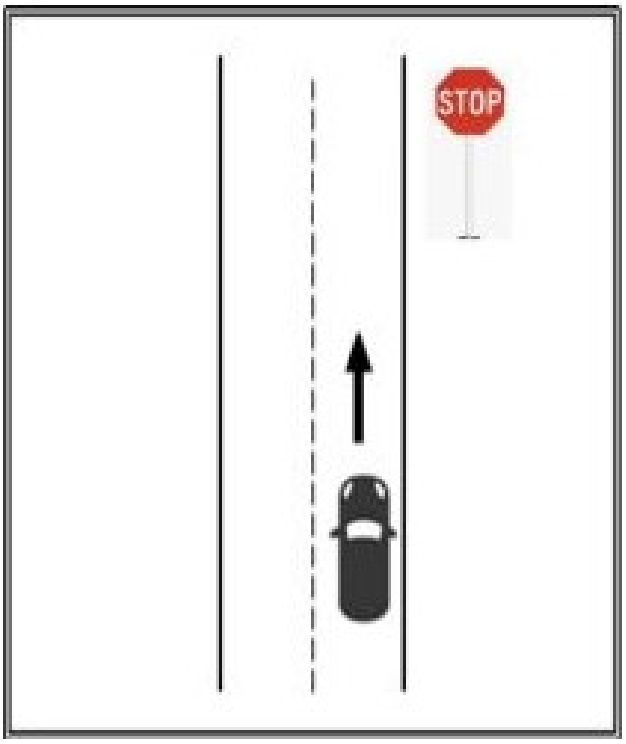


Figure 40: Functions Testing, Stop Sign Detection

B.3 F.3: Intersection

For completion of the intersection tests, the car will need to succeed in a lane keeping, and turning assessment. The purpose of the lane keeping test is to prove the car's ability to turn into the correct lane in an intersection. The first test will begin from rest at the stop sign and continue straight until it reaches a barrel placed in the lane. The turning tests will use a 'One Way' sign to indicate which direction the car should turn. It also should stop at the placed barrel, but needs to accurately navigate either turn first.

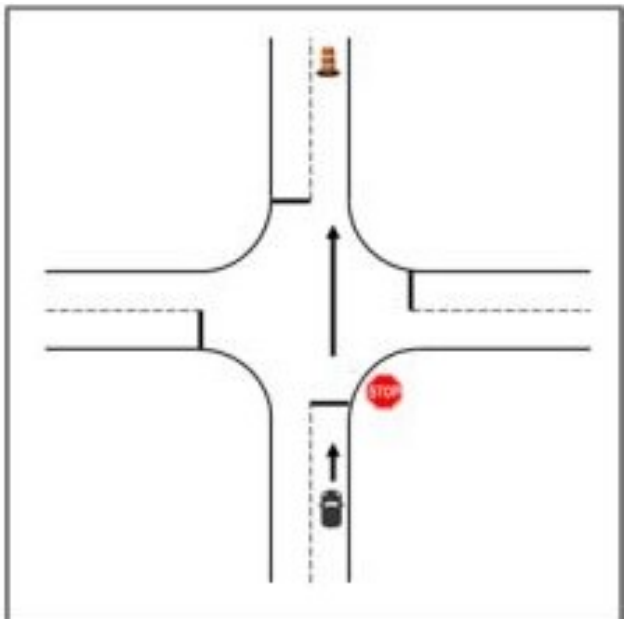


Figure 41: Intersection Tests, Lane Keeping

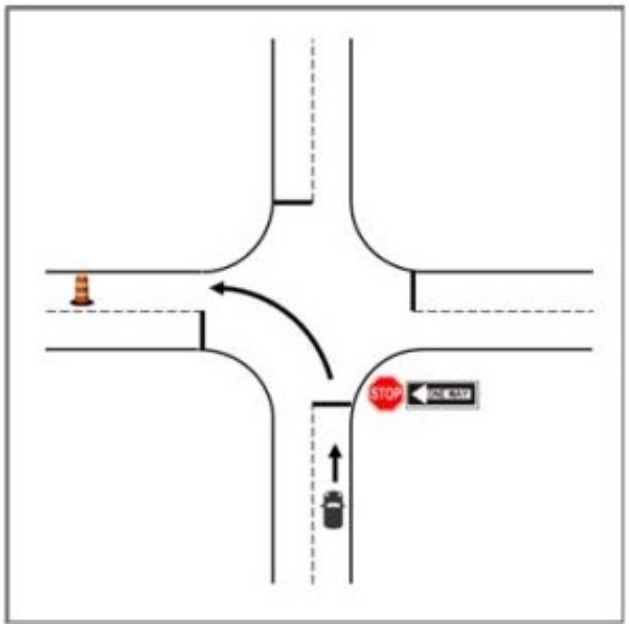


Figure 42: Intersection Tests, Left Turn

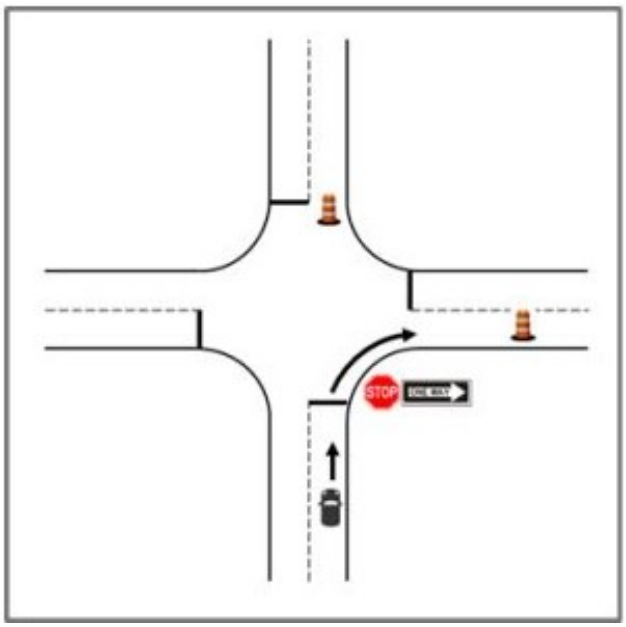


Figure 43: Intersection Tests, Right Turn

B.4 F.4: Parking

The vehicle will also need to possess the ability to park in multiple scenarios. The three that will be tested are moving out of a parking space, pulling into a parking space, and parallel parking. For the pull out test, the vehicle will need to pull out into the desired lane and stop within 3 ft of the barrel placed in that lane. For pulling in, the vehicle will turn out of its lane and navigate into a taped off box without crossing any lines. The last of the three will require the car to parallel park between two barrels. To satisfy the test requirements, the vehicle will also need to navigate into the marked out box and not hit either of the barrels.

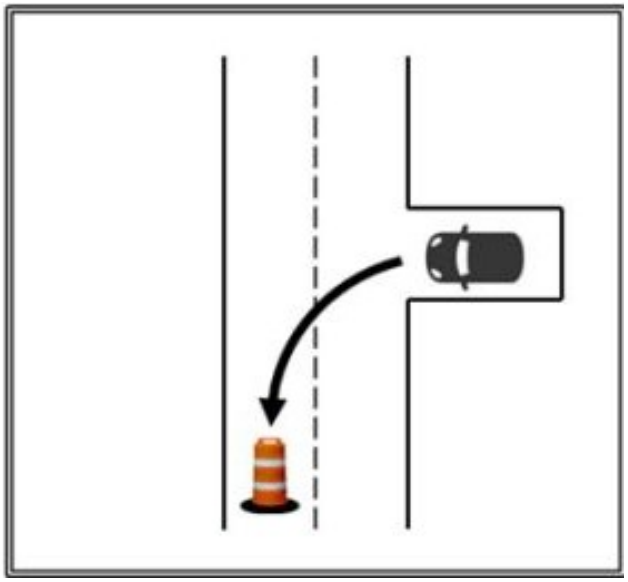


Figure 44: Parking, Pull Out

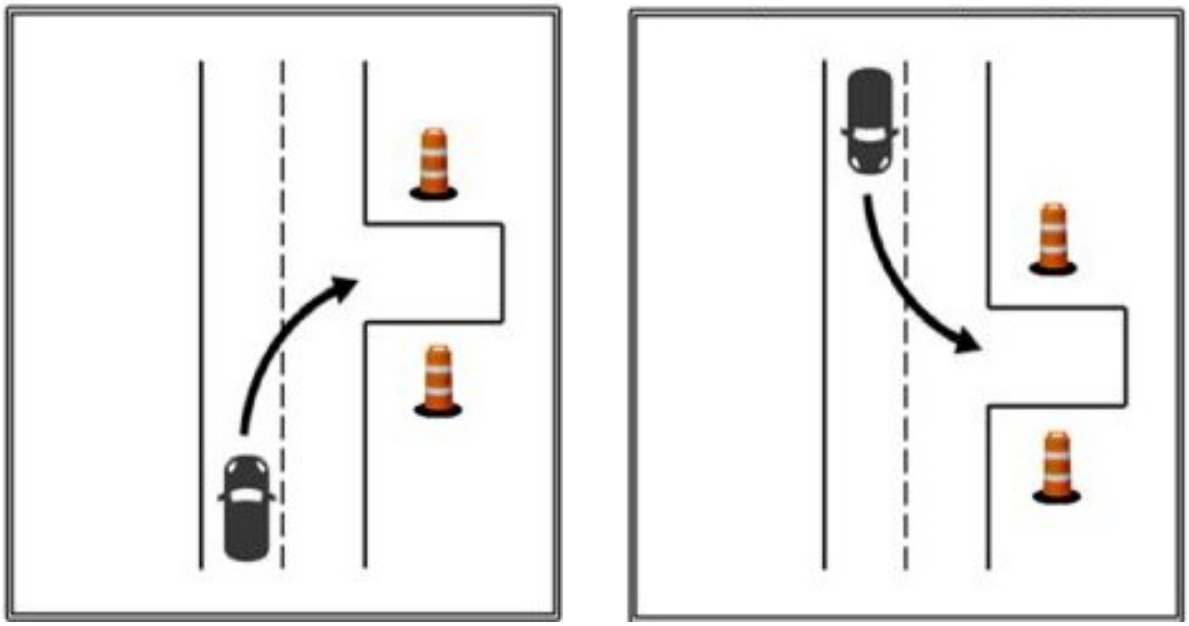


Figure 45: (A, B) Functions Testing, Parking, Pull In

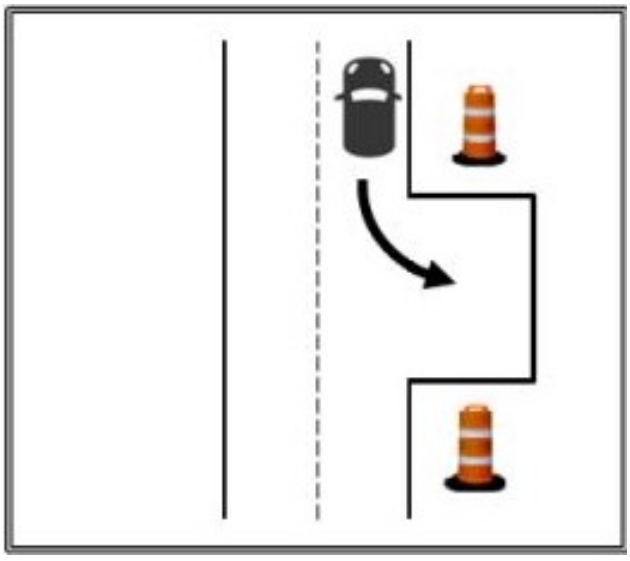


Figure 46: Parallel Parking

B.5 F.5: VRU (Vulnerable Road User)

To satisfy the pedestrian identification capabilites, four different test structures are laid out. The vehicle will need to stop in its lane when approaching a mannequin and if a mannequin is pushed out in front of it. Additionally, the car needs to have the ability to lane change upon pedestrian and barrel detection.To pass the first two tests, the vehicle simply needs to come to a complete stop within 5 ft of the pedestrian. For the second two tests, the car needs to perform a lane change 10 ft away from the object it detected.

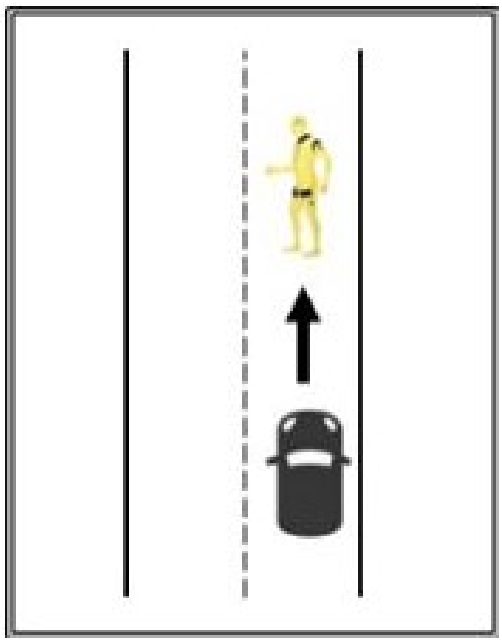


Figure 47: Functions Testing, Unobstructed Static Pedestrian Detection

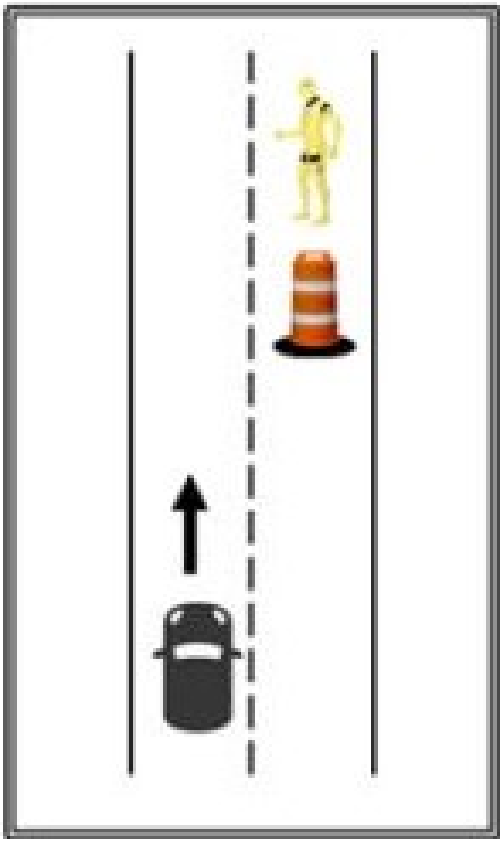


Figure 48: Functions Testing, Obstructed Dynamic Pedestrian Detection

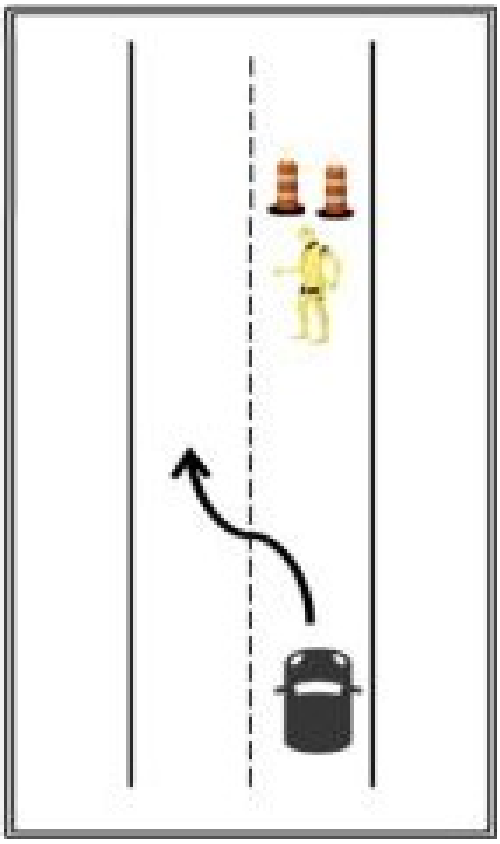


Figure 49: Functions Testing, Pedestrian Detection, Lane Change

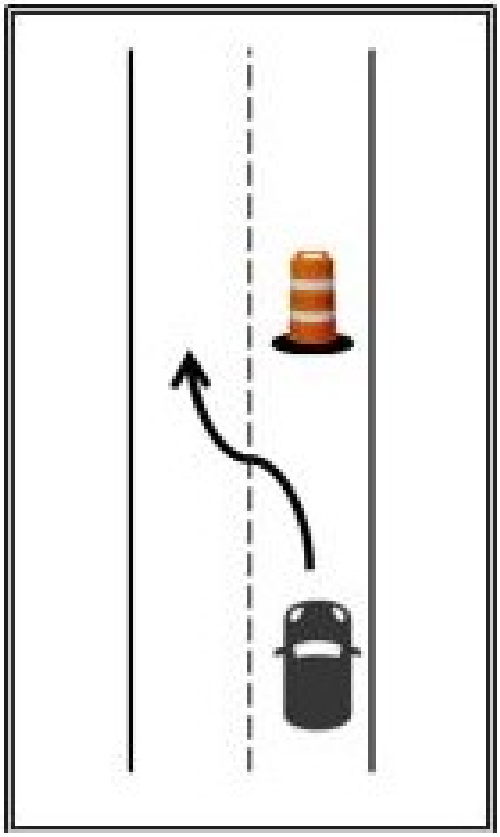


Figure 50: Functions Testing, Obstacle Detection, Lane Change

B.6 F.6: Curved Road Evaluation

Another form of the cars needed ability to navigate in turns is on curved lanes. It will be tested on its ability to accurately keep its lane in a curve and its ability to change lanes during a curve. There will be curved lanes marked out and the vehicle will need to navigate them, approach a barrel, and stop within 3ft or change lanes away from it. Successfully passing this assessment entails not crossing any boundary lines, and keeping the required distance away from the barrel.

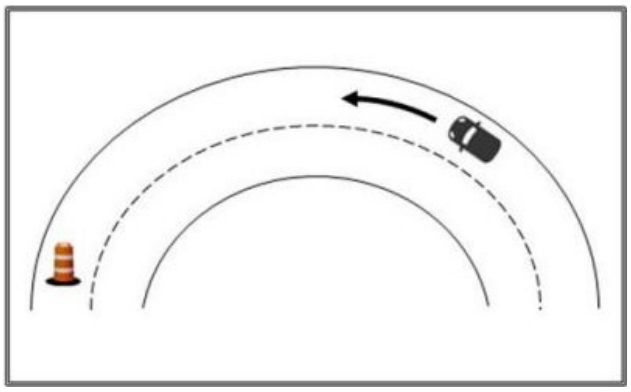


Figure 51: Functions Testing, Curved Road Evaluation, Lane Keeping

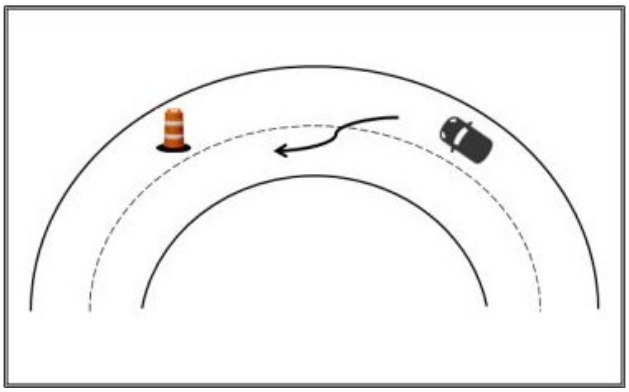


Figure 52: Functions Testing, Curved Road Evaluation, Lane Change

B.7 F.7: Other

The final two tests in the other category are for pothole detection and merging. For the pothole, a 2 ft diameter white circle will be used. The car needs to detect the pothole and change lanes to avoid it. The merging tests simulates an off-ramp onto another road.

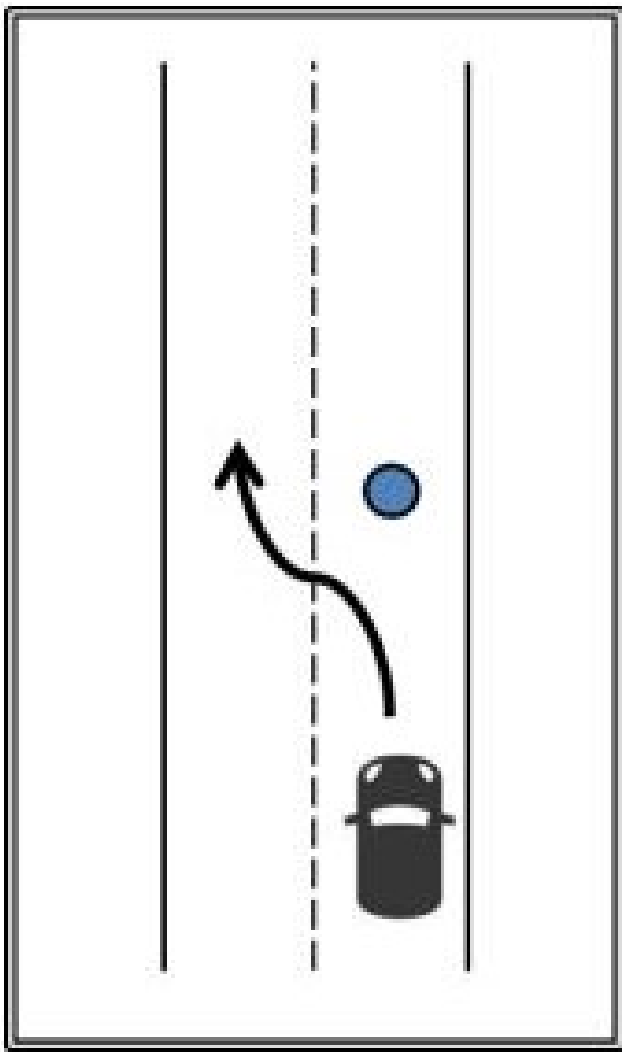


Figure 53: Functions Testing, Pothole Detection

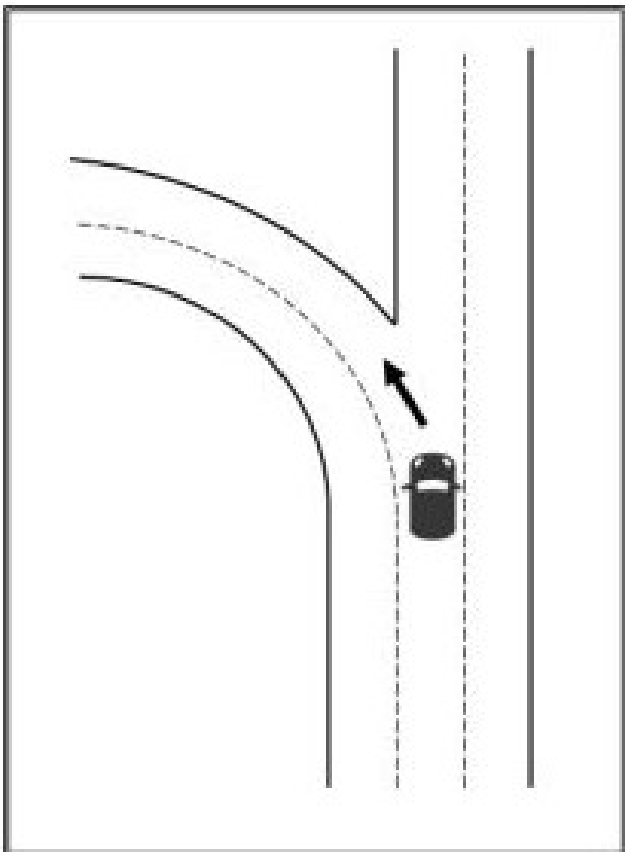


Figure 54: Functions Testing, Merging without Waypoints

All of the functional tests will be laid out as accurately as possible to the actual competition. Many of the tests start from a stationary position and end at a barrel that is used as the stop marker. For the end of the tests, the vehicle needs to stop within 3 ft of the object. Since that is consistent throughout, it will be a large focus to ensure we an easily accomplish that task.

Appendix C Main Course

This portion of the competition is to evaluate each vehicle's ability to perform in multiple categories at once. While we cannot replicate the entire course, we are confident that the qualification and functional tests will simulate every scenario that will be in the overall course. The competition course will consist of a flag to indicate the start, barrels and mannequins for obstacles, and signs for direction intent. The evaluation of the competition test is split up like the functional tests. For this reason, we will largely focus on the qualification and functional testing to validate the vehicle's ability to navigate autonomously.

As of writing (March 2023) the IGVC organizers have not provided additional information as to the main course's competition requirements. The team will assume it will follow previous years in terms of challenges presented.

Figure 55: Self-Drive Course Parking Lot 37, Oakland University