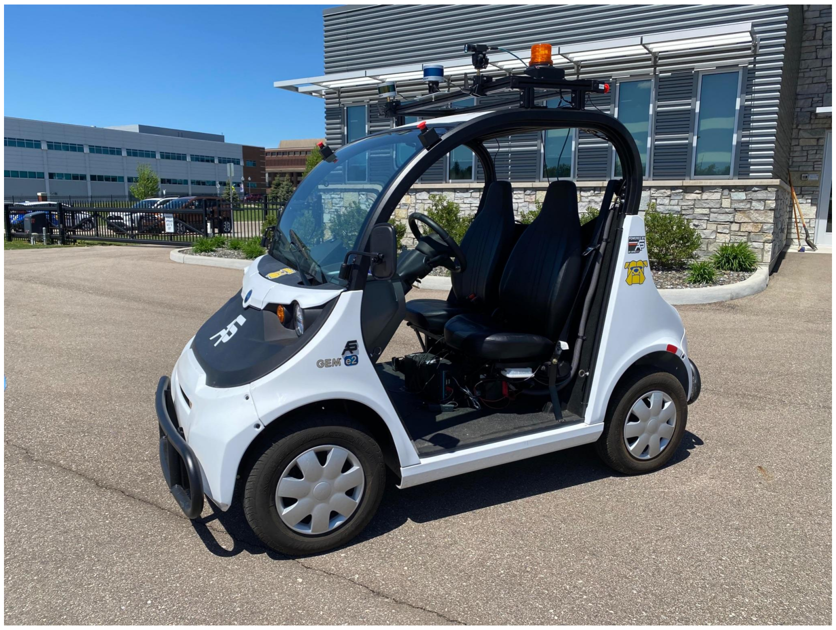
KU IGVC Bulldog 2022 Team Design Report

05/15/2022

Team Captains	Muhammad Suleman (sule4835@kettering.edu) Amanuel Weldemichael (muhaweld1172@kettering.edu) Shane Combs (comb9624@kettering.edu) Hassan Almahdi (alma9032@kettering.edu)
Faculty Advisor	Mehrdad Zadeh, Ph.D. (mzadeh@kettering.edu)



Faculty Advisor Statement of Integrity:

I, Mehrdad Zadeh, of the Department of Electrical and Computer Engineering at Kettering University, certifies that the design and development by the individuals on the design team are significant and are either for-credit or equivalent to what might be awarded credit in a senior design course at Kettering University.

Adviser Signature: Mehrdad H Zadeh	Date: 5/15/2022

TABLE OF CONTENTS

1. DESIGN PROCESS, TEAM ORGANIZATION, INNOVATIONS

1.1 Introduction

Kettering University presents the Bulldog Bolt in the 29th Annual Intelligent Ground Vehicle Competition (IGVC). The IGVC is a competition that includes teams designing and creating an autonomous ground vehicle capable of meeting a specific set of requirements and completing challenges. Kettering University has created a team, The Bulldogs, and plans on competing in IGVC in 2022. The team designed a vehicle that meets the basic autonomous structure.

1.2 Team Organization

Team members are from various disciplines at Kettering University, including computer science/engineering, electrical, and mechanical engineering students.

Name	**Major**	**Role**
Mehrdad Zadeh	Kettering University Professor	Advisor
Scott LaForest	MRC Manager	Autonomous Vehicle Technician
Shane Combs	Computer Engineering	Team Lead
David Xia	Computer Science	Decision Making & Path Planning
Roshan Cheriyan	Computer Engineering	Localization, Map Generation, & Functional Safety
Muhammad Suleman	Computer Engineering	Localization & Map Generation Captain
Amanuel Weldemichael	Computer Engineering	Object Avoidance & Path Planning Captain
Hassan Almahdi	Computer Engineering	Map Generation & Simulation Captain
Maxwell Lagassa	Computer Engineering	Map Generation & Simulation
Trishia Anne Lasaca	Computer Engineering	Map Generation & Simulation
Bravin Link	Mechanical & Computer Engineering	Map Generation & Simulation
Phillip Youn	Computer Engineering	Map Generation & Simulation
Roelle Cruz	Computer Engineering	Map Generation & Simulation
Joseph Saval	Computer Engineering	Map Generation & Simulation
Maxwell Nguyen	Computer Engineering	Functional Safety

Tyler Materna	Electrical Engineering	Functional Safety
Kevin Spike	Computer Engineering	Object Avoidance & Lane Keeping
Josiah Okoro	Computer Engineering	Object Avoidance & Lane Keeping
Erik Reich	Computer Engineering	Object Avoidance & Lane Keeping
Skylar Sabo	Computer Engineering & Computer Science	Object Avoidance & Lane Keeping
Julianna Viviano	Computer Engineering	Design Report & Presentation
Riya Mathews	Computer Engineering	Design Report & Presentation

1.3 Design Process

The details of our design is presented in the following sections. After competing in the 2021 IGVC competition, our team changed according to the feedback and mistakes learned from last year. Our team followed the IGVC rules and requirements and improved our design process. This time around, we improved the car wiring and increased the number of sensors for safety reasons. We also improved the localization and map generation process, and our focus was on getting more consistent results. A virtual reality simulation is created to complete software and hardware in the loop simulation.

1.4 Effective innovations in the design

- Designed, develop, and evaluated a virtual reality simulation to conduct hardware/software in the loop tests using a virtual simulation of the Polaris GEM Vehicle in Kettering University's Mobility Research Center (MRC)
- Designed and implemented a functional safety framework for both virtual and real testing environments based on ISO/PAS 21448 (SOTIF), ISO 26262, and HARA (Hazard Analysis Risk Assessment) to reduce safety risks and component failures
- Designed and developed a sensor fusion approach that synced the feedback of the Lidar and GNSS data together
- Designed and Implemented a strategy to compare two approaches for localization: Lidar localization and GNSS localization for a self-drive car
- Designed a ROS_based generic interface for communication between a ROS-based autonomy layer software and a new drive-by-wire system

2. ELECTRICAL/MECHANICAL AND POWER DESIGN

2.1 Overview

For the GEM car to function, there are a number of electrical components that must all communicate with each other to retrieve and pass on data or signals. These components can all be seen in figure 2.1.

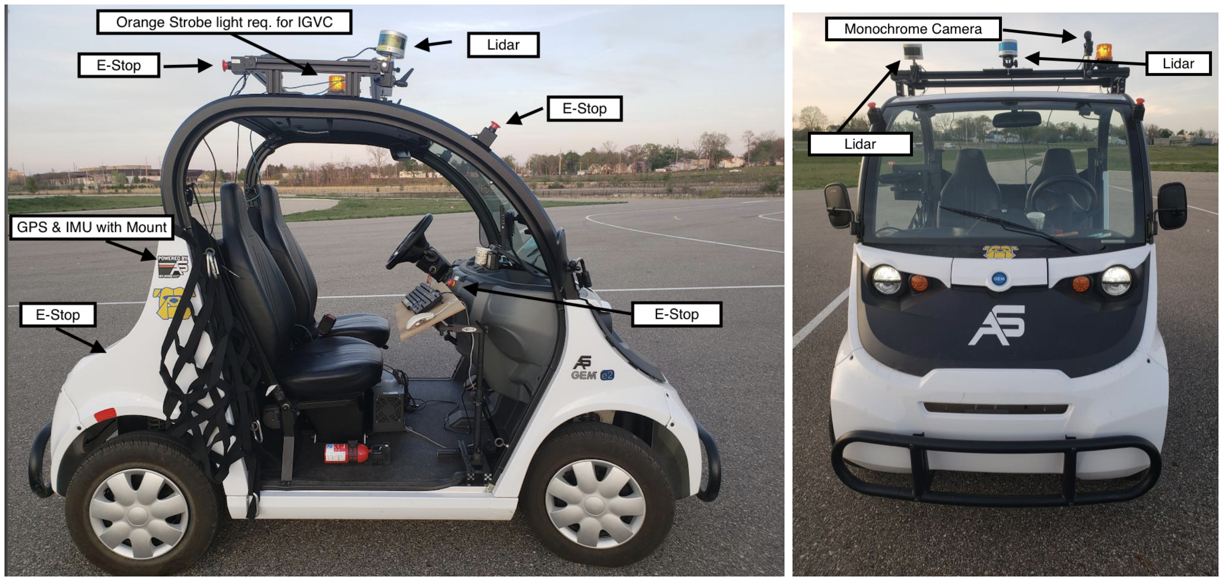


Figure 2.1: Electrical hardware design of GEM Vehicle exterior, including cameras, Lidar, E-Stops, safety light, and GPS

We use the Spectra computer that processes vast amounts of data and information very quickly. It uses an Octa-Core Intel Xeon CPU E3-1275 v5 processor, clocking in at 3.6GHz. It also has the NVIDIA GeForce RTX 2080 SUPER/PCIe/SSE2, which allows us to use GPU acceleration for tasks like Object Recognition/Tracking. We have installed 32GB of DDR4 RAM in the computer to run as many ROS nodes as necessary. The operating system that we are using is Ubuntu 64-bit version 18.04.5 LTS. This UNIX-based is very lightweight compared to Windows and has a big open source community to support its development.

2.2 Sensors

The following sensors are used in the design.

Figure 2.2: Two Camera Lenses - Edmund Optics 33300, 4mm Fixed Focal Length Lens, FOV, Lucid Triton 2.3 MP Sony IMX392 CMOS, Two Velodyne LiDAR: VLP-16, IMU: OXTS RT3000 v3

2.3 Safety Light Design

The safety light is the indicator of the vehicle's state. It is a yellow light tower. It runs off 12 volts, and it has a high output, so it is visible in the sunlight. The transition of the light's state is controlled by an Arduino, which sends a signal from the microcontroller pin to the safety light LED that is continuously lit with a steady-state DC voltage signal when the vehicle is powered or outputs it flashing during autonomous mode. The safety light is placed on the highest point of the vehicle frame to be easily visible. It also includes a plastic mount allowing us to change the location if needed.

2.4 Mechanical & Wireless E-Stop Button

There are six wired emergency stops and one wireless emergency stop (Figure 2.3). If any of these E-stop buttons are pressed, it will publish a ROS node that will set the vehicle's speed to 0 m/s. Wired E-stops are mounted on the sides of the windshield, rack of the car's roof, and on the battery cover on the rear of the car. The E-stops were put in these locations because they are easily accessible to either driver or passenger
in the vehicle. All the wiring of the e-stops is in series connected to the GEM vehicle's built-in E-stop and Arduino microcontroller board. The wires connecting the e-stops are all concealed within the rubber sealing or within the frame of the car to reduce physical mediums being exposed to potential hazards.

The vehicle safety light is required to be on whenever the vehicle is in operation, and it is required to flash when the vehicle enters autonomous mode. An Arduino Nano controls the transition of the light's state. When a signal is received that the vehicle is entering autonomous mode, the Arduino outputs a blinking electrical signal instead of a steady-state DC voltage signal. Figure 2.4 shows the design of the E-Stop.

The Wireless Kar-Tech E-Stop is implemented using an antenna mounted on the driver's side roof inside the GEM car [10]. One of the wires connecting to the antenna had to be spliced to the series connection of the wired E-Stops. The Kar-Tech E-Stop operates on 2.4 GHz and has a range of up to 100 feet.

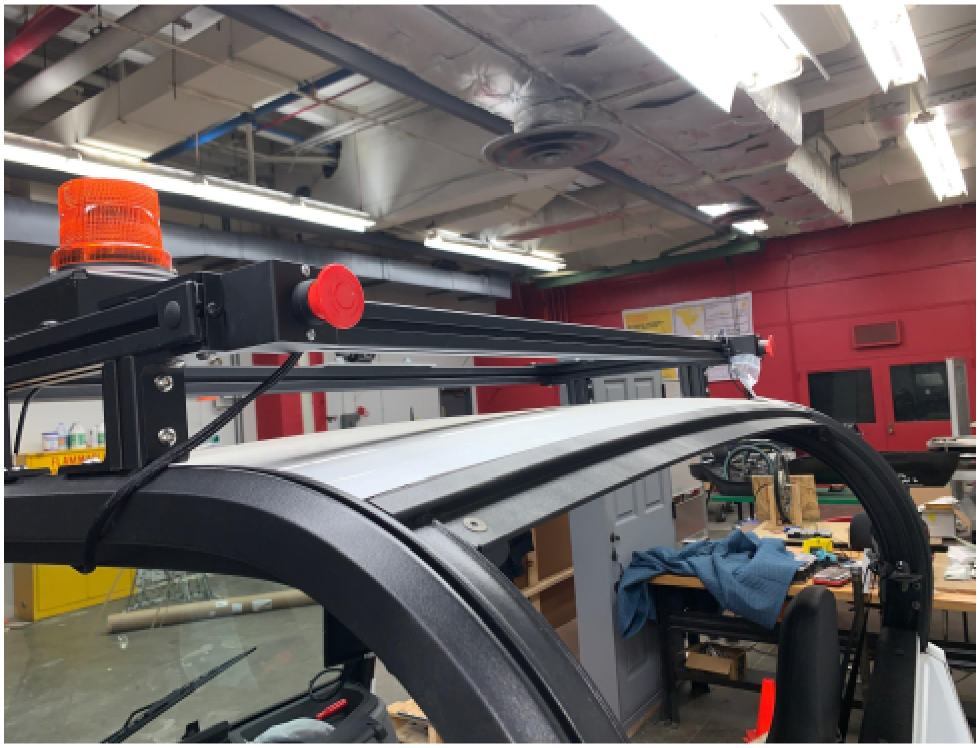


Figure 2.3: Safety light with Top Mounted Wired E-Stops, used to shut off the autonomous mode and give control to the fallback driver.

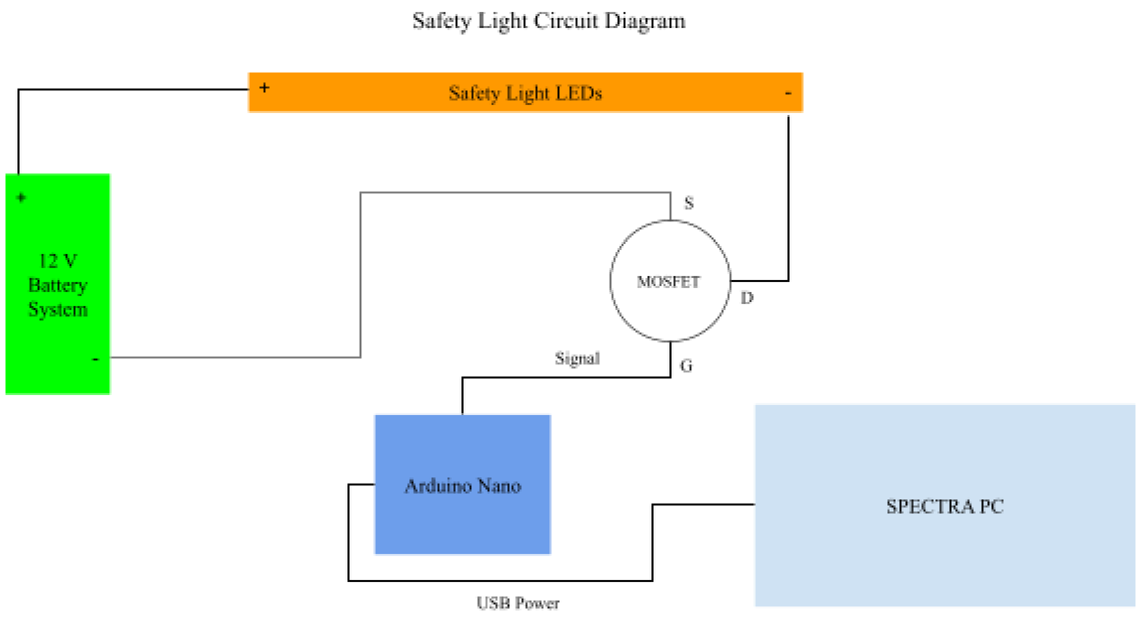


Figure 2.4: Safety Light Control Circuit and Schematic. This circuit allows the LEDs for the safety light to be in steady state when the vehicle is on, and blinking when the vehicle is in Autonomous mode, as commanded by the Arduino Nano.

3. KU IGVC BULLDOG AUTONOMY LAYER SOFTWARE DESIGN

3.1 Overview

The team implemented complex software to make the GEM Car autonomous. Software tasks include object detection, mapping, localization, navigation, path planning, and actuation, as seen in Figure 3.1.

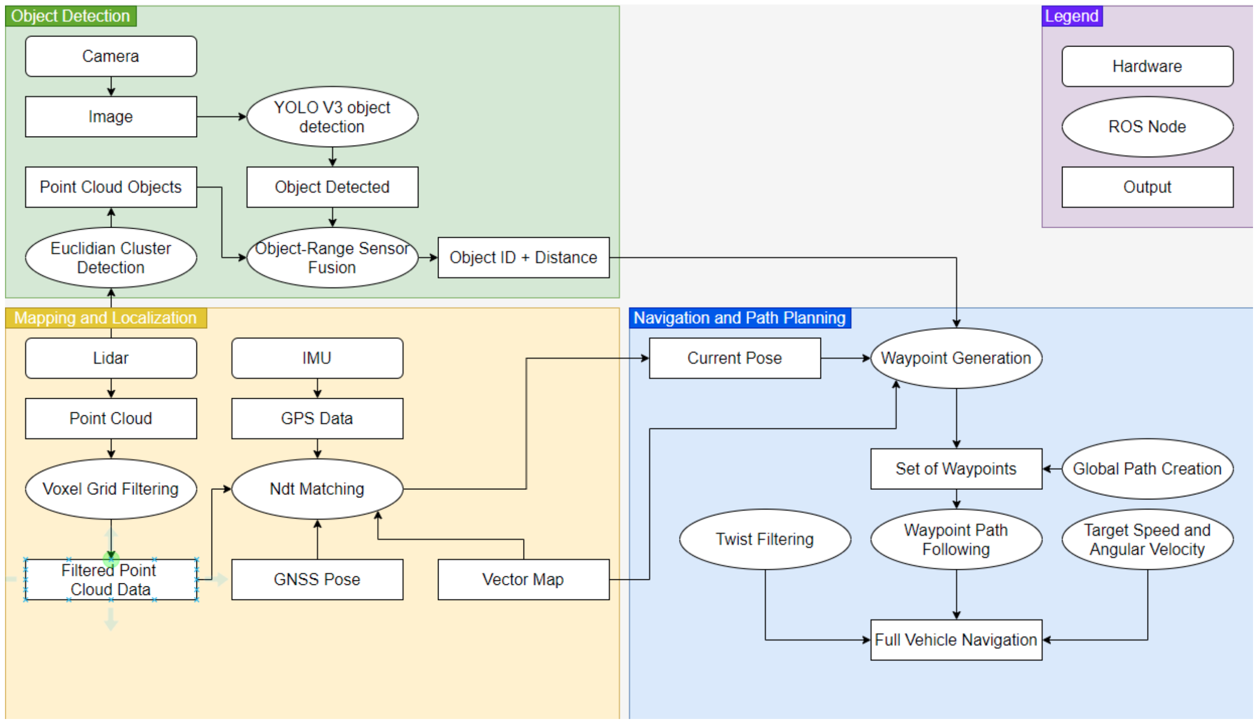


Figure 3.1: Bulldog Car Software Design. This is a high-level view of Object Detection, Mapping and Localization, Navigation and Path Planning, and their interfaces with one another.

3.2 Perception

Using the combination of lidars and cameras through sensor fusion, the GEM car can detect objects, such as street signs, pedestrians, potholes, and other possible hazards on the road, and avoid them autonomously. An electronic emergency brake stops the car if an object is detected in close proximity.

Sensor fusion is the process of fusing the camera and lidar data to detect objects. Each object detected by both is given a distance relative to the camera's location. First, the team has to calibrate the two devices by manually selecting nine points on the camera display and matching them to the corresponding points on the map generated by the lidar.

After the camera and lidar are calibrated, their output data must be in sync to detect an object and its corresponding distance from the GEM car. The ROS software architecture handles this. In ROS, the software is defined as nodes and topics. ROS nodes represent individual component modules, whereas the ROS topics look at the inputs and outputs between nodes. A publish/subscribe model is used to communicate between nodes. This model allows for better modular development. While using this model, ROS nodes communicate by passing messages under a ROS topic. Each topic has a name that describes its function. When a ROS node publishes a message to the ROS topic, another ROS node can subscribe to the topic and receive the data in the message. Messages in each topic are handled using first-in, first-out (FIFO) queues when being accessed by multiple ROS nodes at the same time [9].

The YOLO V3 ROS node [2] detects objects from the visual camera data. This node uses neural networks and a classification system to identify objects. The Lidar uses the euclidean cluster detection node [3] to detect the outlines and positions of objects. Finally, the output from these two nodes is received and fused by the object range sensor fusion node [4]. The image below shows the fused camera and lidar object detection. The team did extensive testing to verify the accuracy and consistency of our software. The team tested the sensor fusion technology at the Kettering University GM Mobility Research Center, where they set up signs for the object detection node to detect. The team precisely measured the signs from the camera at 3, 6, and 10.5 meters away. The team launched the sensor fusion node to detect and determine the distance between the camera and lidar. The distance to the stop sign from the object detection was recorded and compared to manual measurements.

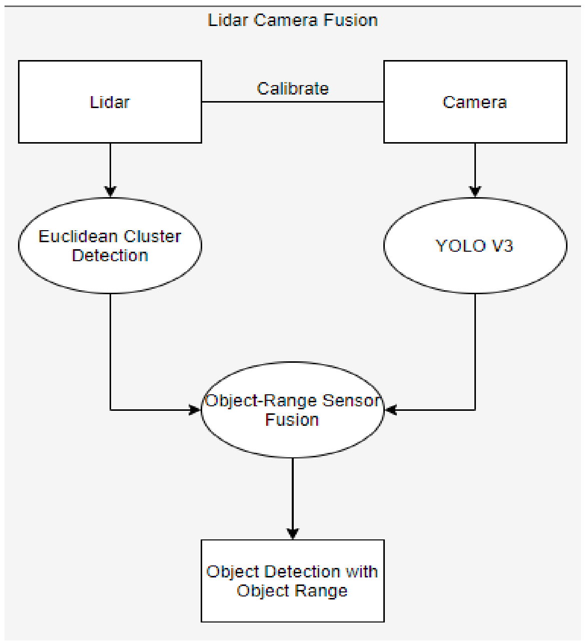

Lidar Camera Fusion
Lidar
Calibrate
Camera
Euclidean Cluster Detection
YOLO V3
Object-Range Sensor Fusion
Object Detection with Object Range

Figure 3.2: The Lidar Camera Fusion system design

3.3 Map Generation

A reference map must be generated from sensor data to localize the GEM car's position in the real world. The team drives the GEM car around different lanes or paths of the MRC track to collect lidar, IMU, and GNSS data and record the data in a rosbag file.

The lidar data is the primary source for creating the map. It is converted to a point cloud map using Simultaneous Localization and Mapping (SLAM) software. The team then draws ADAS vector or lanelet2 maps over the point cloud map using software such as MapToolbox in Unity.

However, if the point cloud data is not sufficient for making a map, the alternative method for creating the reference map is by recording GPS waypoints and converting the points into KML maps. The waypoints are initially geodetic coordinates, latitude, longitude, and altitude (measured vertically from the surface of the ellipsoid). They must be converted to geocentric coordinates (X, Y, Z) to use them in a KML map as seen in Figure 3.3.

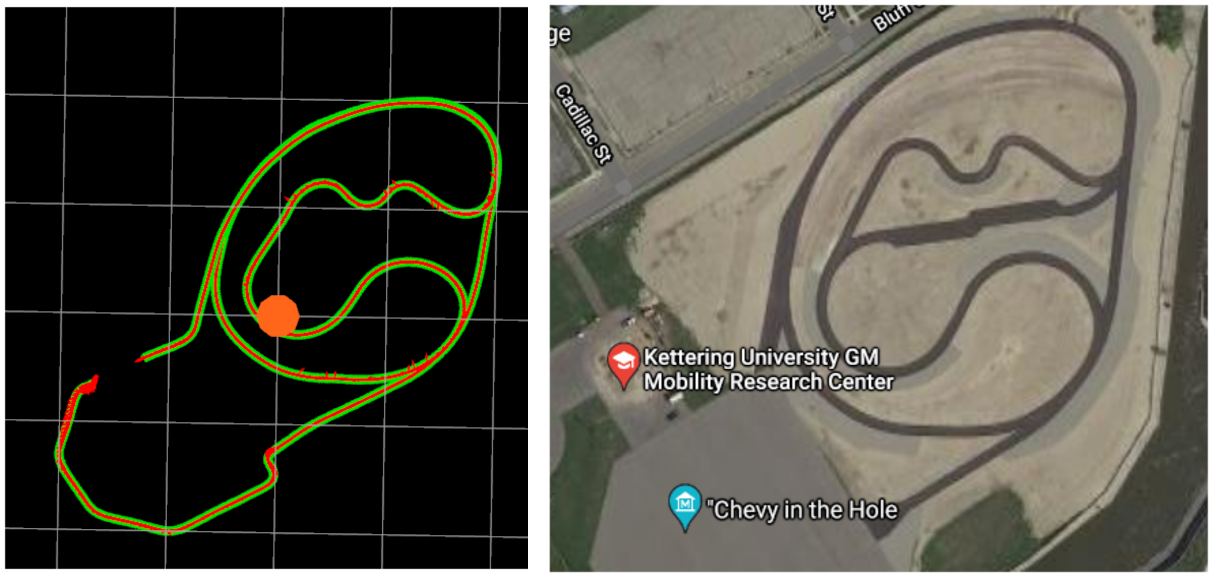


Figure 3.3: Waypoint Map from Assure (Left) Vs. Satellite Map from Google Maps (Right)

3.4 Localization

A vehicle needs to be localized on its reference map to navigate a path. There are generally two approaches for localization: Lidar localization and GNSS localization. For Lidar localization, we use Normal Distributions Transform Matching (NDT_Matching) [5]. The NDT Matching node uses the point cloud data from the map and the Lidar and outputs the current pose of the car. NDT Matching achieves this by comparing the real-time laser scan to the grid of probability functions created from the map. Before the Lidar data is passed to NDT Matching, it is downsampled using Voxel Grid Filter[6] to increase the algorithm's speed.

For GNSS localization, we use the vehicle coordinates to initialize its position. Once we drive near the global path, the path planning detects the vehicle pose and generates the most optimal trajectory.

Both the Lidar and GNSS can be used in conjunction as well. The Lidar localization is accurate. However, it can be slow when dealing with big maps with high resolution. GNSS localization is fast, but the accuracy of the localization suffers. When the methods are combined, we can achieve both efficiency and accuracy in localization.

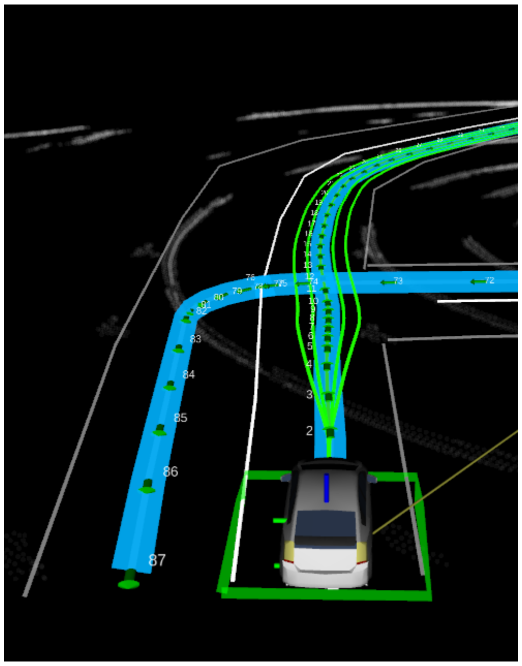


Figure 3.4: Localization Simulation shows the GEM Vehicle mapping out its surroundings utilizing GPS, Cameras, And Lidar

3.5 Navigation: Decision and Planning

Navigation involves generating a waypoint path for the autonomous vehicle to follow. Other than simply following waypoints, the GEM vehicle must also be capable of handling different maneuvers such as merging or parallel parking. Navigation and path planning includes a subset of planning algorithms to achieve mission and motion planning.

Mission planning, also called Global planning, is used to determine the global path based on the GEM car's localization and includes how to get from the starting position of the vehicle to the goal position. Motion planning, or Local planning, generates the local paths needed to achieve the mission planning goal. For instance, if an object is in the way of the global path, motion planning would direct the GEM car to take a small detour around the object before continuing down the global path.

Since the IGVC competition takes place in a parking lot, which is a fairly unstructured environment, the Bulldog team uses the hybrid-state A* algorithm for path planning [1]. The A* algorithm has three inputs: the vector map, the starting position (S_0), and the goal position(S_G). The desired output is a set of positions or waypoints (S_0, S_1, S_2, …, S_G) within a specific resolution. In our hybrid A* algorithm, a 4D search space was used (x,y,θ, r), where r represents whether the car is moving forward or reverse. This r term is used to include a penalty for switching the direction of the vehicle. The hybrid A* algorithm splits up the map into numerous cells, each with a continuous 3D state of the vehicle. For each cell, child cells are created for different drive states (left turn, right turn, straight, etc.). If these child cells have a lower cost than the original parent cell, they are updated, and the vehicle reprioritizes the path.

The op_global_planner ROS node generates the global path, and various local planning nodes generate and evaluate local paths that follow the global path. Then, the waypoint follower [7], twist filter [8], and target speed and angular velocity ROS nodes control the steering and acceleration of the GEM car to follow the waypoints.

3.6 Vehicle Mobility Control

The GEM vehicle uses drive-by-wire signals to control mobility. A drive-by-wire vehicle uses electrical signals to control the steering, throttle, and brakes. This technology is used on most new automobiles and allows for more precise control and management of the vehicle. This technology is necessary for implementing autonomy into a vehicle's mobility controls, as the ability to control steering and acceleration from a computer is vital.

When driving autonomously, the GEM vehicle uses several algorithms for controlling vehicle speed and steering input. The waypoint follower node from generates the steering angle and vehicle speed commands. The Pure Pursuit[7] algorithm calculates the speed and the steering angle based on a lookahead point. This lookahead point is located at a fixed distance from the front of the car on the selected trajectory. The Pure Pursuit node output is filtered by the Twist Filter node [8], which removes jerks from waypoint followers and ensures that the values do not exceed safety limits. These outputs from Twist Filter are taken as inputs by the Speed and Steering Control Module (SSC) *(Figure 3.2)*. The SSC translates those commands into drive-by-wire signals for the vehicle. Lastly, these commands are converted to appropriate CAN messages by Pacmod and sent to the ECU, which controls the vehicle speed and steering angle.

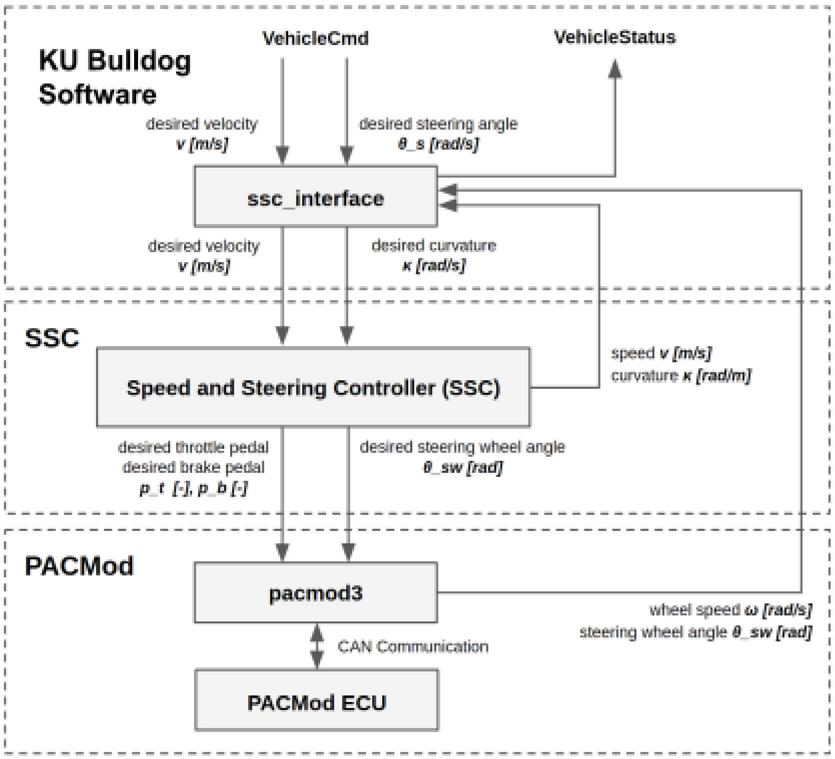


Figure 3.5: Logic Diagram of the SSC Module and how it fits into the control system of the GEM Vehicle.

4. DESCRIPTION OF FAILURE MODES, POINTS, AND RESOLUTIONS

System and software failures are necessary to account for throughout the vehicle design process. Failure modes proved easier to detect and resolve, while failure points proved to be more complex. Each point or node was evaluated based on the HARA (Hazard Analysis Risk Assessment) method. This method evaluates items through ISO 26262 and ISO 21448 (SOTIF) standards. From HARA evaluation, items are identified through their ASIL (Automotive Safety Integrity Level) measurement result, hazardous scenario and consequence, and the safety goal identification. Our vehicle safety concepts include a safety light and a mechanical and wireless e-stop button.

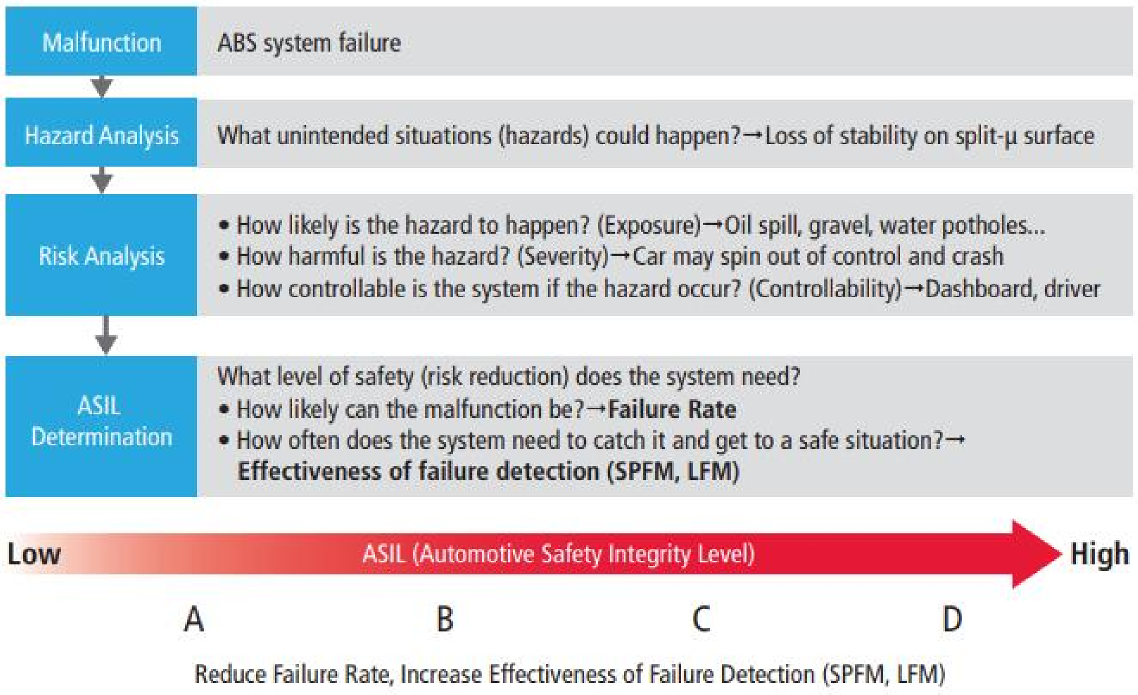

Malfunction
ABS system failure
Hazard Analysis
What unintended situations (hazards) could happen?→Loss of stability on split-μ surface
Risk Analysis
• How likely is the hazard to happen? (Exposure)→Oil spill, gravel, water potholes...
• How harmful is the hazard? (Severity)→Car may spin out of control and crash
• How controllable is the system if the hazard occur? (Controllability)→Dashboard, driver
ASIL Determination
What level of safety (risk reduction) does the system need?
• How likely can the malfunction be?→Failure Rate
• How often does the system need to catch it and get to a safe situation?→ Effectiveness of failure detection (SPFM, LFM)
Low
ASIL (Automotive Safety Integrity Level)
High
A
B
C
D
Reduce Failure Rate, Increase Effectiveness of Failure Detection (SPFM, LFM)

Figure 4.1: Automotive Safety Integrity Level ASIL Flow Chart

4.1 Vehicle failure modes and resolutions

The Arduino controls our safety features and the vehicle's speed. If an E-stop is hit, the Arduino will send the motor command to the vehicle speed to drop to zero and send another signal to the safety light to light up the LED to indicate it is in E-stop mode. It will notify the spectra autonomous path planning software to stop the task. Once the E-stop is cleared, the system will go into standby mode and wait for the operator to send a new command to initialize the autonomous task. In case of software malfunctions, the vehicle will stop immediately.

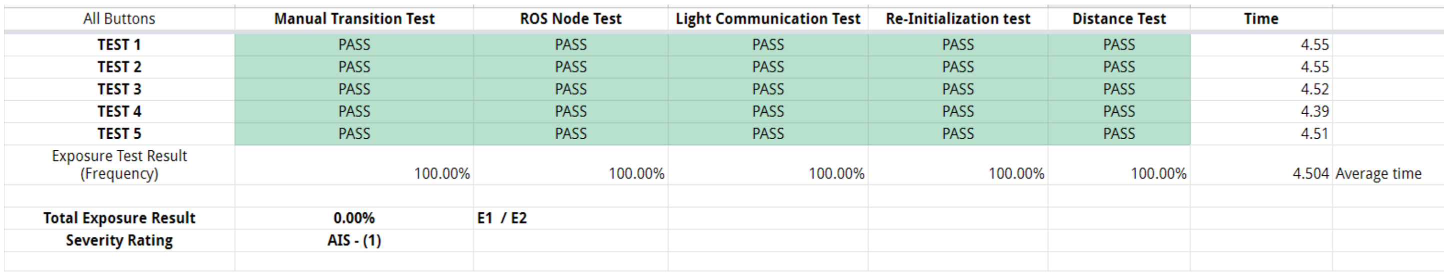
The team evaluated potential failures with the HARA evaluation method. Suppose failure points such as loss of power, switch malfunction, camera malfunction, lidar malfunction, GPS malfunction, E-stop malfunction, etc., occur. In that case, we equipped the vehicle with waterproof connectors built on the side of the spectra computer to allow for ease of plugging and unplugging of sensors and electrical lines to resolve sensor malfunctions. We also ensured all wires were secure so there won't be any issue of wires coming loose or short circuits. We have the Arduino microcontroller placed in a wooden box and covered on top with a plastic case. An extra Arduino is also available if we run into any problems with our microcontroller. If an electrical issue occurs at the competition despite our preventative measures, we will have extra wiring materials and equipment to troubleshoot and fix the problems.

4.2 Failure prevention strategy testing

Multiple trials of software testing were done in the simulation environment mentioned in the software and simulation sections. This testing was searching for repeatable software failures. All testing involving the vehicle was done at the Mobility Research Center located across the street from the university. During testing involving vehicle safety protocols were followed to protect the drivers and anyone not in the car.

To evaluate the probability of each failure, testing proceeds in a "ground-up" fashion. We start by validating the functionality of individual components (Spectra PC, LIDAR, EStop, etc.), and then move forward to incrementally more advanced tests, maintaining compliance with ISO standards. Standard test procedure:

1) Test intended scenario in simulation to verify theoretical performance
2) Validate the functionality of required components for a given test (e.g., ensure the camera is functioning correctly before testing stop sign detection)
3) Run at least five trials of the test scenario
4) Identify points of failure based on test data, evaluate using HARA



All Buttons	Manual Transition Test	ROS Node Test	Light Communication Test	Re-Initialization test	Distance Test	Time	
TEST 1	PASS	PASS	PASS	PASS	PASS	4.55	
TEST 2	PASS	PASS	PASS	PASS	PASS	4.55	
TEST 3	PASS	PASS	PASS	PASS	PASS	4.52	
TEST 4	PASS	PASS	PASS	PASS	PASS	4.39	
TEST 5	PASS	PASS	PASS	PASS	PASS	4.51	
Exposure Test Result (Frequency)	100.00%	100.00%	100.00%	100.00%	100.00%	4.504	Average time
Total Exposure Result	**0.00%**	**E1 / E2**					
Severity Rating	**AIS - (1)**						

Figure 4.2: Completed evaluation matrix for E-Stop functionality.

4.3 Functional Testing & Scenarios

Functional testing ensures the vehicle does what the team needs for the competition and goes through various scenarios. We first verify the functionality of the software and its components, including Spectra-PC/sensors, Unreal Engine virtual reality simulation, E-Stop pushbutton, camera, and lidar.

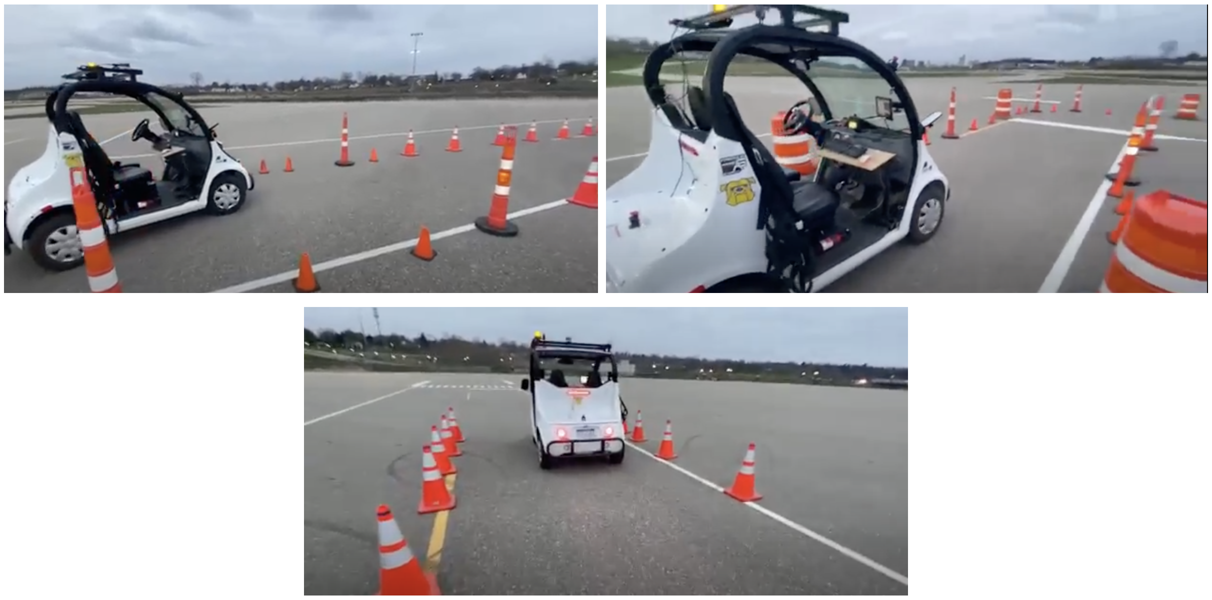


Figure 4.3: Pictures of the vehicle completing a turn-right functional navigation test.

Then, we test that the vehicle properly follows the path denoted by the software before moving on to live tests of behavior selection, such as object avoidance and stop sign behavior.

4.4 Object Avoidance Functionality

There are two ways to avoid an object depending on the user's desired behavior.

1) Stopping completely: the range for a Velodyne 16 lidar is 100 meters. The software will tell the vehicle to slow down and stop when an object is in the way. In case the software fails to act on the desired speed and given distance, the driver must brake or press the E-stop to stop the vehicle to avoid a collision.

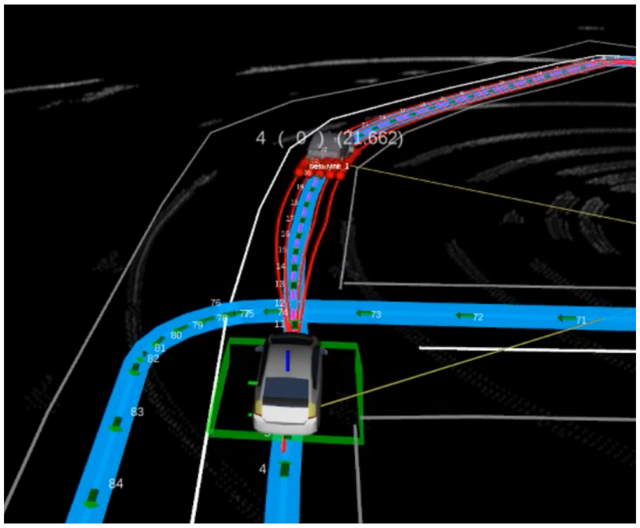


Figure 4.3: OpenPlanner simulation of vehicle stopping completely to avoid collision with a simulated object

2) Lane changing: it follows the same criteria of stopping but should instead change the line to avoid the object. Afterward, the vehicle should then return to following the planned path.

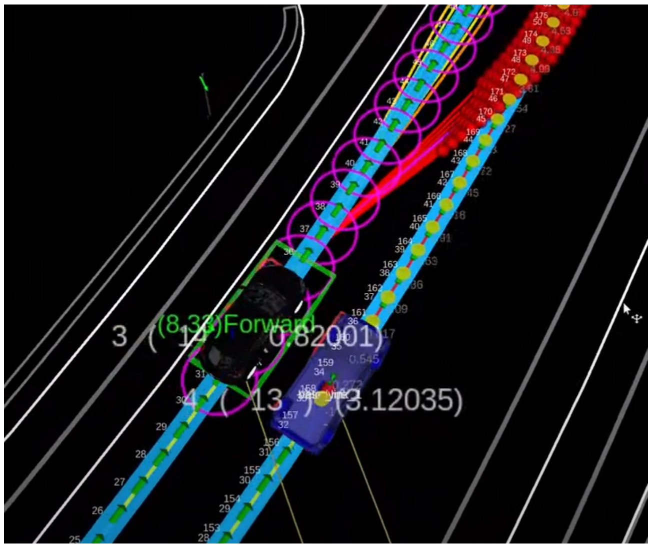


Figure 4.4: Vehicle avoiding a simulated object by doing lane changing

5. VIRTUAL REALITY SIMULATION DESIGN

5.1 Objective

The main objective is to design a virtual simulation of the Polaris GEM Electric Vehicle in Kettering University's Mobility Research Center (MRC) and examine the autonomy layer software a Software In the loop (SIL) process. Software In the loop (SIL) enables the earliest detection of system-level defects or bugs, significantly reducing the costs of later stage troubleshooting when the number and complexity of component interactions are more significant. After testing in SIL, the next step is Hardware in the Loop (HIL). Hardware in the loop (HIL) provides simulation capabilities that enable the integration of actual equipment into the simulation and allow us to test our vehicle in real life.

Various software are employed to complete the simulation, including OpenStreetMap (OSM), satellite images, Blender, RoadRunner, Unreal Engine (UE), Autoware OpenPlanner, and the CARLA simulator project. The Unreal Engine is used as a rendering engine for our simulation.

5.2 Method

We begin by modeling the vehicle and our map with Blender, a free and open-source 3D modeling software.

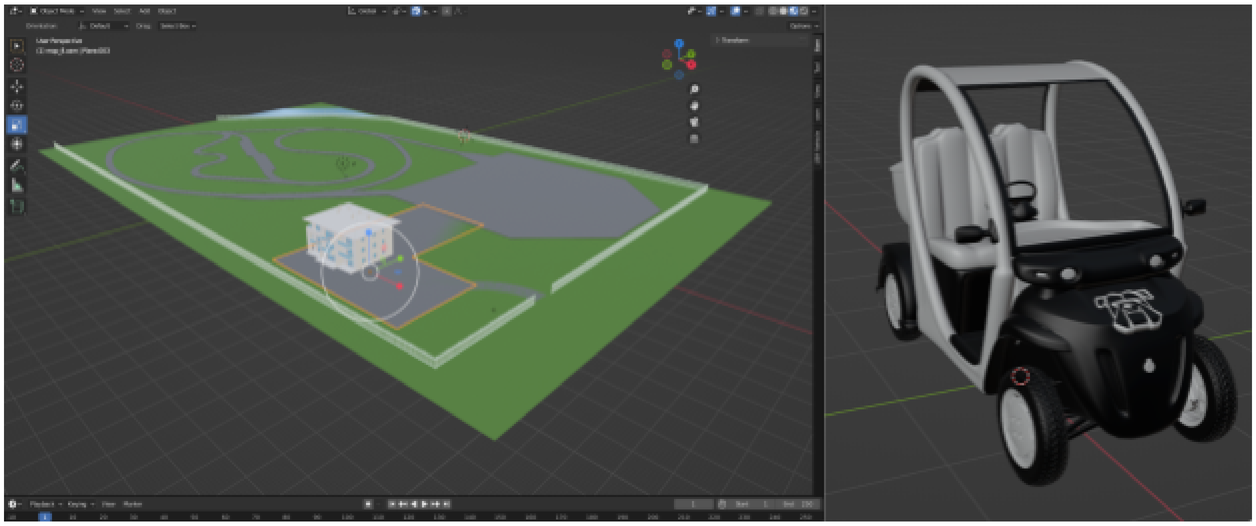


Figure 5.1 Blender model of MRC and the Polar GEM E2 side by side.

We modeled the MRC from scratch in Blender to serve as the 3D environment for our project. Measurements were obtained from satellite imagery and modified with a Geo-referenced point cloud to allow us to have accurate dimensions of the KU Mobility Research Center for our simulation. The right image details the model of our Polaris GEM Electric Vehicle.

With the map of the MRC completed, we then implemented important road features such as stop signs, pedestrian crossings, to name a few, into the map using RoadRunner. Unreal Engine was later used to apply physics to the vehicle within the simulation environment. Both models were exported in FilmBox (FBX) file format to preserve the high-definition features of the models.

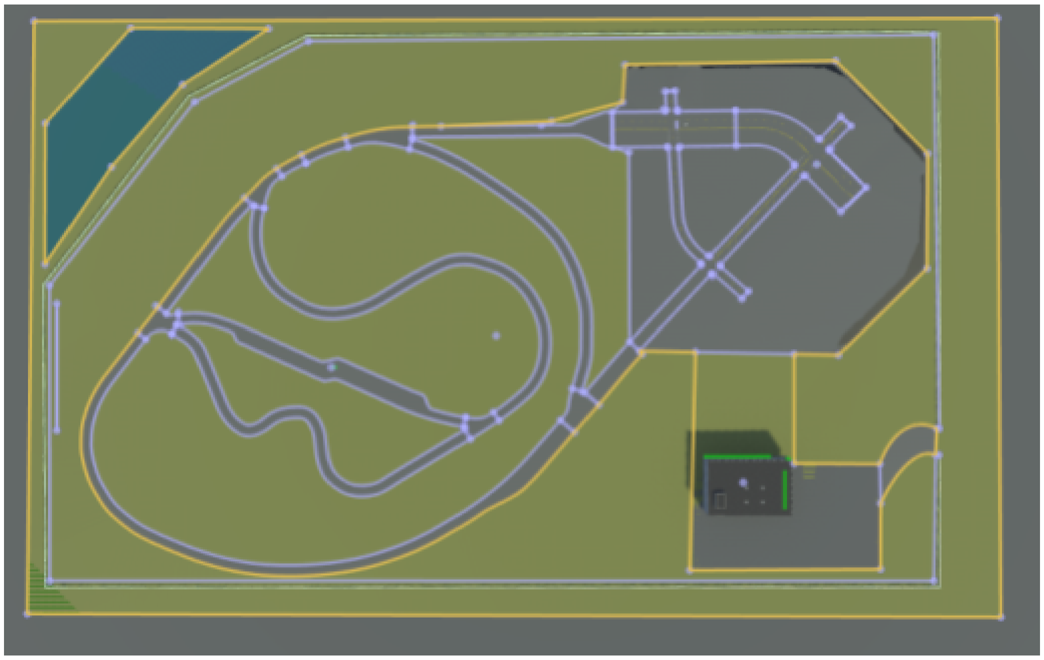


Figure 5.2: Bird's eye view of KU MRC 3D model designed using RoadRunner and Blender.

RoadRunner is an interactive editor used to design 3D scenes for simulating and testing automated driving systems. It allows us to add road signs, markings, and other road features. Using RoadRunner, we replicated the road features as seen on the MRC track. The map was exported from RoadRunner as a .fbx (Autodesk Filmbox) and .xodr files (OpenDrive) and imported into the Unreal Engine.

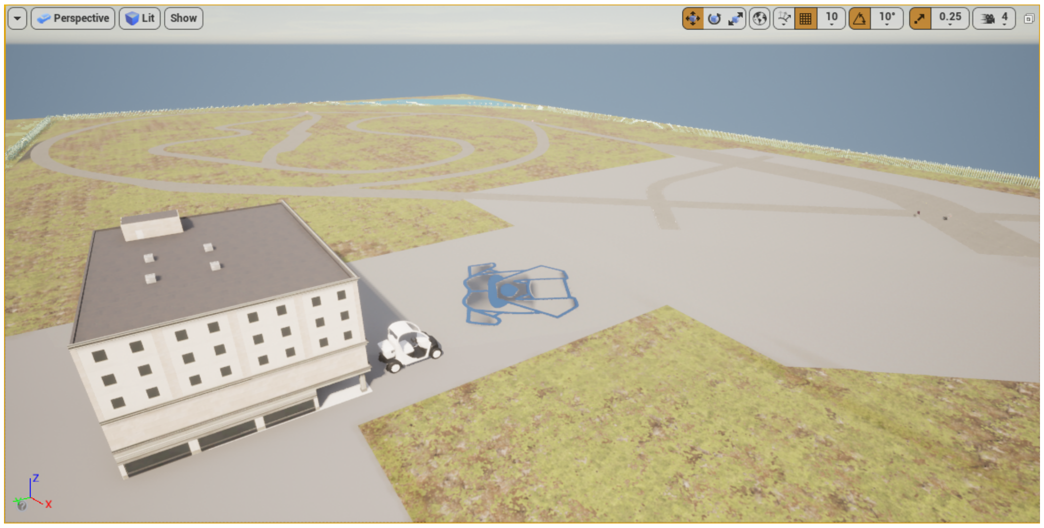


Figure 5.3 Unreal Engine View of MRC and Bulldog Bolt Car

Unreal Engine is a 3D computer graphics engine used for developing video game environments. However, recently it has become popular in the automotive industry to simulate autonomous vehicles in virtual environments. The simulated environment allows us to test autonomous vehicle software as a proof of concept before real-world testing, which is expensive and time-consuming. CARLA[11] is an open-source project built in Unreal engine for autonomous vehicle scenarios. CARLA allows us to interact with vehicle sensors and hardware using Python API from software in loop. We deployed our application in Unreal Engine to test the designed map and modified vehicles. Using Python API, we bridged our simulated environment with our autonomy layer software.

In order to visualize the OpenPlanner, we used a tool called RViz, a 3D visualization tool designed for ROS. Since the Map is not communicated through the CARLA-ROS Bridge, we have to load it separately in OpenPlanner. Using ASSURE mapping tools, a tool used for viewing, editing, and saving road network maps, we converted the .xodr file produced from RoadRunner into a .kml file (geographic modeling information in XML format). Once the .kml map was created, we imported the map to OpenPlanner. By following these steps, we ensure that the same map is loaded in Unreal Engine and OpenPlanner. From here, we created Software in the loop (SIL) and Hardware in the loop (HIL) scenarios.

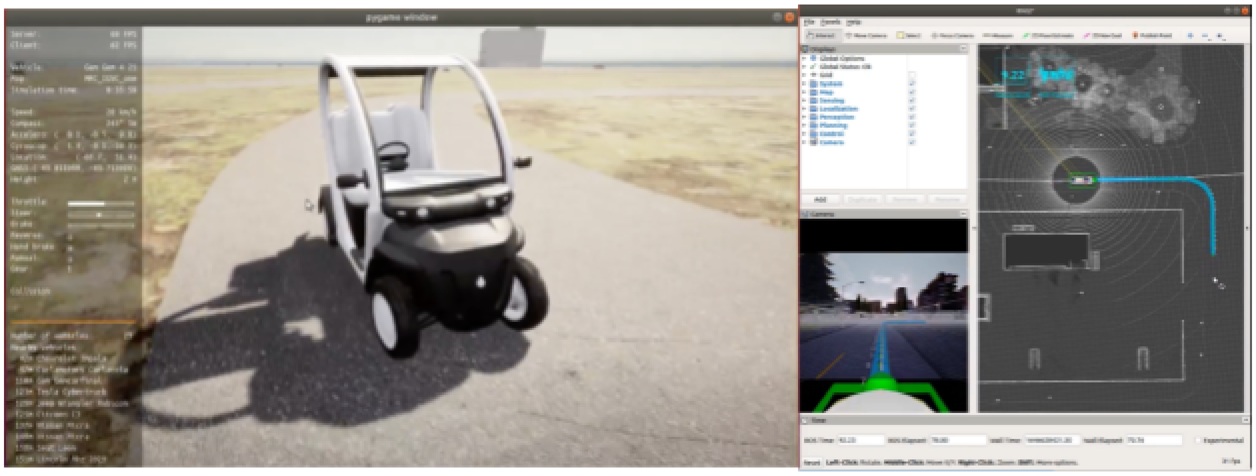


Figure 5.4 Gem Car in the model on the left. And on the right side, we can see the first-person view from the Gem car during the Software in the Loop (SIL) simulation.

5.3 Functional Safety in Simulation

To build a safety-critical system, a simulation guide from ISO/PAS 21448 (SOTIF) and ISO 26262 are followed to reduce safety risks and component failures. The SOTIF guidance is followed to have different scenarios that the SIL simulation could not identify. For example, icy roads or an object crossing the road. The V-cycle is a method recommended by ISO 26262 to use simulation in each level of the block diagram from the requirements to the implementation. By doing HIL simulation several times, the software can be tested with the chosen component, minimizing systematic risks and failures.

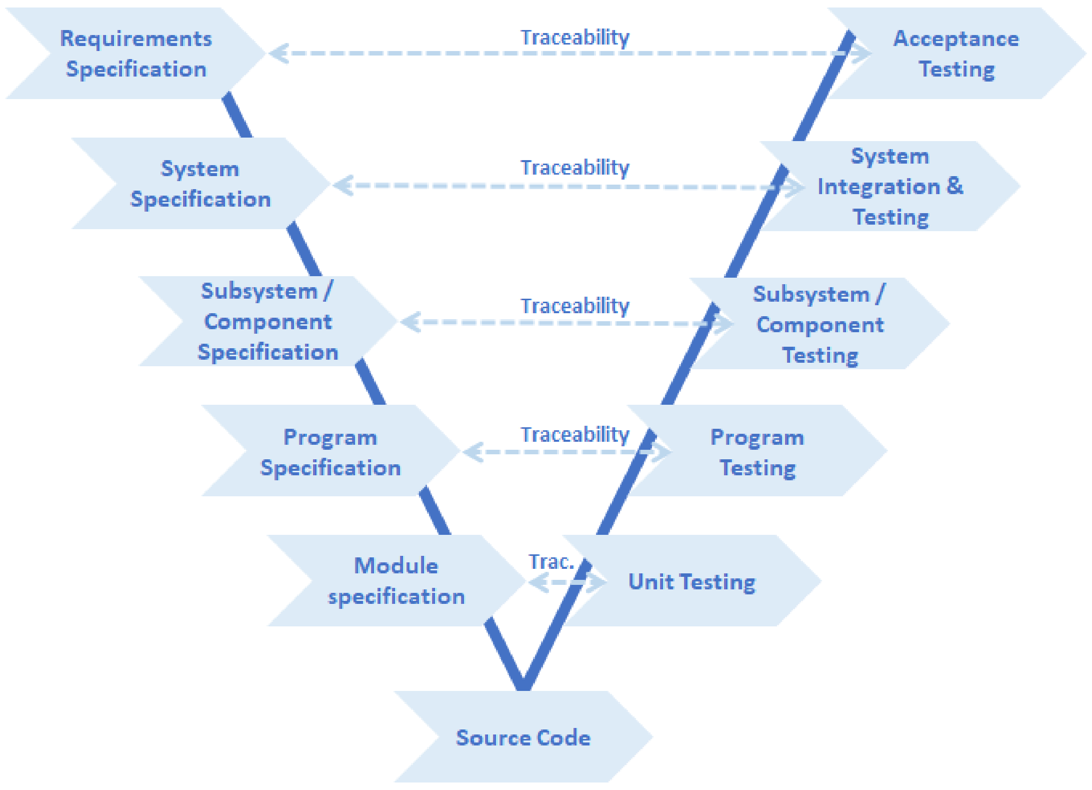


Figure 5.5: Representation of the V-Cycle for ISO 26262 Functional Safety from the requirement to the software delivery[12].

6. PERFORMANCE TESTING TO DATE AND INITIAL ASSESSMENT

- Several functional tests are completed, including stop, turns, and object avoidances. The team is in the testing process of a main course navigation, autonomously.
- Sensor fusion can be done automatically by syncing the feedback of the Lidar and GNSS data together. We are able to verify the visualization in Rviz, where the lidar and camera automatically compute the distance from the vehicle to objects in the environment.
- Collected GNSS data through our OXTS software, and we are able to generate custom data for comparison to the original map and simulate vehicle path following real-time.
- E-stops are and the safety line is engaged by toggling the buttons and when entering in autonomous mode.
- Vehicle can output two sets of velocity by following the recorded waypoint velocity or setting the velocity parameter of a ROS node to the desired speed of our own choosing
- We minimize the cost of sensor usage by equipping only one camera and lidar and GPS, and we are able to perform data collection and have the sensors engaged during autonomous simulations. We optimized the brightness of safety lights for less power consumption.

Thank you to the team for all their hard work!

7. REFERENCES

[1] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path planning for autonomous vehicles in unknown semi-structured environments,” The International Journal of Robotics Research, vol. 29, no. 5, pp. 485–501, 2010.

[2] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” CoRR, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[3] A. Ben-Hur and I. Guyon, “Detecting Stable Clusters Using Principal Component Analysis,” In: Brownstein M.J., Khodursky A.B. (eds) Functional Genomics. Methods in Molecular Biology, vol 224. Humana Press, 2003. https://doi.org/10.1385/1-59259-364-X:159

[4] D. Klimentjew, N. Hendrich and J. Zhang, "Multi sensor fusion of camera and 3D laser range finder for object recognition," 2010 IEEE Conference on Multisensor Fusion and Integration, Salt Lake City, UT, USA, 2010, pp. 236-241, doi: 10.1109/MFI.2010.5604459.

[5] P. Biber & W. Straßer, “The Normal Distributions Transform: A New Approach to Laser Scan Matching,” IEEE International Conference on Intelligent Robots and Systems. 3. 2743 - 2748 vol.3. 10.1109/IROS.2003.1249285, 2003.

[6] H. Qin, G. Guan, Y. Yu, and L. Zhong, “A VOXEL-BASED FILTERING ALGORITHM FOR MOBILE LIDAR DATA,” ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-3. 1433-1438. 10.5194/isprs-archives-XLII-3-1433-2018, 2018.

[7] R. Gutierrez, E. López-Guillén, L. Bergasa, R. Barea, O. Pérez, C. Huélamo, F. Arango, J. del Egido, and J. López, “A Waypoint Tracking Controller for Autonomous Road Vehicles Using ROS Framework,” Sensors. 20. 4062. 10.3390/s20144062, 2020.

[8] P. H. E. Becker, J. M. Arnau and A. González, "Demystifying Power and Performance Bottlenecks in Autonomous Driving Systems," IEEE International Symposium on Workload Characterization (IISWC), Beijing, China, 2020, pp. 205-215, doi: 10.1109/IISWC50251.2020.00028, 2020

[9] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, T. Azumi “Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems,” *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. doi:10.1109/iccps.2018.00035

[10] Kar-Tech Hydraulic Controls and Control Systems. [Online]. Accessed 5/07/2021. Available: https://kar-tech.com

[11] Dosovitskiy, Alexey, et al. "CARLA: An open urban driving simulator." Conference on robot learning. PMLR, 2017.

[12] Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/V-model-in-ISO-26262_fig5_347948016 [accessed 14 May, 2022]

8. ABBREVIATIONS

Table 2: Abbreviations of all the Acronyms used in the paper

Abbreviation	**Explanation**
KU	Kettering University
MRC	Mobility Research Center, an autonomous vehicle testing track located on KU campus
SIL	Software in the Loop
HIL	Hardware in the Loop
CARLA	Car Learning to Act, an open simulator for urban driving.
IGVC	Intelligent Ground Vehicle Competition
ROS	Robotic Operating System
ASIL	Automotive Safety Integrity Level, a risk classification scheme defined by the ISO 26262
GNSS	Global navigation satellite system (GNSS)
LIDAR	Light Detection and Ranging