UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DESIGN AND RAPID PROTOTYPING FOR COST-EFFECTIVE SLAM-BASED SEMI-AUTONOMOUS GROUND VEHICLES

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By EVAN JACKSON

Norman, Oklahoma

2026

DESIGN AND RAPID PROTOTYPING FOR COST-EFFECTIVE SLAM-BASED SEMI-AUTONOMOUS GROUND VEHICLES

A THESIS APPROVED FOR THE

SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Yingtao Liu, Chair

Dr. Wei Sun, Co-Chair

Dr. Bin Xu

Acknowledgements

Firstly, I would like to give thanks to Dr. Yingtao Liu. His continued support of an untraditional thesis project has allowed me to go further and achieve more milestones with the SAGV than I saw plausible. Many opportunities that have come along with this research have been so due to his willingness to support a project outside his own lab.

Secondly, I would like to acknowledge the iHub Fabrication Lab and the University of Oklahoma Softball coaching staff for their continued support, feedback, and resources in both V1 and V2 of the SAGV.

Lastly, I would like to thank my spouse, Karly. She has been a pillar throughout the ups and downs of higher education, and I would not be able to produce such work if she had not been pushing me to achieve my best every day.

Table of Contents

Table of Figures

Table of Tables

Abstract

This thesis explores the integration of additive manufacturing and LiDAR-based navigation to manufacture a semi-autonomous ground vehicle (SAGV) capable of operating in diverse environments. The research scope aims at scalable and cost-effective manufacturing of a SAGV product. By utilizing additive manufacturing, specifically Fused Deposition Modeling (FDM), critical structural components such as mounts, chassis modifications, and protective casings were rapidly prototyped. Thermoplastic polyurethanes were evaluated for their high-impact durability and stress-fatigue performance, ensuring the vehicle's reliability during testing and use in high impact scenarios.

LiDAR was integrated into a compact ground vehicle platform. Through ROS (Robot Operating System), LiDAR data was processed in real time for obstacle detection, path planning, and Simultaneous Localization and Mapping (SLAM). The autonomy stack incorporated algorithms such as A* and Dijkstra for pathfinding, in addition to real-time collision-avoidance strategies. Extensive testing involved both simulation-based trials, allowing for rapid iteration on software parameters, and physical experiments that validated hardware resilience and system responsiveness.

Testing showed that additive manufacturing reduced design lead time and supported rapid redesign of structural parts, while the LiDAR stack provided repeatable obstacle detection and map generation in GPS-denied trials. The thesis identifies possible applications in logistics, sports training, and reconnaissance, and discusses how the platform could be adapted to each use case. Future work will include exploring additional sensors, advanced SLAM capabilities, and 3D point cloud generation.

Chapter 1: Introduction

The idea for the autonomous ground vehicle (AGV) originated during my work at the iHub Fabrication Lab. OU Softball Coach JT Gasso approached the lab about using an RC car to run the bases during practice so scout-team players would not continue to get injured. The first version delivered was a basic RC car equipped with a pool-noodle mount and a 360-degree bumper for vehicle and athlete protection. The pool noodle served as the tag target, while the bumper protected both the car and the players.

Since then, the project has developed into a startup effort and a leading FabLab project. I have continued working alongside OU softball as a pilot partner in developing new technology for the AGV. The long-term goal is to integrate LiDAR and autonomy systems for navigation.

The current state of the car is summarized below:

- Modified chassis for bumper mounting and upgraded electronic speed controller
- Acceleration, deceleration, and speed parameters to simulate specific players' baserunning performance

Figure 1-1: Completed SAGV Build

Figure 1-2: Assembled Version 1 "Base Runner"

This RC car is a HOSIM 1/10-scale off-roader. It measures 22 x 12 in., while the full bumper system increases the overall footprint to 31 x 18 in. The complete vehicle weighs 12.5 lb.

1.1 Review of Additive Manufacturing

Additive Manufacturing (AM), often referred to as 3D printing, has rapidly expanded across multiple industries, including Aerospace, Automotive, and Biomedical. While Fused Deposition Modeling (FDM) is the most common form of AM for hobbyists and low-cost prototyping, more advanced techniques have found roles in manufacturing environments [1]. AM's main advantages lie in rapid prototyping, creating specialized low-production parts, and producing shapes too complex for traditional methods. The cost of equipment varies widely: FDM printers are typically inexpensive, while more sophisticated processes such as

Powder Bed Fusion (PBF) remain costly and are often confined to commercial or research settings. In 2015, the American Society for Testing and Materials (ASTM) classified AM into seven process categories: binder jetting, directed energy deposition, material extrusion, material jetting, powder bed fusion, sheet lamination, and vat photopolymerization. These processes differ in terms of materials, operating principles, and final part characteristics. For instance, material extrusion (like FDM) generally uses thermoplastics, whereas PBF can accommodate a range of metals [1].

Despite these differences, AM processes follow a similar workflow. First, a 3D model of the desired part is created using computer-aided design (CAD) tools. This model is then converted into an STL file, which captures the surfaces and geometry. Next, slicing software divides the 3D model into discrete layers, determining how each layer will be deposited. This is also when supports, infill patterns, or other structural features are added. Once the printer is prepared with the required material and checked for proper operation, printing begins. The machine deposits material layer by layer until a complete 3D object is formed. The printed component is then removed, which can be simple or hazardous depending on the technology and materials involved. Post-processing steps may include cleaning, support removal, heat treatment, machining, or the addition of inserts. Only after these procedures is the final part considered ready for use [2].

While the term “3D printing” suggests an entirely automated process, high-end equipment often demands precise operating conditions, regular calibration, and ongoing supervision. AM also intersects with material science, since each printer and material combination can

introduce unique variables. This variability sometimes causes hesitancy in adopting AM for mission-critical applications, leading to rigorous testing protocols and repeatable validation efforts by industries and regulatory bodies. For example, technical societies and governing agencies rely on proven engineering practices to build confidence in these newer processes. There is additional complexity in how materials are handled before printing; many require special storage conditions or have limited shelf lives, which affects both cost and feasibility [3].

Additive manufacturing is increasingly used for prototyping and low-volume specialized parts. FDM remains common because of its low cost, while PBF is more often used when higher material performance is required. Regardless of the chosen method, all AM processes rely on similar steps—CAD design, slicing, printing, and post-processing, but machine upkeep, material selection, and process validation remain critical to ensuring reliable, high-quality parts [1][2][3].

1.1.1 Fused Deposition Modeling

Fused Deposition Modeling (FDM), sometimes referred to as Fused Filament Fabrication (FFF), is an additive manufacturing process that fabricates three-dimensional objects by depositing molten thermoplastic material in successive layers. FDM distinguishes itself from other 3D-printing processes through its relatively straightforward operation, cost-effectiveness, and suitability for a wide range of prototyping and functional applications. Given the available resources and the cost-effectiveness of the process, FDM was selected instead of more advanced methods such as SLS or DMLS [4].

FDM is predicated on the controlled extrusion of a thermoplastic filament through a heated nozzle. The material is incrementally deposited along a predefined toolpath, and each newly deposited layer fuses with the previous one as it cools. In contrast to techniques such as Stereolithography (SLA) or Selective Laser Sintering (SLS), which rely on photopolymerization or powder bed fusion, FDM's defining characteristic is its direct extrusion of filament without requiring a liquid resin or powder-based feedstock [6].

FDM commonly employs engineering-grade thermoplastics, including [5]:

- Polylactic Acid (PLA): Biodegradable, low warp, suited for beginners.
- Acrylonitrile Butadiene Styrene (ABS): Stronger and more temperature-resistant but prone to warping.
- Polyethylene Terephthalate Glycol (PETG): Combines strength, temperature stability, and limited warping.
- Nylon (Polyamide) and Polycarbonate (PC): Noted for high tensile strength and durability.

The primary hardware components of FDM systems include [5]:

- Extruder and Nozzle: Responsible for heating and depositing molten filaments; nozzle diameters vary to control extrusion width.
- Heated Build Platform (Bed): Maintains uniform temperature, reducing warping and promoting layer adhesion.
- Motion Control System: Typically driven by stepper motors that move the nozzle and/or build platforms along the X, Y, and Z axes.

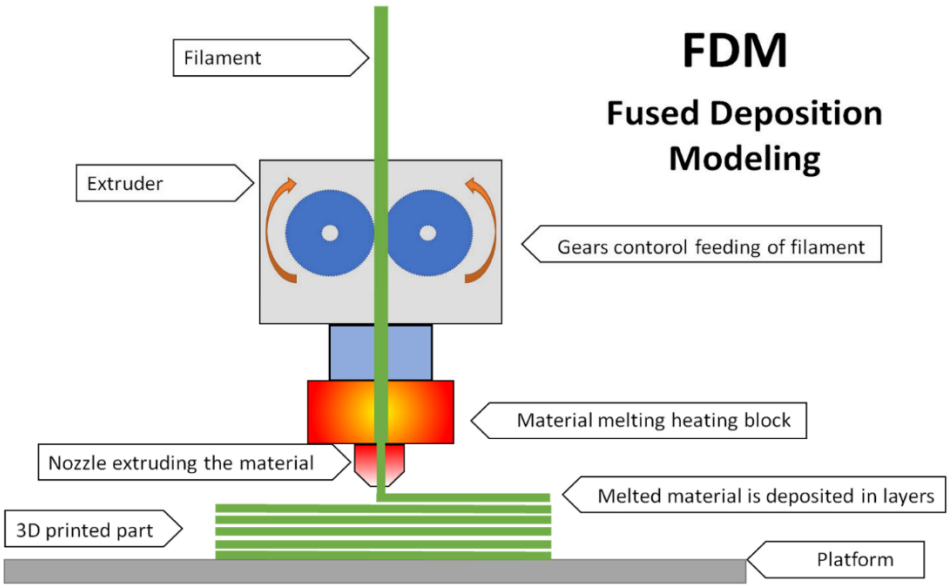


Figure 1.1.1-1: FDM Graphic [4]

The FDM workflow begins by slicing a three-dimensional digital model into discrete layers, a process governed by slicing software that also adjusts parameters such as layer height, print speed, infill density, and shell thickness. The machine then melts the filament at a temperature specific to the material's properties, extruding it through the nozzle. As each layer solidifies, the platform (or extruder, depending on the printer build) moves to accommodate additional layers, forming the object incrementally. [4][6]

Print settings significantly influence part quality, including the following:

Layer Height: Thinner layers enhance surface finish but increase build time.

Print Speed: Faster speeds shorten print time but may sacrifice dimensional accuracy.

Infill Density and Pattern: Higher infill yields greater mechanical strength but elevates material use and print duration. [4][6]

Advantages

- Ease of Use: FDM machines are relatively user-friendly and widely available.

- Cost-Effectiveness: Filament materials and equipment are less expensive than those used in many other 3D printing processes.
- Material Availability: A broad range of filaments with diverse mechanical and thermal properties can be employed. [1][4][6]

Disadvantages

- Surface Finish: Visible layer lines and surface roughness often necessitate post-processing. This may be crucial in other research projects but will not be significant for my AGV research.
- Mechanical Properties: Parts can exhibit weaker interlayer bonding, which impacts structural integrity.
- Warping and Dimensional Accuracy: Without optimal temperature control or bed adhesion, parts may warp or shrink during cooling. [1][4][6]

FDM's accessibility has led to its widespread adoption across multiple industries. In product design, it supports rapid prototyping of consumer goods, while in the automotive and aerospace sectors it enables the fabrication of custom jigs, fixtures, and low-volume end-use parts. FDM will be used in this research because it is accessible and supports rapid design iteration for both mechanical testing and AGV chassis changes [6].

1.2 Review of Autonomous Systems

Autonomous systems are being adopted in multiple industries because they can automate sensing, planning, and control tasks. The emergence of sophisticated control algorithms, advanced sensor technologies, and artificial intelligence (AI) has enabled the development of

automated mobile robots, ranging from aerial drones to terrestrial ground vehicles, that can perform complex tasks with minimal human intervention. This section reviews core concepts in autonomous systems, common applications of mobile robots, and the sensing, perception, planning, and control functions most relevant to autonomous ground vehicles [7].

Autonomous systems now play a major role in many engineering applications. These systems encompass technologies that perform tasks and make decisions with limited human input. Advances in machine learning (ML), control theory, and sensor fusion have expanded the capabilities of mobile robots, especially autonomous ground vehicles. These vehicles use GPS, LiDAR, radar, and computer vision to perceive their surroundings and navigate accordingly [7]. This review focuses on the design and operation of AGVs, emphasizing the components and challenges most relevant to this thesis.

Key Components of Autonomy

- Sensing and Perception: Autonomous systems rely on sensors (e.g., cameras, LiDAR, ultrasonic, inertial measurement units) to gather data about their environment. This raw sensor data is processed using signal processing and computer vision techniques to extract information, such as object detection, localization, and environmental mapping [8] [9].
- Decision-Making and Planning: Once the system perceives the environment, it uses decision-making algorithms (e.g., machine learning classifiers, Markov Decision Processes, or rule-based systems) to plan subsequent actions. Path planning

algorithms, such as Rapidly exploring Random Trees (RRT) or A*, can be used to find optimal routes [8][9]

- Control and Execution: After planning, the system needs to execute the chosen actions. In robotics, controllers (e.g., proportional-integral-derivative (PID) controllers or Model Predictive Control) ensure that actuators (motors, servos, wheels, arms) follow the planned trajectory accurately while adapting to changing conditions or terrain. [9]
- Communication and Coordination: Many modern autonomous systems operate in networked environments, exchanging data with other machines or a central control system. This can involve advanced communication protocols or cloud-based processing. This project, however, avoids reliance on device-to-device communication [9].

Levels of Autonomy

The levels of autonomy (adapted from the Society of Automotive Engineers (SAE) levels for vehicles) can be applied broadly to robotics concepts and projects [10]:

- Level 0 (No Autonomy): All tasks require human operation.
- Level 1 (Assisted): Certain functions (like speed or steering) are automated under specific conditions, but the human operator remains in primary control.
- Level 2 (Partial Autonomy): The system can handle more functions simultaneously (e.g., steering and acceleration), but the human driver must monitor and intervene.
- Level 3 (Conditional Autonomy): Under specific conditions, the system can make decisions and control the vehicle, with the human ready to intervene.

- Level 4 (High Autonomy): The system can operate in self-driving mode with no human intervention in certain environments or geofenced areas.
- Level 5 (Full Autonomy): The system can handle any environment or condition without any human intervention.

Intended Industrial and Commercial Applications

- Athletics: SAGVs can take the place of a scout player or moving target, reducing unnecessary exertion during training and lowering the risk of injury.
- Military and Defense: Unmanned ground vehicles (UGVs) carry out surveillance, explosive ordnance disposal, and reconnaissance missions without risking human life. [12]

Technical Foundations [10]

Automated mobile robots, including ground vehicles, draw heavily on the concepts of simultaneous localization and mapping (SLAM), robust path planning, and sensor fusion. Ground vehicles often employ:

- LiDAR and Radar: For obstacle detection and distance measurement.
- Computer Vision: For object recognition, lane detection, and navigational aids.
- GPS/INS: For coarse global localization (though can be supplemented or replaced in indoor or GPS-denied environments by vision-based localization techniques).
- Onboard Microcontrollers or High-Level Processors: For real-time planning and decision-making.

1.2.1 LiDAR Navigation

LiDAR (Light Detection and Ranging) is a remote-sensing method that measures distance using laser returns, allowing a ground vehicle to estimate surrounding geometry with high spatial resolution. The technology works by emitting laser pulses that bounce off surrounding objects and return to the sensor. By measuring the time each pulse takes to return, the system calculates the distance to objects and creates a detailed spatial map. LiDAR is widely used in autonomous ground vehicles because it provides real-time, high-resolution depth perception that is well suited for navigation, obstacle avoidance, and mapping [3].

LiDAR navigation operates through a series of key processes. First, the system emits thousands of laser pulses per second, which interact with objects and reflect to the sensor. The time each pulse takes to return is used to determine the distance to surrounding objects. This data is then compiled into a point cloud, a three-dimensional representation of the environment made up of many data points. Using point clouds, LiDAR-based systems can accurately identify obstacles, open paths, and other environmental features essential for navigation. To further enhance autonomous capability, LiDAR works in conjunction with Simultaneous Localization and Mapping (SLAM) algorithms [8]. SLAM enables a vehicle to build a real-time map of its environment while simultaneously determining its position within it. This is particularly useful for autonomous ground vehicles because it allows them to navigate without external localization technologies such as GPS. LiDAR and SLAM are the two main autonomous features implemented in this AGV [3].

A key advantage of LiDAR navigation is its ability to detect and avoid obstacles in real time. By continuously scanning its environment, a LiDAR-equipped vehicle can identify objects, determine their size and position, and adjust its movement accordingly [8]. Path planning algorithms such as A* and RRT are used to determine optimal routes while dynamically adjusting speed and direction based on changing environmental conditions. This makes LiDAR well suited for autonomous systems operating in cluttered or changing environments.

In small ground vehicles, such as RC cars, mobile robots, and unmanned ground vehicles (UGVs), LiDAR plays a crucial role in autonomous navigation. Unlike large-scale autonomous cars, small ground vehicles operate in confined spaces and require lightweight, efficient navigation systems [11]. One of the primary applications of LiDAR in these vehicles is real-time mapping in unknown environments. Equipped with LiDAR, small vehicles can autonomously explore and generate detailed maps of unfamiliar terrains, a capability particularly valuable for search-and-rescue missions, warehouse automation, and military applications. LiDAR's high-precision obstacle detection is another major advantage, as it can detect small and complex objects far more accurately than ultrasonic or infrared sensors [12].

Another important aspect of LiDAR navigation is its effectiveness in indoor and GPS-denied environments. In settings where GPS is unreliable or unavailable, such as tunnels, dense forests, or industrial facilities, LiDAR enables accurate positioning and movement. When integrated with ROS 2 (Robot Operating System 2), LiDAR can efficiently process point clouds, run SLAM algorithms, and optimize navigation paths. The ROS ecosystem simplifies the development of intelligent autonomous systems by providing pre-built tools and libraries

for working with LiDAR data, making it easier for developers to implement real-time mapping and motion planning [13].

For small ground vehicles, LiDAR offers several practical advantages. It provides high accuracy compared to other sensing technologies, allowing for precise environmental mapping. It is also effective in all-weather conditions, functioning reliably in dark, foggy, or low-visibility settings where cameras and infrared sensors may struggle. Additionally, certain LiDAR models offer 360-degree coverage, capturing a panoramic view of the surroundings for better situational awareness. Its adaptability makes it suitable for various applications, including industrial automation, sports tracking, and military defense systems [14].

Chapter 2: Literature Study and Process Considerations

2.1 Materials for High-Impact Durability

Selecting the right materials is important for ensuring the structural integrity, durability, and longevity of an autonomous ground vehicle, especially one that will be used in high-impact environments. The chassis and structural components of the vehicle must withstand collisions, mechanical stress, and environmental factors while maintaining a lightweight design to ensure efficiency. This project incorporates structural components for hardware installation, vehicle safety measures (bumpers), and object-deflection protection [5].

Additive manufacturing (3D printing) will allow for rapid prototyping and optimization of parts. The primary materials considered for high-impact durability include nylon, carbon-fiber-reinforced polymers (CFRP), and thermoplastic polyurethanes (TPU). Nylons are

widely used in robotics due to their high strength-to-weight ratio, impact resistance, and thermal stability. When reinforced with carbon fibers, CFRP offers superior rigidity and durability, making it suitable for structural components like motor mounts and protective casings for electronics [6]. Thermoplastic polyurethane (TPU) is an excellent material for shock-absorbing elements. TPU exhibits high elasticity and abrasion resistance, making it an ideal choice for protective covers and impact-resistant structures. The choice of materials in this project will be guided by considerations such as mechanical stress, fatigue resistance, and ease of manufacturability [6].

2.1.1 Stress and Fatigue Testing in 3D-Printed Components

In high-impact environments, stress and fatigue testing are needed to assess the long-term durability of 3D-printed parts used in SAGVs. Stress testing evaluates a material's ability to withstand static and dynamic forces, while fatigue testing simulates repeated loading cycles to determine how a material degrades over time. Given the expected operational conditions of this LiDAR-equipped RC ground vehicle, stress and fatigue testing will be critical in ensuring that key structural components maintain their integrity over prolonged use [5].

One of the most common methods used for stress testing is tensile testing, which measures a material's ability to endure stretching forces before failure. Additionally, impact testing will be used to evaluate the material's resistance to sudden impacts, simulating collisions the vehicle may encounter. For fatigue testing, cyclic loading experiments will be conducted to assess the lifespan of printed components subjected to repeated forces, ensuring they do not fail prematurely. The materials chosen for this project, including carbon-fiber-reinforced

nylon and TPU, will be tested under various environmental conditions, such as exposure to temperature changes, moisture, and mechanical shocks. The goal is to ensure that the parts can endure stress without significant wear or failure, making them viable for use in military applications or rugged terrains. Additionally, computational modeling with FEA will provide insights into the weakest points in the design, allowing for reinforcement in future iterations [6].

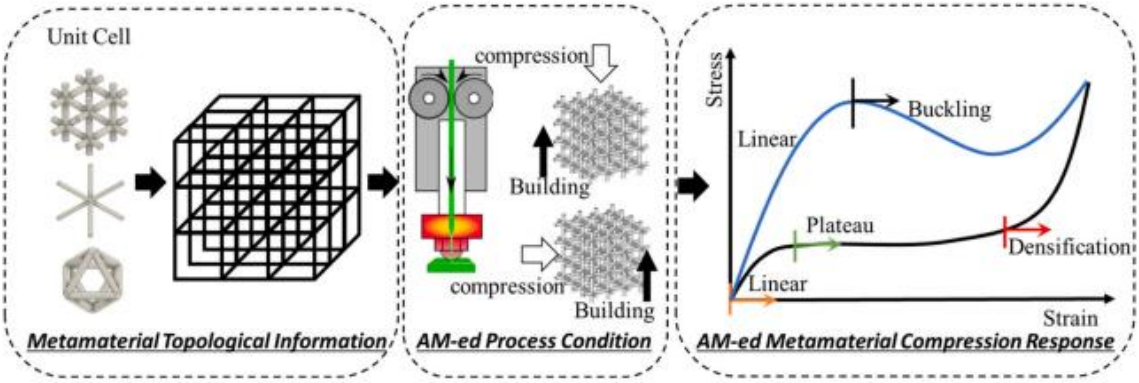


Figure 2.1.1-1: Example Workflow for Material Analysis [5]

By evaluating stress and fatigue performance, this project will ensure that 3D-printed components maintain high durability while optimizing weight and manufacturability. The ability to rapidly iterate and improve designs through additive manufacturing provides a distinct advantage, allowing for cost-effective customization of parts for high-performance autonomous systems [5].

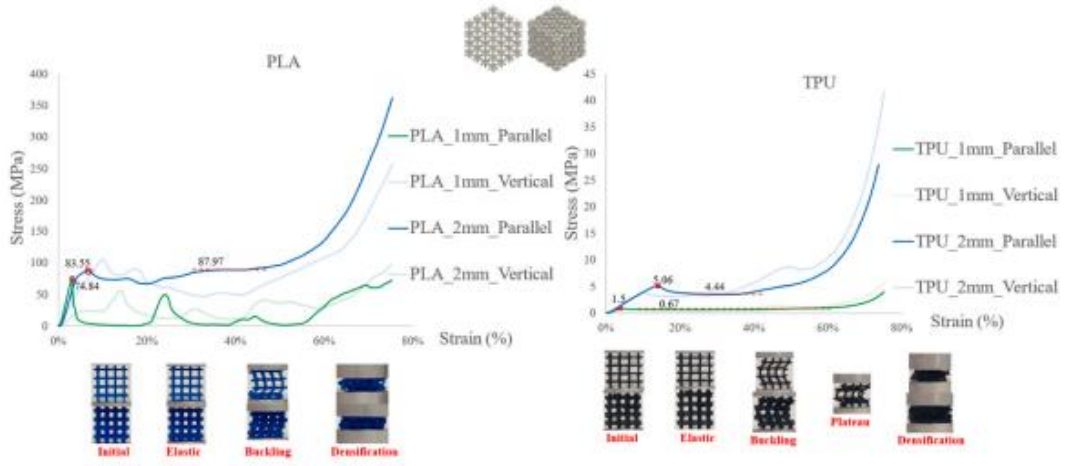


Figure 2.1.1-2: Example of PLA and TPU Compression Behavior with Cross Cube Lattice [5]

2.2 Introduction to ROS (Robot Operating System)

ROS 2 is a framework of software libraries and tools for building robot applications, effectively functioning as an "operating system" for robots. It provides standardized methods for communication between subsystems, such as a camera node sending data to a navigation algorithm. Compared with ROS 1, ROS 2 uses the Data Distribution Service (DDS) as its middleware. This DDS-based design provides several advantages that are important for autonomous vehicles [13-18][22].

- Real-Time and Reliability: ROS 2's DDS communication is designed for high efficiency and low latency, allowing messages to be delivered quickly and reliably. It removes the single point of failure (the ROS 1 setup) by letting nodes communicate peer-to-peer, which is best for safety in a moving vehicle. [14][15][18]
- Multi-Platform and Multi-Language Support: ROS 2 is cross-platform (Linux, Windows, Mac) and built on a new library that makes it easier to support multiple

languages. This means developers can write code in Python, C++, and even Java if needed, which broadens who and what can be utilized with ROS. [16]

- Better Use of Hardware Resources: ROS 2 natively supports multi-threaded execution, allowing it to leverage modern CPUs more effectively. For example, on an NVIDIA Jetson with multiple CPU cores, ROS 2 can run several tasks in parallel without major bottlenecks. This is important for computationally intensive functions such as real-time sensor processing and SLAM [17].
- ROS 2 also supports RViz, which allows precise visualization and testing of AGV control and navigation methods. The simulator window shown below depicts a preset map with a LiDAR-enabled AGV in the frame [13].

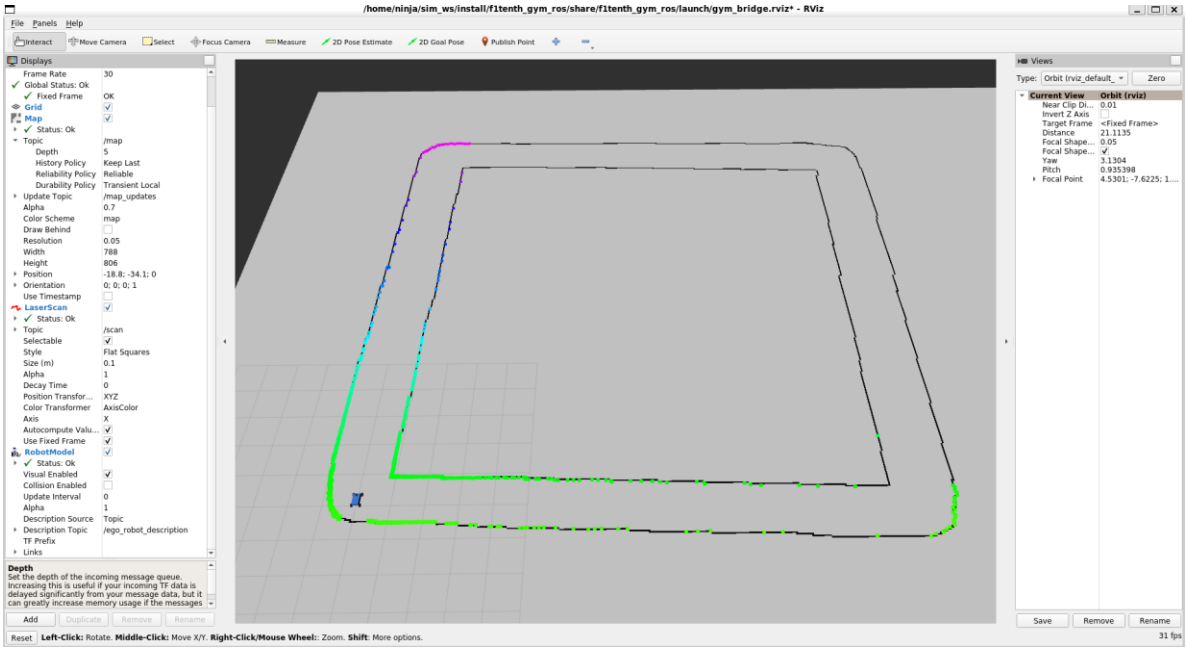


Figure 2.2-1: RViz Simulator [7][13]

Figure 2.2-1 shows the main interface window for the ROS-based simulator used for developing SLAM algorithms and AGV code for this project. On the left is the tool window

that provides access to sensors, vehicles, and maps. The default map can be seen in the middle [7][13].

ROS 2 Architecture: Nodes, Topics, and Messages

A core idea in ROS is to break the robot's software into small pieces called nodes. Each node is like a mini program with a specific role: one might read a sensor, while another makes driving decisions. ROS 2 nodes communicate by sending messages over topics, similar to radios using different frequencies as separate channels. This architecture is often described as a publish/subscribe model: a sensor node publishes data messages on a topic, and other nodes subscribe to that topic to receive the data. The AGV uses the following node structure [19-21][22]:

- A LiDAR driver node publishes LaserScan messages on the /scan topic, which carries the latest distance readings from the laser. Any node interested in perceiving obstacles or mapping can subscribe to /scan. [13]
- The vehicle's odometry (distance traveled and speed) is published by another node connected to the motor controller, called the VESC, on the /odom topic as Odometry messages. This provides other components with information about how the car has moved [20].
- Control commands are published to the /drive topic. Nodes that make driving decisions publish AckermannDriveStamped messages to /drive, which the motor controller uses to adjust steering angle and throttle. The Ackermann message contains fields for the desired speed and steering angle [21].

ROS 2 topics separate the producers and consumers of data. This means a subsystem can be changed without breaking the rest of the stack. For instance, different algorithms for wall-following or race-line generation can subscribe to /scan and /odom and publish to /drive. As long as the topics and message types remain consistent, ROS 2 handles the communication [22]. This modular design makes it easier for teams to integrate perception, planning, and sensor-control components into one working system. Developers can visualize this network of nodes and topics using tools such as rqt_graph or RViz, which show how data flows through the system. ROS 2 therefore provides the communication layer that allows sensors, computers, and actuators on the car to operate in real time [23].

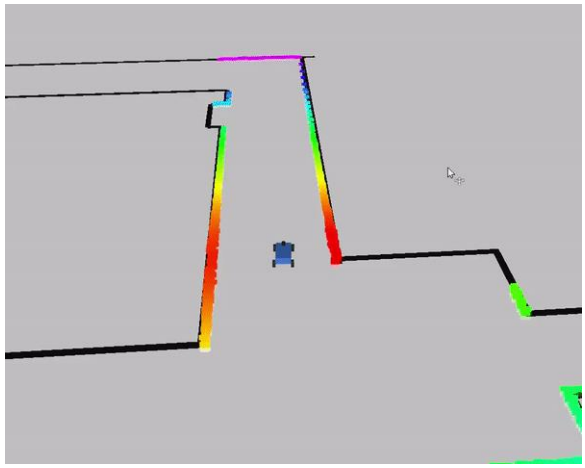


Figure 2.2-2: RViz with LiDAR Scan Intensity Visualization [9]

Figure 2.2-2 shows an isometric view of the default LiDAR car being driven around the default map in RViz. The coloring on the walls represents scan intensity, with red indicating higher intensity and blue indicating lower intensity. The default RViz setting uses 10 Hz, while the LiDAR on the AGV will run at 20 Hz [13].

Autonomous driving involves multiple complex functions: understanding the environment (perception), merging data from different sensors (sensor fusion), deciding where to go (planning), and controlling the vehicle to get there (control). ROS 2 offers both the infrastructure and packages to implement them [24]:

- Sensor Fusion: In robotics, no single sensor is perfect. For example, AGVs have wheel encoders (odometry) and often an IMU (inertial measurement unit) in addition to LiDAR. ROS 2 provides packages like robot_localization that combine data from wheel odometry and IMU using an Extended Kalman Filter, giving a more accurate estimate of the car's position than either sensor alone. This helps with localization even if the wheels slip or sensors get noisy. In ROS 2, the odometry data might come in on /odom and IMU data on another topic; a sensor fusion node subscribes to both and publishes a filtered odometry on, say, /odom_filtered. All of this happens in real time with ROS 2's efficient messaging, so the planning system always has the best possible state estimate. [25]
- Perception: Beyond just mapping with SLAM, perception can include detecting obstacles or features. The ROS 2 ecosystem has packages for point cloud processing, lane detection, and more. A common form of perception is building a 2D occupancy grid from LiDAR, essentially marking where there are walls or objects. If multiple perception sources exist (say LiDAR and camera), ROS 2 can help fuse them or at least manage them concurrently without one blocking the other. [26]
- Motion Planning: Once the car knows where it is and has a map of its environment, it needs to plan a route or trajectory. ROS 2 provides the Navigation 2 (Nav2) framework, which can handle path planning, path following, and collision avoidance

for ground robots. Nav2 can take a goal point and produce a safe path on the map, then send velocity/steering commands to follow that path. Nav2 is ideal for the testing and end goal of the AGV project. Some other projects use custom planning algorithms geared for racing. For example, computing an optimal racing line around a track, or using Rapidly Exploring Random Trees (RRT*) to avoid obstacles on the track. These planning components are implemented as ROS 2 nodes. Because ROS 2 supports real-time control loops, the car can adjust its steering every few milliseconds as it moves. The planning and control nodes operate in tandem: one plans the path, the other executes it, communicating over ROS 2 topics. [26]

ROS 2 enables sensor fusion by synchronizing multiple data sources, supports perception by distributing sensor data to the appropriate processing nodes, and coordinates planning and control by allowing those modules to share information. The result is a cohesive autonomy stack for AGV navigation. LiDAR scans become maps, maps and odometry become position estimates, and positions and goals become steering and speed commands [22][23].

2.2.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the process of building a map of an unknown environment while estimating the vehicle's pose within that same environment in real time. For the semi-autonomous ground vehicle (SAGV), SLAM solves the core problem that odometry alone cannot: wheel encoders and inertial sensors drift over time, so the estimated position diverges unless corrected by observing the environment and enforcing global position with consistency. [19][23]

Technically, most LiDAR SLAM pipelines are organized into a front end and a back end. The front end converts raw range measurements into incremental motion estimates by matching the current scan against a local representation of the world. This scan-matching step produces a relative pose constraint between consecutive timestamps, often augmented by a motion prior from wheel odometry and IMU. The back end then takes many such constraints and solves a global optimization problem, typically a pose graph, in which nodes represent vehicle poses and edges represent relative measurements. When the system revisits a previously mapped area, loop-closure detection adds long-range constraints that pull the trajectory into alignment and reduce drift. The resulting optimized trajectory is used to update a global map representation such as an occupancy grid or a higher-resolution point-cloud map. Demonstrations of LiDAR-based SLAM in compact, high-speed ground vehicles like the SAGV have shown reliable performance in constrained environments [19].

In a ROS 2-based system, SLAM is expressed as a set of nodes connected by topics and transforms. A LiDAR driver publishes scans, an odometry node publishes incremental motion, and a SLAM node subscribes to these streams to calculate poses and publish the map and the map-to-odom transformation. ROS 2's publish/subscribe model enables each component to run independently with unified graphical interfaces, which improves testability (replaying logs), modularity (swapping SLAM algorithms), and real-time integration with planning and vehicle control settings. [22]

For the SAGV, SLAM is ideal because it provides localization even when GPS is unavailable by generating a map that downstream navigation modules can use for path planning. A practical architecture fuses wheel odometry and IMU data in an extended Kalman filter to produce a smooth, high-rate state estimate, while SLAM provides drift-correcting global

updates through the transform tree. A localization package is commonly used for this multi-sensor fusion, providing a consistent state estimate that bridges local control needs with global corrections from SLAM [23]. This combination yields stable closed-loop driving, repeatable routes, and reliable autonomy in changing environments.

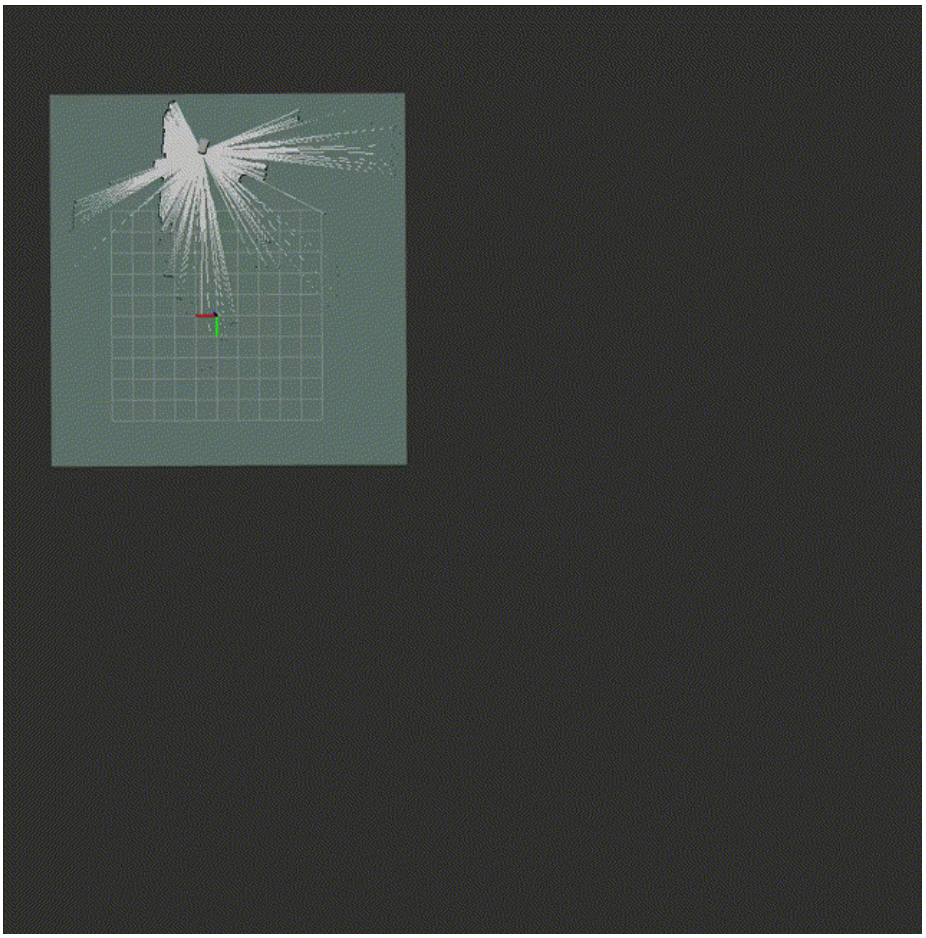


Figure 2.2.2-1: SLAM GIF [23]

Chapter 3: SAGV Design and Development

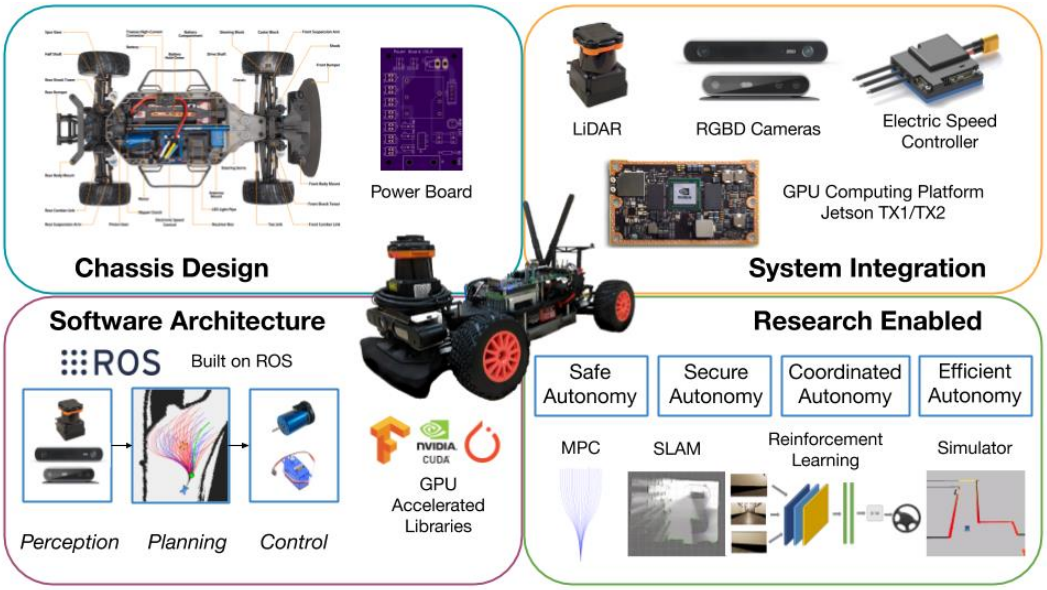


Figure 3-1: Overview of SAGV Project [3]

Component Installation: SAGV Chassis, Lower Deck, and Upper Deck Assembly

The SAGV chassis and autonomy stack were assembled by designing a repeatable conversion of a stock 1/10-scale Traxxas platform into an autonomy-ready vehicle with a modular two-deck architecture. The intent of this building phase was to create a rigid mechanical foundation with aluminum standoffs and platform deck, then integrate a serviceable upper deck that carries compute, power distribution, and sensing hardware. This section documents the physical process from teardown through final integration, and it notes the changes required to support a Jetson Orin Nano and Hokuyo UST-20LX as upgrades to the baseline open-source configuration [9].

Procurement and Build Planning

The build began with a procurement plan organized in a bill of materials (BOM) spreadsheet used to track parts, quantities, vendors, lead times, cost, and delivery status. The open-source documentation provides a baseline BOM for the chassis hardware, autonomy stack, and

standard fasteners required for deck assembly [9]. A spreadsheet-based tracker was used to avoid integration delays from missing small items such as standoffs, M3 screws, and cable adapters, and to document substitutions that were made during the upgrade to the Orin Nano and the UST-20LX. This approach also provided a record of revision changes as mounts and wiring were iterated during assembly.

red = already have, blue = key part *Scroll right to see links*	Qty	Cost per Unit	Comments	Link
Chassis				
Traxxas Fiesta ST Rally **VXL Edition**; 1/10	1	$429.95	Slash 4x4 Ultimate also	https://traxxas.com/products/landing/fiesta
2 Lipos and Charger Combo	1	$239.95		https://www.amainhobbies.com/traxxas-ez
Antenna Mount (3D print)	1	$10.00	Files found in google drive	https://drive.google.com/drive/folders/1sy-
Platform Deck (lasercut)	1	$20.00	Fits Orin Nano or Xavier NX	https://drive.google.com/drive/folders/1NU
Fasteners				
M3 Socket Head Kit	1	$22.98		https://www.amazon.com/iexcell-Metric-Sc
M5 Socket Head Kit	1	$8.66	You'll need 2 M5x20mm screws	https://www.amazon.com/iExcell-Stainless-
6mm hex 14mm M3 standoffs	2	$4.09		https://www.mcmaster.com/94868A009
6mm hex 25mm M3 standoffs	4	$5.98		https://www.mcmaster.com/94868A013
6mm hex 45mm M3 standoffs	6	$4.31		https://www.mcmaster.com/94868A190
M2.5 x 0.45mm, 6mm long	4	$5.75	You'll only need 4 screws. This s	https://www.mcmaster.com/94500A213/
M2.5 x 0.45mm, 10mm long FF Standoffs	4	$0.73		https://www.mcmaster.com/95947A005/
Compute module				
Jetson Orin Nano	1	$499.99	ebay for ~$100	https://www.sparkfun.com/products/22098
Micro SD Card 32 GB	1	$9.50		https://www.amazon.com/Samsung-Class-A
NVMe SSD Card 500 Gb	1	$75.00		https://www.amazon.com/BLACK-SN750-5(
Sensors				
Hokuyo 10LX	1	$1,670.00	ebay for ~$400	https://www.robotshop.com/en/hokuyo-ust
Electronics				
VESC 6 MkV	1	$259.00		https://trampaboards.com/vesc-6-mkv-in-c
Joystick Opt 1: Logitech	1	$39.00	Choose between this or the PS4	https://www.amazon.com/Logitech-940-00
Joystick Opt 2: Dualshock 4 for PS4	1	$69.99	(Preferred joystick)	https://www.amazon.com/s?k=ps4+control
LiPo safety bag like the Aketek Silver Large Size	2	$9.99	battery catches fire or leaks (as	https://www.amazon.com/Fireproof-Battery-D
Barrel Jack to Pigtail	1	$4.33	Read this for more information	https://www.digikey.com/product-detail/en/tensility
12V 5A Power Adapter (optional)	1	$11.99	Read this for more information	https://www.amazon.com/Adapter-100-220V-S
Antenna	1	$8.99		https://www.amazon.com/Antenna-MHF4-IPE
Intel RealSense D345i (optional)	1	$334.00		https://store.intelrealsense.com/buy-intel-rea
Miscellaneous				
TRX to XT90 Adapter	1	$9.99		https://www.amazon.com/dp/B07QMJ8XG
Traxxas id charge lead adapter	1	$15.99		https://www.amazon.com/Charger-Cable-A
Header Pins	1	$8.99	You'll only need 3 header pins.	https://www.amazon.com/DIKAVS-Break-H
VESC ppm cable	1	$6.95		https://www.ollinboardcompany.com/prod
DP emulator	1	$6.99		https://www.amazon.com/FUERAN-DP-Disp
short (~1 ft) A USB-to-USB C cable	1	$8.99	sold in bulk (pack of 5)	https://www.amazon.com/Durable-Chargin
Bullet Adapter 4mm Male 3.5mm Female	1	$9.19		https://www.amainhobbies.com/bullet-ada
	Total:	$3,823.45	(estimated)	

Figure 3-2: Bill of Materials made for the SAGV

Lower-Level Chassis Teardown

Traxxas disassembly began by removing the body shell and extra brackets, stripping the RC vehicle down to the mechanical subsystems needed for driving. Stock electronics were removed to free space for the autonomy stack, including the motor controller, receiver, and

associated wiring. The steering servo was retained to preserve the factory steering linkage and simplify later calibration [9]. Fasteners removed during teardown were saved and sorted because multiple screws are reused when installing the standoffs and reattaching structural features [9]. After removal, the chassis was inspected for drivetrain binding, loose steering components, and interference points that could affect cable routing and deck placement.

Lower Deck Setup and Standoff Installation

The lower deck work establishes the mechanical interface that supports the autonomy stack. Chassis standoffs are installed into existing mounting points to create rigid posts that later support the platform deck [9]. These standoffs set the upper-deck height and define the space available for battery placement, wiring runs, and connector access. At this stage, battery placement was standardized and secured because the battery is the heaviest component and the main power source for the system. The pack was positioned to keep the vehicle balanced and was restrained so it could not shift during acceleration, braking, or impacts.

Upper Deck Fabrication and Layout

The upper deck, or platform deck, is the modular plate that carries the autonomy components. For the SAGV, the platform deck was manufactured as a laser-cut acrylic plate using a DXF layout, which allowed mounting hole locations to be updated quickly as components and cable interfaces changed. Before mounting hardware, all components were test-fit on the deck to verify connector clearance, body-shell clearance, and cable routing. The layout priority was to keep high-current wiring short and protected, while ensuring the Jetson remained accessible for USB, Ethernet, and power connections.

Figure 3-3: Torn Down Chassis with Platform Deck

Manufacturing Mounts and Brackets

Several small fixtures were manufactured to support the deck assembly, including printed brackets for the antenna mount, a protective Jetson cover, and adapters used to match sensor hole patterns to the acrylic deck. These parts were designed in CAD and produced through FDM printing so they could be revised quickly during fit checks.

Mounting Sequence on the Platform Deck

Hardware was installed onto the upper deck in the same order used in the open-source guide to keep wiring manageable and to preserve the intended packaging. The VESC was mounted first and positioned toward the rear of the deck to reduce motor lead length and isolate high-current wiring from the computer hardware [9]. The VESC was mounted on standoffs or spacers to protect the PCB and to avoid contact between solder joints and the deck surface. After the VESC, the WiFi antenna mount was installed, and antenna cables were routed so they could be connected to the Jetson before other components blocked access [9].

Figure 3-4: Vedder Electronic Speed Controller (VESC)

Computer Mounting and Orin Nano Adaptations

This project adapted the deck to mount an NVIDIA Jetson Orin Nano. The mechanical approach remained consistent with the guide: the computer board was mounted using standoffs to keep the PCB rigid and to protect the underside while maintaining airflow around the fan and heat sink [9]. The Orin Nano has different connector locations, so hole locations and component spacing were adjusted to preserve access to Ethernet for the LiDAR, USB for the VESC interface, and any peripheral connections needed for bringup. Power routing was also considered during placement because the computer supply must remain stable during motor transients.

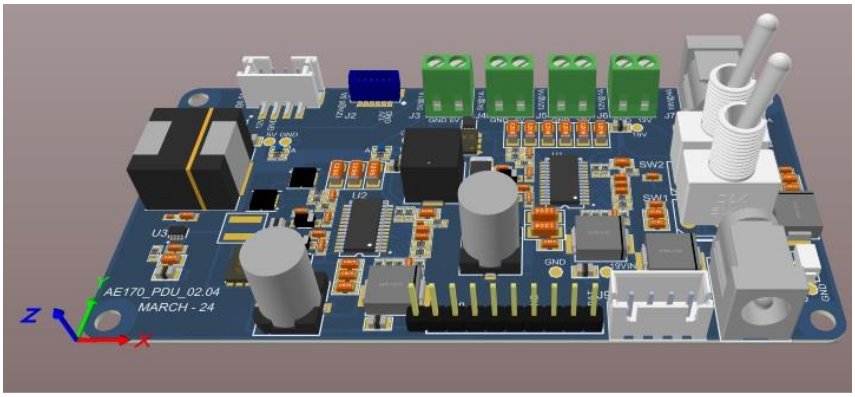


Figure 3-5: Ambimat PCB Model

Lidar Mount And UST-20LX Integration

The LiDAR was mounted near the front of the upper deck to maximize forward field-of-view and keep the scan plane clear of vehicle structure [9]. This build used a Hokuyo UST-20LX as an upgraded sensor. The primary mechanical change was adapting the mount geometry to the UST-20LX footprint while keeping the sensor level and rigid. Unlike common USB LiDARs, the UST-20LX uses Ethernet, so the cable routing strategy was designed around a strain-relieved Ethernet run to the Jetson and a separate power feed from the PCB. Power was supplied from the system battery through the vehicle power distribution path, with regulation used as required to keep the sensor within its input range [9].

Joining The Upper and Lower Decks

With the autonomy components mounted, the upper deck was joined to the lower chassis standoffs. Fasteners were installed through the deck into the standoffs, and a clearance check confirmed the battery could still be inserted and removed and that the body shell could be installed without contacting the hardware. The rear-facing orientation of the VESC was maintained so motor leads could be routed directly to the drivetrain without wiring interference from other components.

Final Electrical Integration and Safety Configuration

After the decks were connected, final electrical integration was done. Motor phase wires were connected to the VESC, then the battery input was connected through the PCB and VESC, so both the drivetrain controller and compute stack were powered from a single pack [9]. The Jetson was connected to the VESC using the interface used by the ROS driver stack,

enabling command and telemetry exchange during teleoperation and autonomy [13][20]. The LiDAR Ethernet cable was connected to the Jetson and secured so it could not unplug during handling. A manual control path was preserved by routing a PPM connection from the RC car servo/receiver to the VESC, maintaining a direct failsafe mode for early testing and safety during development [9].

Build Verification

Initial power-on was performed with the vehicle lifted so the wheels could spin freely. Basic checks verified correct steering direction, motor response, stable power delivery, and LiDAR communication to the Jetson before higher-level autonomy software was introduced. Completing this assembly provided a stable hardware platform for the driver stack integration described later in Chapter 3 and for the autonomy methods developed in Chapter 4 [13][22].

Figure 3-6: Completed SAGV Build with Cover

3.1 Ground Vehicle Bumper Design

The RC Car V1 was initially designed to be rugged, cheap and easy to drive around softball bases. The HOSIM was selected for these reasons as well as its 2600kv motor and wide wheelbase. As mentioned in the introduction, the car is 22x12 inches in size. Small enough to be portable and cheap, but still large and durable enough to be used outdoors and handle repeated impacts with walls and base plates. Below is a summary of the main components for the build and pictures to go along with them:

- HOSIM 1/10th scale RC Car was utilized for initial setup
- The ESC was changed and tuned using castle link
- 1/8th inch aluminum was cut out on a waterjet to make an undertray and mounting plate
- Impact areas (IAs) were made in SolidWorks and mounted to the front and rear bumpers
- A pool noodle mount was made in SolidWorks and bolted on top of the IA. A dowel rod would be inserted in the mount and the other end inside the pool noodle.
- A bumper mounting system was built using foam cushioning and PVC for the external frame wrapping around the car.

Figure 3.1-1: Front End of Aluminum Plate and Black Printed Parts for Impact Area

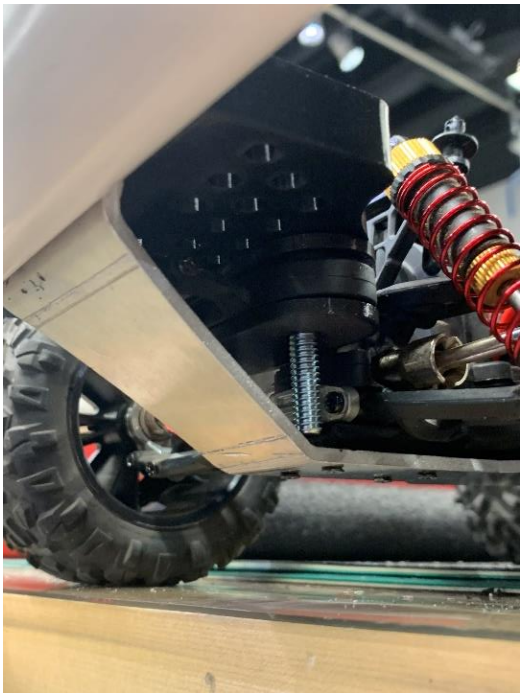


Figure 3.1-2: Impact Area (big black part) and Underside of Aluminum Mounting Bracket

Figure 3.1-3: Top Side of Impact Area and PVC Pipes used as Bumper Frame

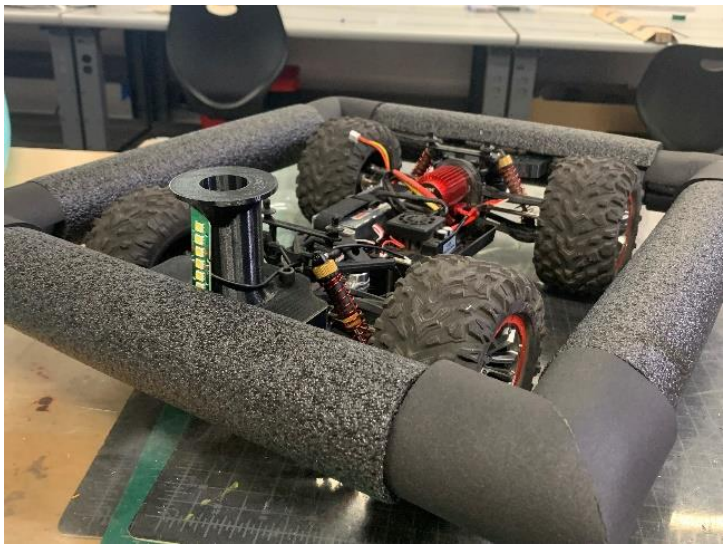


Figure 3.1-4: Assembled & Cushioned Bumper System and Pool Noodle Mount (part with lights)

Figure 3.1-5: Trimmed and Painted Body for OU Softball (squares on sheet are 1/2" inch)

3.2 Additive Manufacturing for Component Design

Multiple components have been printed for use in assembling LiDAR and autonomy stack components as well as for mounting targets to the frame of the vehicle. Images of these components are as follows:

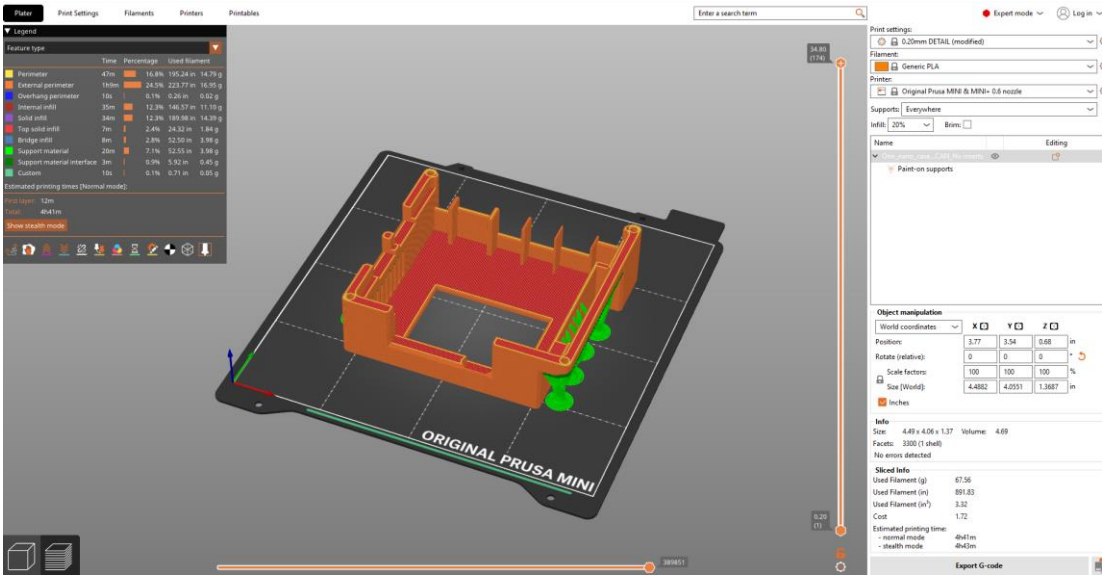


Figure 3.2-1: Jetson Orin Nano Cover Slice

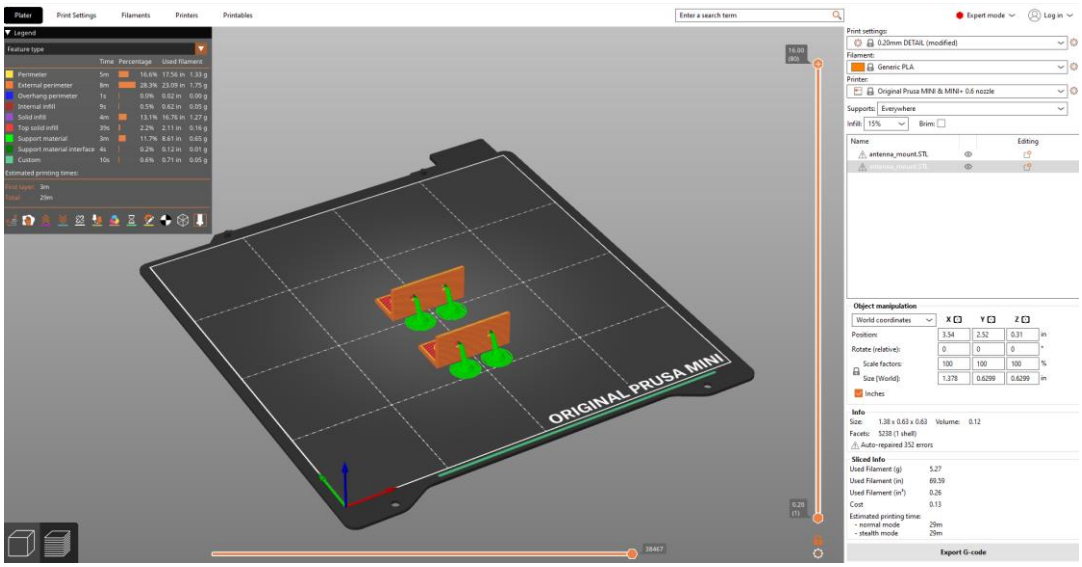


Figure 3.2-2: Antenna Mount Bracket Slice

Figure 3.2-3: Target Mount Print

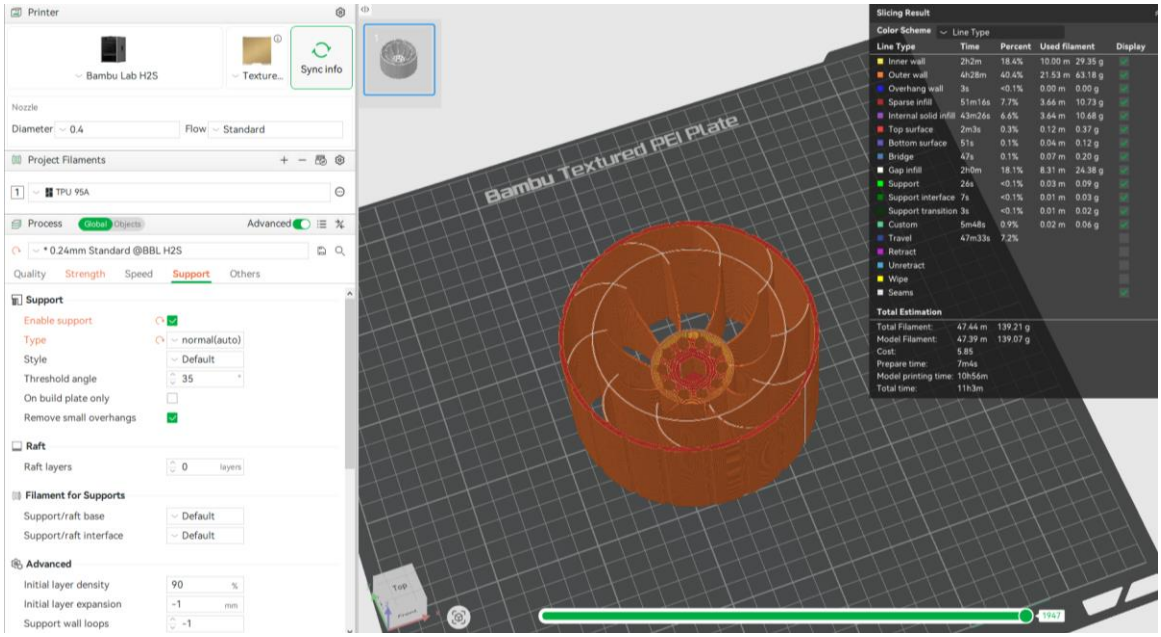


Figure 3.2-4: Sliced Airless TPU Tire

3.2.1 Design of LiDAR Mounts and Structural Components

The full autonomy stack was assembled onto a platform deck for modularity and easy installation. The platform deck was laser cut from continuous-cast acrylic using a DXF file. The antenna mounts and autonomy-stack mounting holes can be seen in Figure 3.2.1-1.

Figure 3.2.1-1: Platform Deck and Antennas

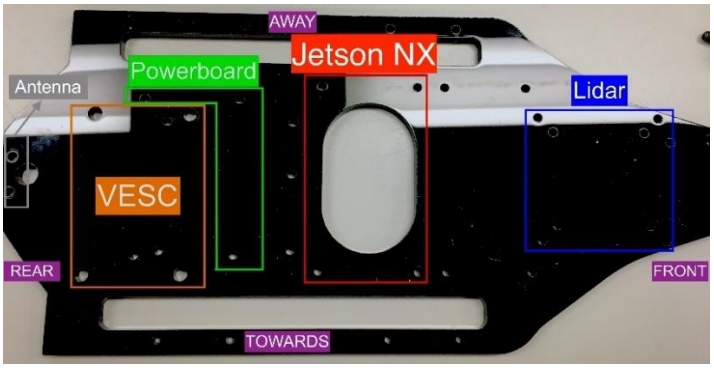


Figure 3.2.1-2: Autonomy Stack Arrangement

Although this project iteration focuses mainly on Castle Link tuning for softball use, the second version of the car is already in development. Figure 3.2.1-3 shows the current mockup assembly of the AGV with the platform deck and labeled holes for electronics, the antenna mount, the bumper, and the target system.

Figure 3.2.1-3: Mockup Assembly with Autonomy Stack and Bumper System on the HOSIM

3.2.2 TPU Impact FEM with SolidWorks

To extend this framework toward applied additive manufacturing, this project focused on the design and performance evaluation of 3D-printed airless tires fabricated from thermoplastic polyurethane (TPU) for radio-controlled (RC) vehicles [30][35]. The work involved developing multiple lattice and spoke geometries in SolidWorks CAD, performing finite element simulations to examine stress distribution, deformation behavior, and load response, and assessing manufacturability through additive fabrication trials [30][33]. Emphasis was placed on balancing structural integrity, flexibility, and print efficiency to achieve a reliable alternative to traditional pneumatic tires [29][36]. The effort demonstrates the feasibility of using low-cost, customizable airless tires as durable and maintenance-free components in cost-efficient, small-scale mobility systems [35][37].

The intent of this section is to explore how additive manufacturing and computational modeling can be used not only to optimize performance, but also to develop a design that is marketable and accessible to anyone with an RC vehicle and access to a 3D printer [30][35]. This idea came from the growing potential of low-cost fabrication technologies to deliver customizable and sustainable engineering solutions at the consumer level [30][33]. The overall design methodology is presented, including material research, CAD modeling, and simulation studies [30][35].

Mechanical Performance Requirements

The functional performance of the 3D-printed TPU airless tire was established through a set of criteria aimed at ensuring structural reliability, resilience, and long-term usability under

dynamic operating conditions [36][37]. Since the tire must sustain repeated loading cycles during continuous rolling and turning, its material and geometry were designed to achieve high impact durability, resistance to permanent deformation, and stable performance over extended operating time [29][35]. From a mechanical standpoint, the design focused on maintaining stress levels below the material's yield strength, where TPU's elastic recovery remains dominant [31][32]. According to literature, FDM-grade TPU exhibits a yield stress of 25–35 MPa, depending on print orientation and infill density [31][32]. Finite element analysis (FEA) of von Mises stress distribution identified localized stress concentrations at lattice junctions and spoke–hub interfaces [36][37]. To ensure full elastic recovery under static loading, the maximum stress was kept below 80% of the yield stress (≈22.4 MPa), providing a safety factor of about 1.25 and reducing the risk of plastic deformation or micro-crack formation [36].

Approach/Research

This work started with research into airless tire technologies and the mechanical behavior of thermoplastic polyurethane (TPU) for flexible, load-bearing structures [29][36]. The goal was to combine additive manufacturing and computational analysis to create a durable, low-maintenance tire optimized for RC cars and small ground vehicles [30][35]. Existing concepts such as Michelin's Uptis and Resilient Technologies' Non-Pneumatic Tire (NPT) were reviewed to understand how lattice and spoke geometries replace internal air pressure while maintaining stiffness and flexibility [29]. Material characterization of TPU was a key focus of the project, since understanding its mechanical properties was needed for accurate simulation and performance optimization [31][32]. The material was defined with an elastic

modulus of 25–35 MPa, Poisson's ratio near 0.48, and a yield stress around 30 MPa [31][32]. These values were applied in finite element simulations to analyze von Mises stress, deflection under static loading, and energy absorption, ensuring the design remained below 80% of the yield stress to maintain full elastic recovery under repeated loads [36]. In parallel, market comparisons and open-source 3D-printed tire models were examined to establish realistic geometric constraints and consumer expectations [37].

Technical Approaches

Finite Element Analysis (FEA) and motion simulations in SolidWorks were used to predict stress distribution, deflection, and contact behavior [37]. Multiple iterative designs were explored, evaluating structural performance under static and dynamic loading conditions [34][38]. The final design was tailored to balance flexibility, stiffness, and manufacturability while achieving the desired ride quality [31][33].

Design Process

- Wheel Shape/Form: Four initial prototypes featured varied internal geometries—spoke, honeycomb, helical, and hybrid lattice structures—to study load paths and compliance [29][37].
- Functionality: Each version aimed to replicate pneumatic tire cushioning while maintaining traction and stability [29][36].
- Manufacturability: Designs were optimized for FDM 3D printing using PLA for fit checks, then TPU for functional testing [30][32].

- Accessibility: The final design used open-source materials and standard RC mounts for easy replication [35].

From a production standpoint, print efficiency and reliability were improved through layer adhesion optimization, reduced support usage, and print orientation tuning [32]. Maintaining a constant wall thickness supported repeatable, high-quality prints with minimal post-processing [31][32]. The design's parametric structure allows rapid scaling and customization, enabling adjustments to lattice spacing, tire width, and hub fit for different RC car models [33][34].

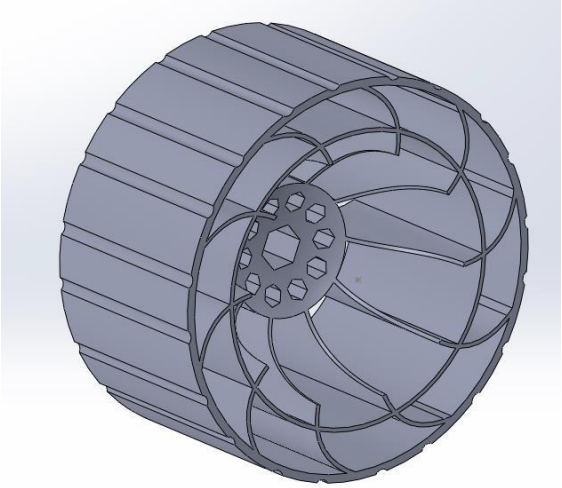


Figure 3.2.2-1: Chosen Design

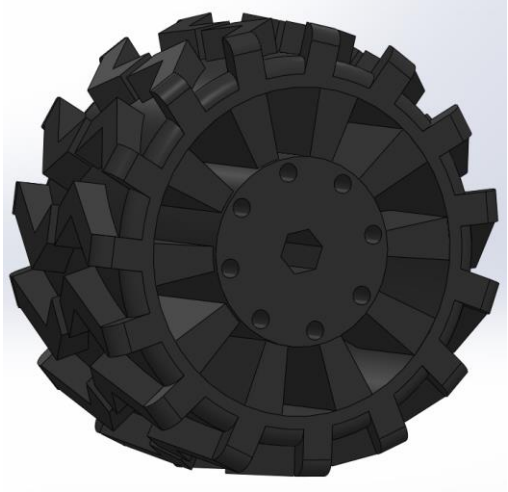


Figure 3.2.2-2: Design 2

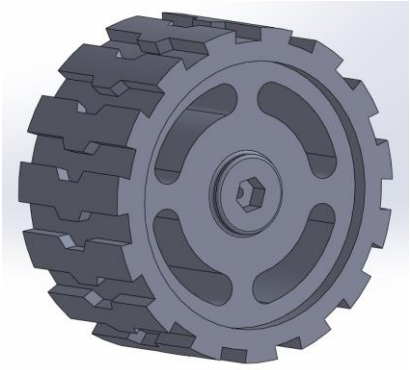


Figure 3.2.2-3: Design 3

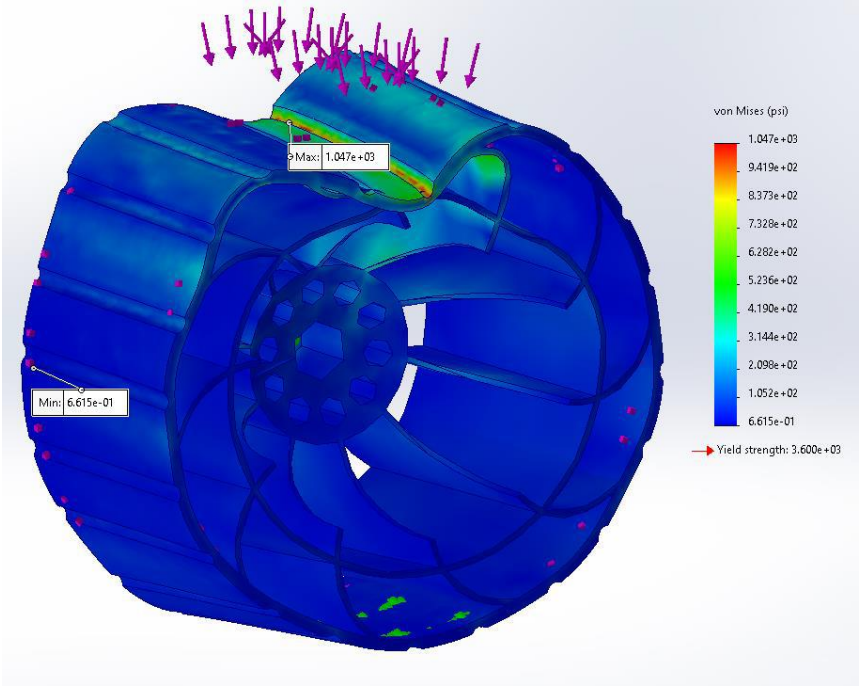


Figure 3.2.2-4: Von Mises Stress on Design 1

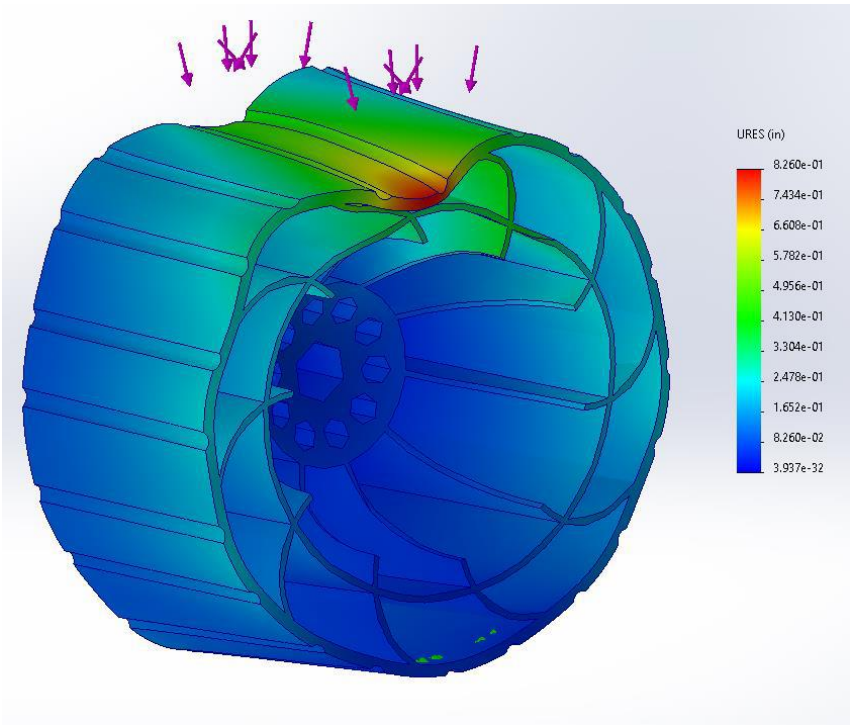


Figure 3.2.2-5: Displacement on Design 1

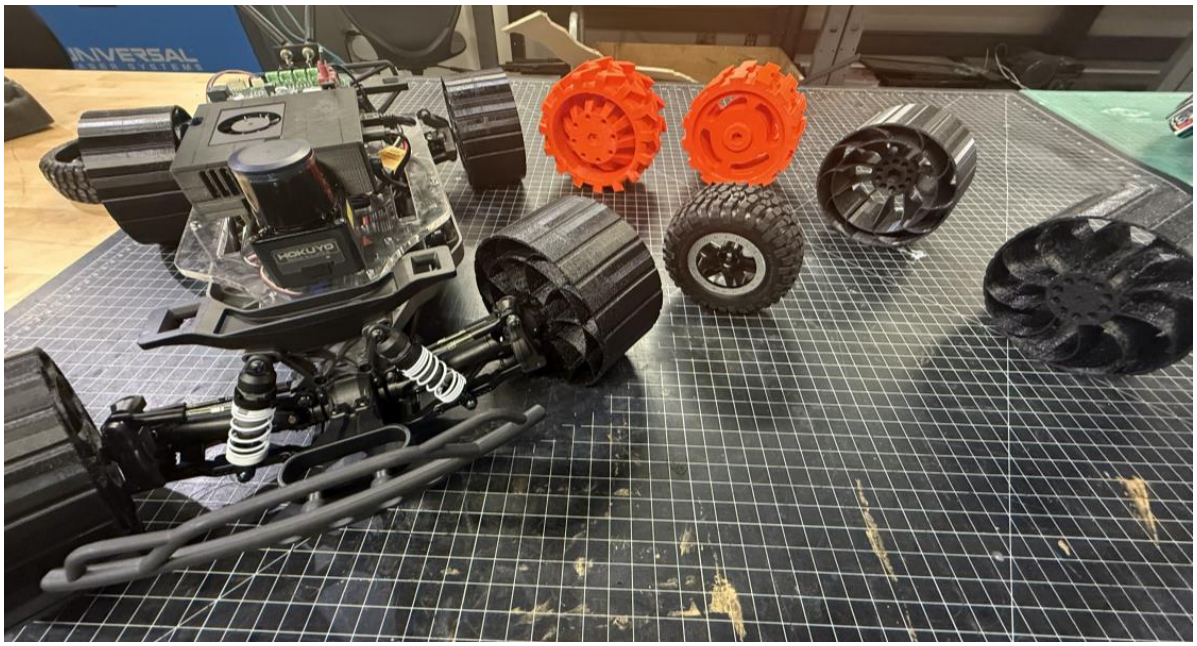


Figure 3.2.2-6: Wheel Model Display next to SAGV

3.3 Driver Stack Integration

The “driver stack” is the ROS bringup layer that interfaces directly with vehicle hardware and exposes it through standard ROS topics and services for higher-level autonomy nodes. It typically includes Linux device setup such as udev rules for stable /dev paths, ROS driver nodes for the motor and steering controller, LiDAR, and joystick teleoperation, plus the launch and configuration files that start and parameterize these components into readable groups. Functionally, it translates hardware I/O into ROS data streams (e.g., /scan, /odom, telemetry) and accepts command inputs, commonly through an Ackermann drive command topic, so planning and control stacks can operate without hardware-specific code [13][20][25].

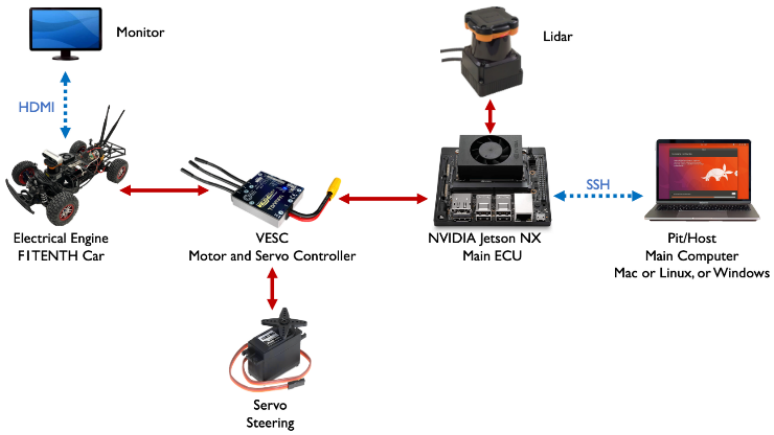


Figure 3.3-1: Diagram of System Interactions

3.3.1 Configuring the Jetson

The NVIDIA Jetson Orin Nano is a powerful embedded computer for AI, machine learning, and robotics applications and is a core component of the SAGV integration. The selection criteria and configuration process are summarized below.

The SAGV compute stack is centered on the Jetson, so the initial effort is best understood as establishing a reliable embedded Linux platform with deterministic access (power, storage, network, and remote administration) rather than as a sequence of one-off setup actions. The required hardware is:

- Pit/Host computer
- SDSH Card
- Terminal/SSH client computer

This supports three core objectives: (1) provisioning the Jetson with a known software baseline, (2) enabling repeatable connectivity for deployment, and (3) ensuring the system remains stable under the memory and computing loads typical of LiDAR navigation workloads.

Base Image Provisioning and Storage Strategy.

The Jetson Orin Nano is provisioned using NVIDIA's official developer-kit image and flashed using the Software Development Kit Manager (SDKM). Using the vendor image is not simply a convenience; it constrains early variability by providing a kernel, drivers, CUDA-enabled user space, and support packages that are already validated together. The storage device functions as the medium for initial bring-up and the root filesystem as a performance and reliability choice. LiDAR navigation pipelines involve frequent logging, map serialization, and package builds, all of which benefit from higher I/O throughput and improved endurance compared with consumer microSD media.

Network Configuration as an Engineering Control

For development, SSH is the primary interface between the Pit/Host workstation and the vehicle. Wi-Fi is configured to provide stable addressing for remote access. Assigning a static IP to the Jetson's interface turns the vehicle into a predictable network node, which reduces iteration friction (no discovery step, fewer connection failures) and improves debuggability.

Memory Headroom for LiDAR Navigation Workloads

LiDAR-based navigation workflows can be memory-intensive, especially during compilation (e.g., ROS2 workspaces), map generation, and high-rate data logging. A swap file provides a controlled mechanism to increase virtual memory availability, reducing the likelihood that the system will terminate processes under memory pressure. The intent is not to substitute

swap for physical RAM, but to add resilience during worst-case bursts (build steps, simultaneous nodes, or debugging sessions) where brief overcommit is common.

Controller/Teleoperation Integration

Finally, installing gamepad driver support (e.g., a PS5 controller) closes the loop between development and field testing. For an autonomous ground vehicle, a reliable teleoperation path is a safety and validation requirement because it enables controlled manual intervention during mapping and navigation trials, provides a fallback mode during integration issues, and supports repeatable test procedures in which human-in-the-loop driving is needed to collect datasets or validate localization behavior. This requirement is consistent with the project's semi-autonomous design intent.

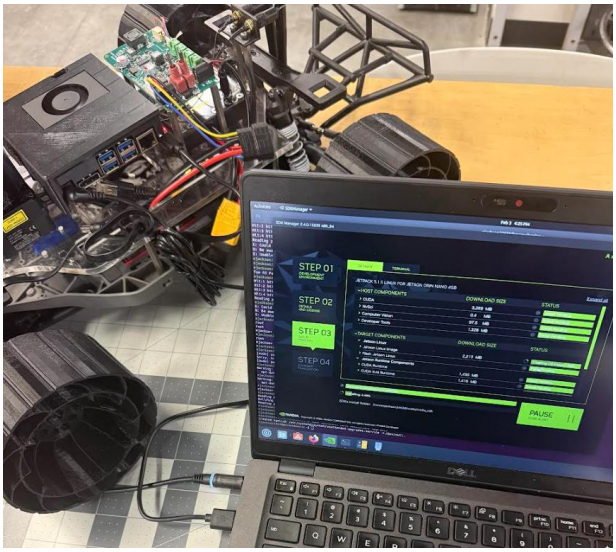


Figure 3.3.1-1: Flashing via SDKM

Figure 3.3.1-2: NVME SDSH Card

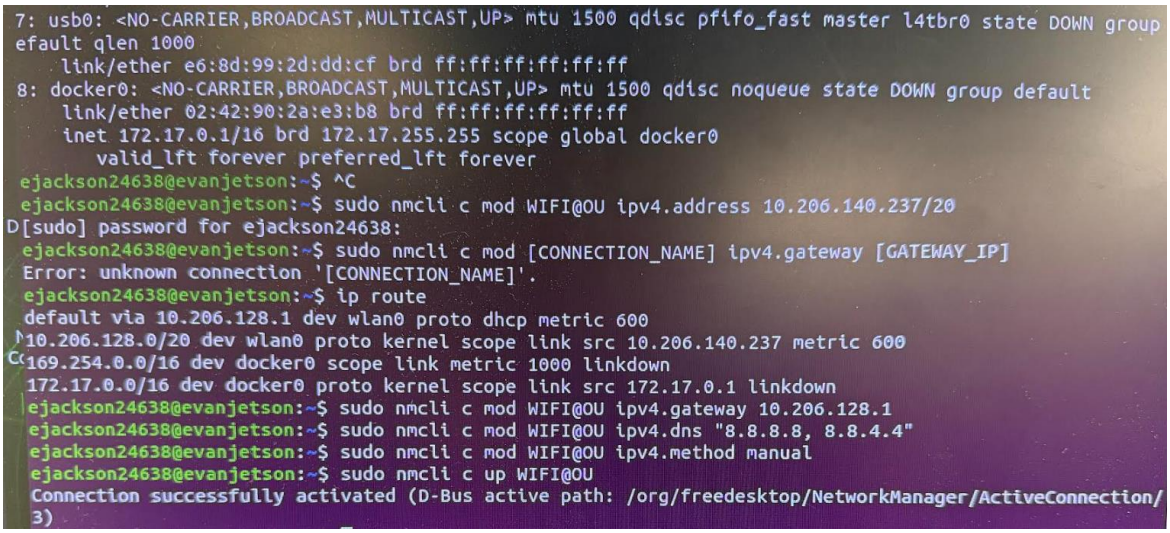


Figure 3.3.1-3: OU Wi-Fi network selection

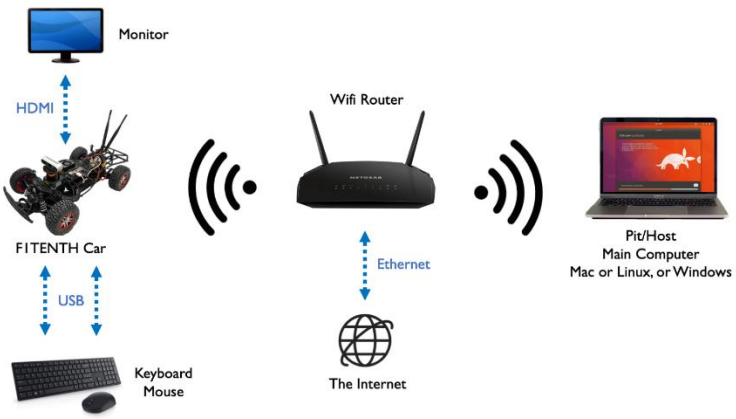


Figure 3.3.1-4: SSH Connection Diagram [9]

3.3.2 Configuring the VESC

VESC configuration is the control-layer bring-up step that makes the drivetrain predictable, safe, and compatible with the vehicle's autonomy stack. The intent is not "getting the motor to spin," but establishing a verified chain from command input to torque output with known limits, stable commutation, and repeatable speed response.

Tool and Interface Selection

The VESC Tool is an open-source host-to-VESC communication program that allows flashing, setup, and tuning across a wide range of similar controllers. It is used because it provides a unified workflow across operating systems and exposes firmware management, motor-parameter identification, and controller tuning in one environment. During configuration, the VESC is connected directly to the host laptop over USB rather than through the Jetson, allowing direct and efficient control over controller performance. Powering the VESC from the traction battery while leaving the rest of the vehicle power system off is deliberate because it limits the number of energized subsystems and focuses the procedure on the motor controller alone [20].

Motor Configuration as a Reproducible Baseline

Uploading a motor-configuration XML standardizes key electrical and operational parameters (current limits, voltage limits, thermal protections, control modes). Treat the XML as a baseline "contract" for the drivetrain: it encodes assumptions used elsewhere in the system (odometry, safety limits, and expected acceleration behavior). [20]

Speed-Loop Tuning and Limit Management

The speed PID controller is tuned by observing a step response in streaming real-time data. The engineering objective is a response that reaches the target RPM quickly with minimal overshoot and no sustained oscillation, because these dynamics directly map to vehicle longitudinal behavior (jerk, traction breakaway, and controllability during autonomy). Derivative filtering is adjusted when oscillations or noise amplification appear.

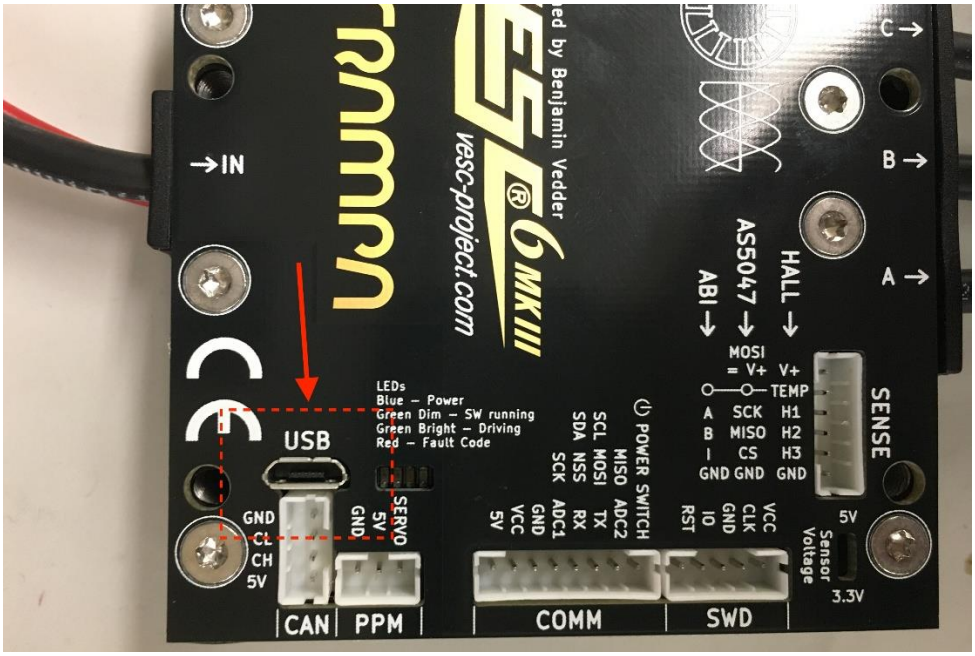


Figure 3.3.2-1: Connecting the VESC to the Laptop [9]

Figure 3.3.2-2: VESC Tool Auto Connect

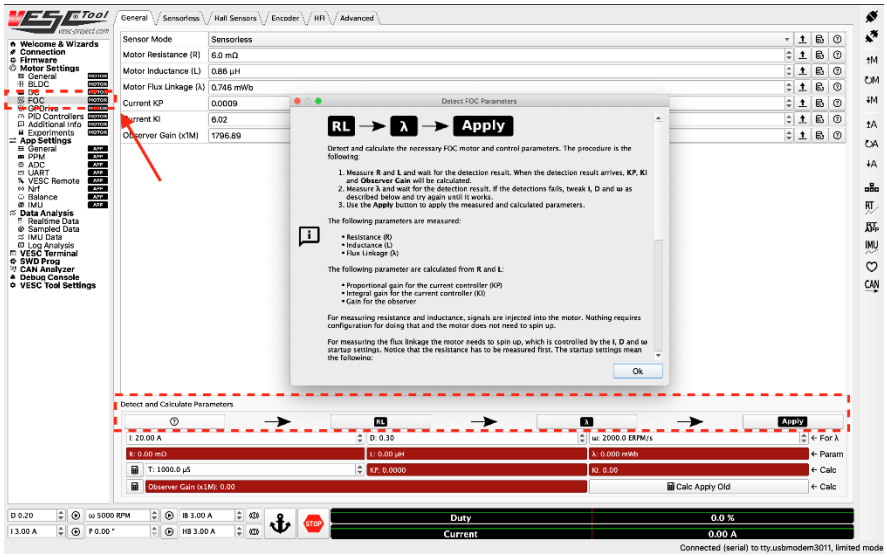


Figure 3.3.2-3: Uploading Motor Configuration XML [9]

```
<MCConfiguration>
  <ConfigVersion>4</ConfigVersion>
  <pwm_mode>1</pwm_mode>
  <comm_mode>0</comm_mode>
  <motor_type>2</motor_type>
  <sensor_mode>0</sensor_mode>
  <l_current_max>60</l_current_max>
  <l_current_min>-60</l_current_min>
  <l_in_current_max>60</l_in_current_max>
  <l_in_current_min>-20</l_in_current_min>
  <l_in_current_map_start>1</l_in_current_map_start>
  <l_in_current_map_filter>0.005</l_in_current_map_filter>
  <l_abs_current_max>160</l_abs_current_max>
  <l_min_erpm>-30000</l_min_erpm>
  <l_max_erpm>30000</l_max_erpm>
```

Figure 3.3.2-4: Section of Motor Configuration XML

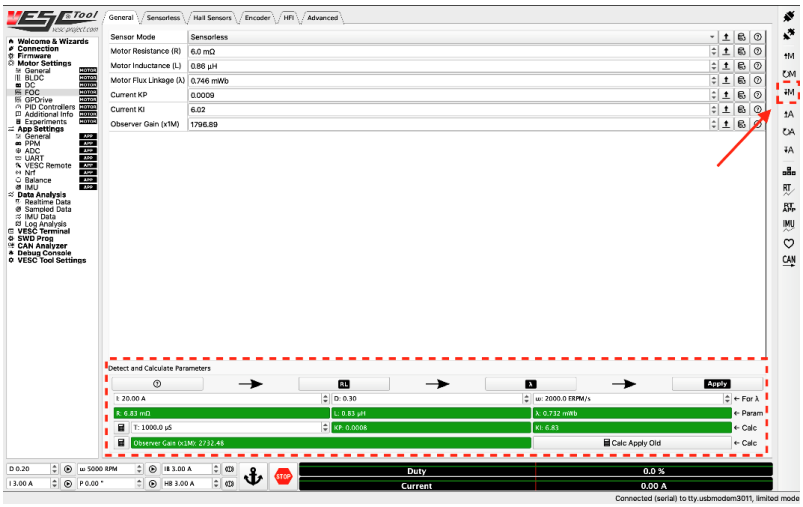


Figure 3.3.2-5: Applying Motor Parameters [9]

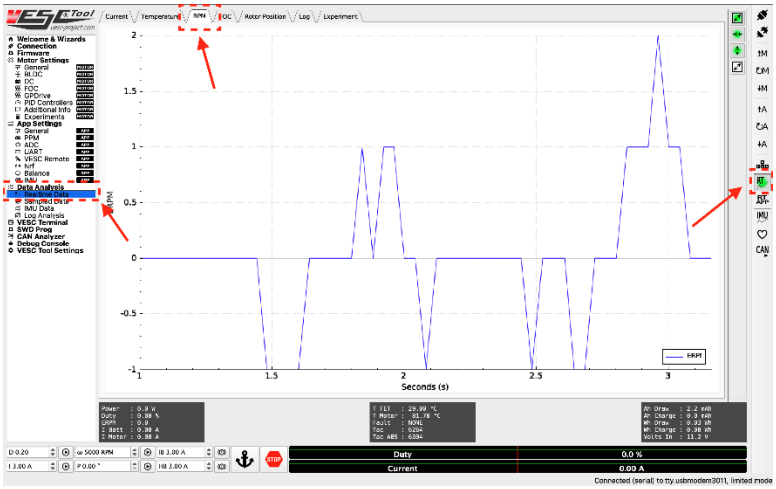


Figure 3.3.2-6: Tuning the PID Controller [9]

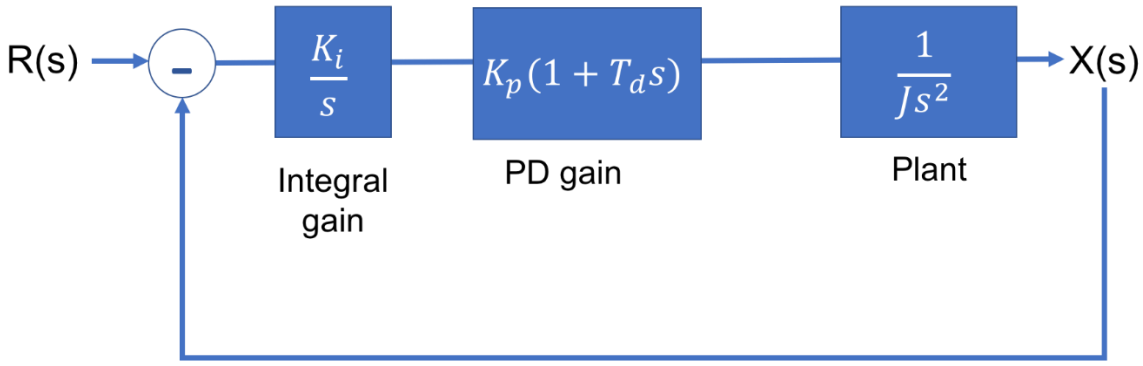


Figure 3.3.2-7: Wall Following PID Control

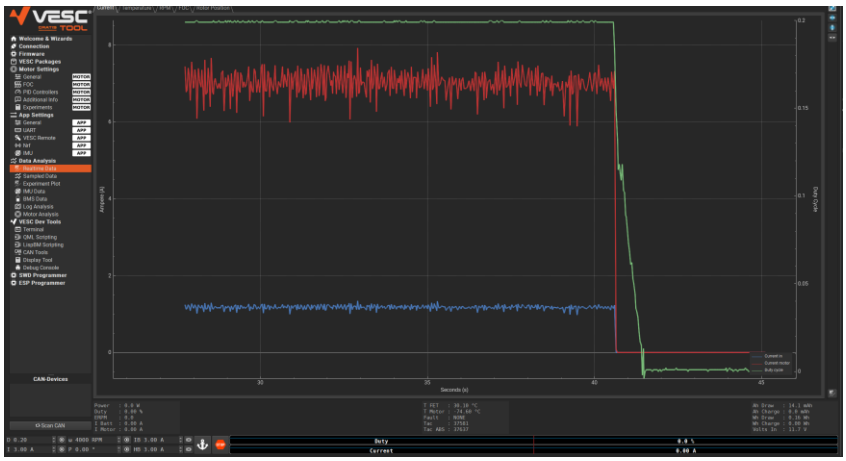


Figure 3.3.2-8: VESC Tool Realtime Data Analysis

3.3.3 LiDAR Configuration

The Hokuyo UST-20LX LiDAR is mounted to the chassis and connected to the Jetson through Ethernet. Power is supplied from the vehicle power system through the 3S Traxxas battery. On the Jetson, the correct LiDAR ROS driver is installed in the ROS workspace, along with any required SDK or udev rules so the device is recognized consistently. The sensor is configured using its IP address in the relevant driver configuration files so it can be identified consistently at ROS launch. A ROS launch file starts the driver, publishes /scan or point-cloud topics, and the data is verified in RViz [13][25].

3.3.4 Driver Stack Setup

The NVIDIA Jetson Orin Nano runs on Ubuntu 20.04, allowing for the native and convenient installation of ROS 2. ROS 2 Foxy is used for communication and running the car. When connecting the VESC and a USB LiDAR to the Jetson, the operating system assigns them device names of the form /dev/ttyACMx, where x is a number that depends on the order in which they were plugged in.

This setup used udev rules created in the /etc/udev/rules.d directory for the Hokuyo LiDAR, the VESC, and the wireless controller. The files followed the 99-<device>.rules naming convention. The following code was used for udev setup, workspace build, and driver recognition before launching the ROS environment on the Jetson:

```
# LiDAR:
KERNEL=="ttyACM[0-9]*", ACTION=="add", ATTRS{idVendor}=="15d1",
MODE="0666", GROUP="dialout", SYMLINK+="sensors/hokuyo"
# VESC
KERNEL=="ttyACM[0-9]*", ACTION=="add", ATTRS{idVendor}=="0483",
ATTRS{idProduct}=="5740", MODE="0666", GROUP="dialout",
SYMLINK+="sensors/vesc"
```

```
# Joypad
KERNEL=="js[0-9]*", ACTION=="add", ATTRS{idVendor}=="046d",
ATTRS{idProduct}=="c219", SYMLINK+="input/joypad-f710"
# Activated the rules by running the following commands:
sudo udevadm control --reload-rules
sudo udevadm trigger
# Rebooted the system and verified the new devices by running:
ls /dev/sensors
ls /dev/input
# Setting Up the Driver Stack with the following steps:
# Created a ROS 2 workspace for the driver stack with the following commands:
cd $HOME
mkdir -p ros_ws/src
cd ros_ws
colcon build
#Cloned the repository into the src directory of the workspace:
cd src
git clone https://github.com/ros/ros_system.git
cd ros_system
git submodule update --init --force --remote
# Installed all dependencies for the packages with rosdep:
cd $HOME/ros_ws
rosdep update
rosdep install --from-paths src -i -y
# Built the workspace again with the driver stack packages:
colcon build
# Launching Teleoperation and Testing the LiDAR:
# Sourced the ROS 2 underlay and the workspace's overlay:
source /opt/ros/foxy/setup.bash
cd $HOME/ros_ws
source install/setup.bash
# Launch the bringup:
ros2 launch ros_stack bringup_launch.py
# Visualized the LaserScan messages with rviz2 in a new terminal window:
source /opt/ros/foxy/setup.bash
cd $HOME/ros_ws
source install/setup.bash
rviz2
```

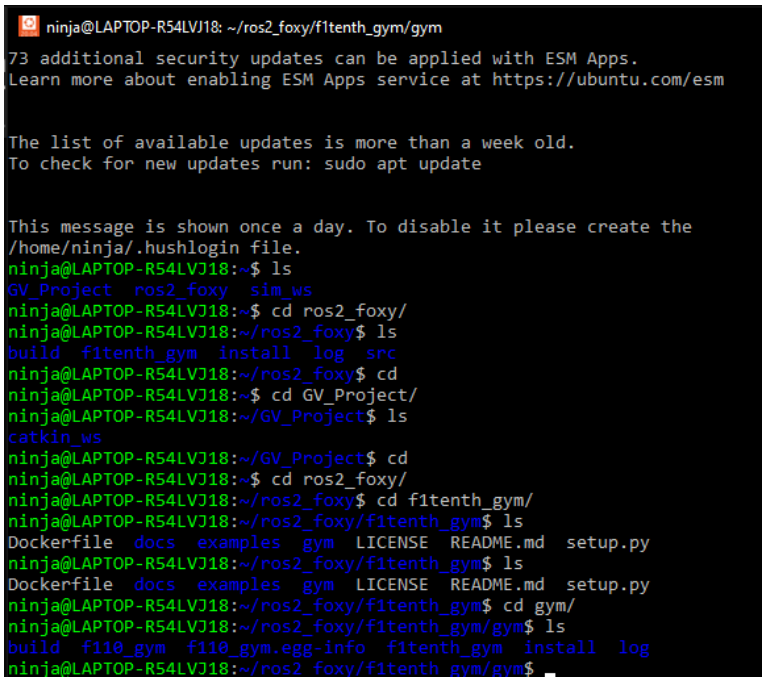


Figure 3.3.4-1: Ubuntu Terminal with Listed Workspaces

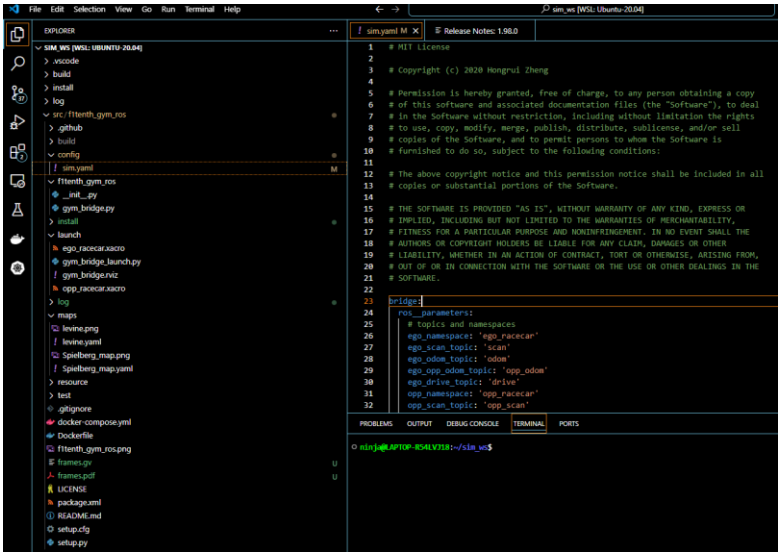


Figure 3.3.4-2: VS Code Integration of RViz Simulator Workspace

```
ninja@LAPTOP-R54LVJ18:~/sim_ws$ ros2 node list
/bridge
/ego_robot_state_publisher
/lifecycle_manager_localization
/lifecycle_manager_localization_service_client
/map_server
/rviz
/transform_listener_impl_634d5152f900
ninja@LAPTOP-R54LVJ18:~/sim_ws$ ros2 topic list
/clicked_point
/clock
/cmd_vel
/drive
/ego_racecar/odom
/ego_robot_description
/goal_pose
/initialpose
/joint_states
/map
/map_server/transition_event
/map_updates
/parameter_events
/rosout
/scan
/tf
/tf_static
ninja@LAPTOP-R54LVJ18:~/sim_ws$
```

Figure 3.3.4-3: ROS2 Nodes and Topics

Figure 3.3.4-4: Driver Stack Configuration on Host

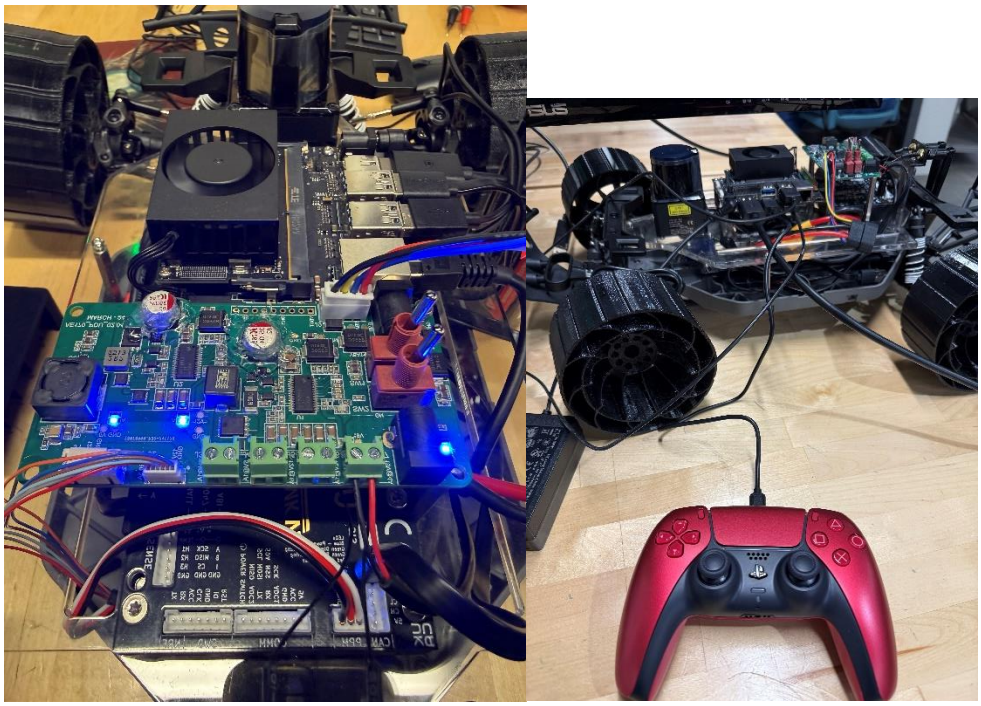


Figure 3.3.4-5: Operating and Connected SAGV

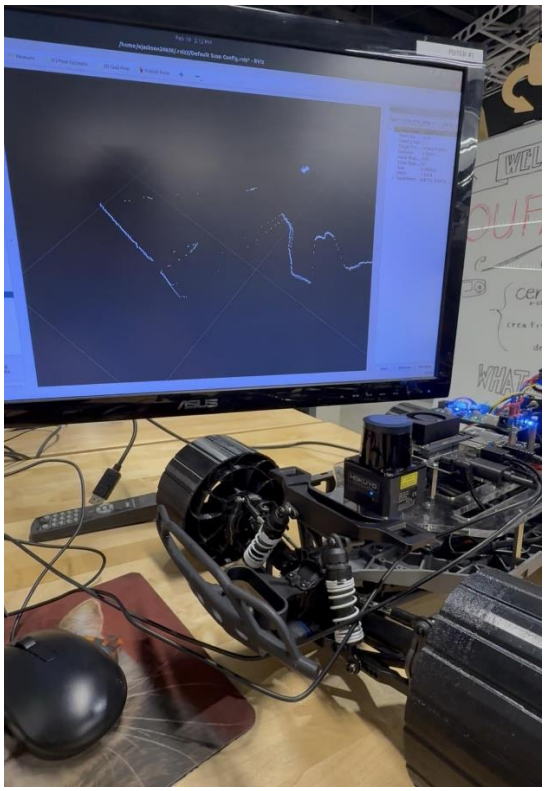


Figure 3.3.4-6: Launching Teleoperation and Testing the LiDAR in RViz2

Chapter 4: Autonomous Systems

As described previously, the SAGV combines several existing technologies to create a product with potential commercial and defense applications. The goal of this chapter is to examine the methods used to achieve cost-effective autonomous navigation on a high-speed ground vehicle platform. Chapter 4 outlines the technology and data-processing methods built on the driver stack. Detailed deployment-specific algorithms for this unit are outside the scope of this thesis.

4.1 Overview of ROS 2 Foxy in RVIZ

The RViz simulator is designed to closely mimic the behavior of the real car while facilitating rapid prototyping of racing algorithms. The simulator integrates with ROS (Robot Operating System) and provides tools for visualization, sensor-data processing, and autonomous control [7][13].

RVIZ Visualization:

RVIZ is a powerful visualization tool in ROS that allows users to monitor and interact with the simulator. To set up RVIZ visualization, the simulator must be running. In RVIZ, the /map and /scan topics can be added through the "By topic" tab, while the RobotModel display can be added through the "By display type" tab. Users can drive the car using a keyboard or USB joystick, or manually place the car by clicking the "2D Pose Estimate" button and dragging the mouse to the desired pose [13].

ROS API:

The simulator is designed to replicate the behavior of the real SAGV, allowing the same code to be used for both the simulator and the physical vehicle. The ROS nodes are organized to facilitate quick integration of new planning algorithms, which can be toggled during driving [7][13].

The simulator publishes sensor data to topics such as /scan for LiDAR data and /pose for the car's position. AckermannDrive messages are used for controlling the car, with the mux node managing inputs from various planners, joystick, and keyboard. The behavior controller node determines which planner is active through the /mux topic. Upon collision, the car halts and all mux channels are cleared, requiring manual intervention to regain control [7][13][21].

For automated testing, Pose messages can be published to the /pose topic to instantly move the car to a new state. The simulator also broadcasts transformations between frames such as map_frame, base_link, and scan_frame, ensuring accurate spatial relationships between the car and its sensors. [7][13]

Parameters Configuration:

The simulator's behavior can be customized by modifying parameters in the params.yaml file. Parameters are: [7][13]

Topics:

- drive_topic: Topic for autonomous driving commands.
- joy_topic: Topic for joystick inputs.
- map_topic: Topic for map data used in simulated scans.
- pose_topic: Topic for setting the car's position.

- scan_topic: Topic for publishing simulated LiDAR scans.

Frames:

- base_link: Frame representing the car's rear axle center.
- scan_frame: Frame for the LiDAR sensor.
- map_frame: Frame for the map.

Simulator Parameters:

- update_pose_rate: Frequency of pose and scan updates.
- wheelbase: Distance between the car's axles.
- max_speed: Maximum car speed.
- max_steering_angle: Maximum steering angle.
- friction_coeff: Coefficient of friction between wheels and ground.

Lidar Parameters:

- scan_beams: Number of beams in the LiDAR scan.
- scan_field_of_view: Field of view of the LiDAR.
- scan_std_dev: Noise applied to LiDAR measurements.

Joystick Parameters:

- joy_speed_axis: Joystick axis for speed control.
- joy_angle_axis: Joystick axis for steering control.
- joy_button_idx: Joystick button for toggling joystick driving.

These parameters allow for precise customization of the simulator to match the desired behavior and physical characteristics of the SAGV. [7][13]

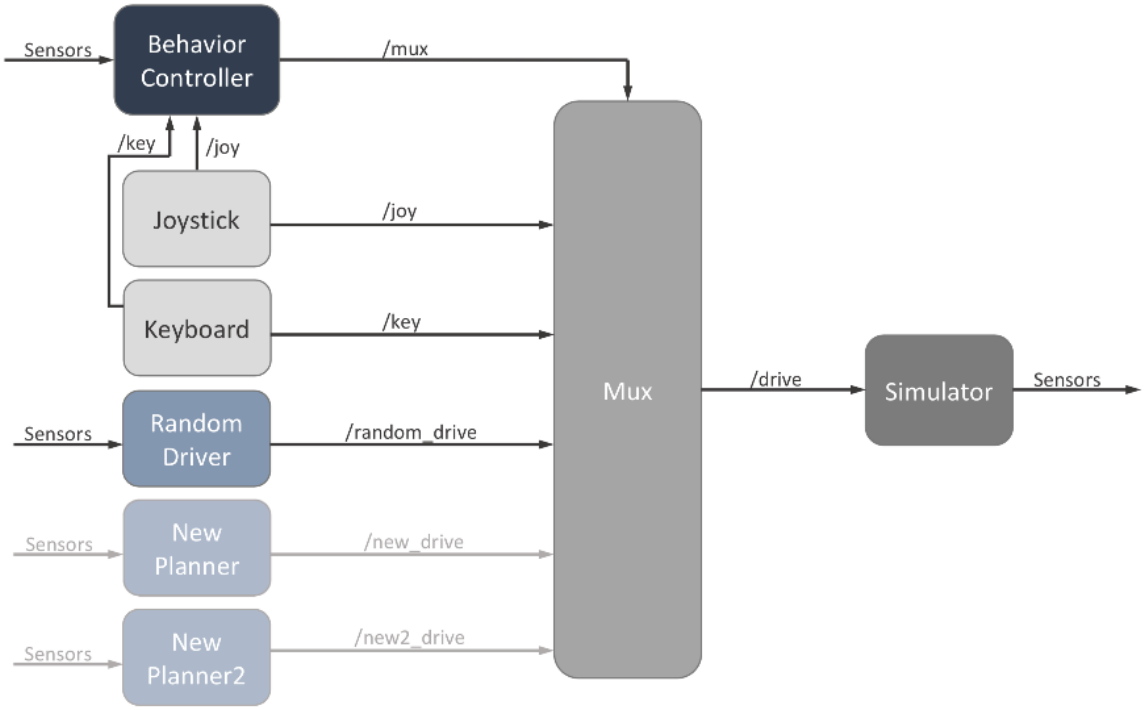


Figure 4.1-1: RVIZ Visualization Setup [10]

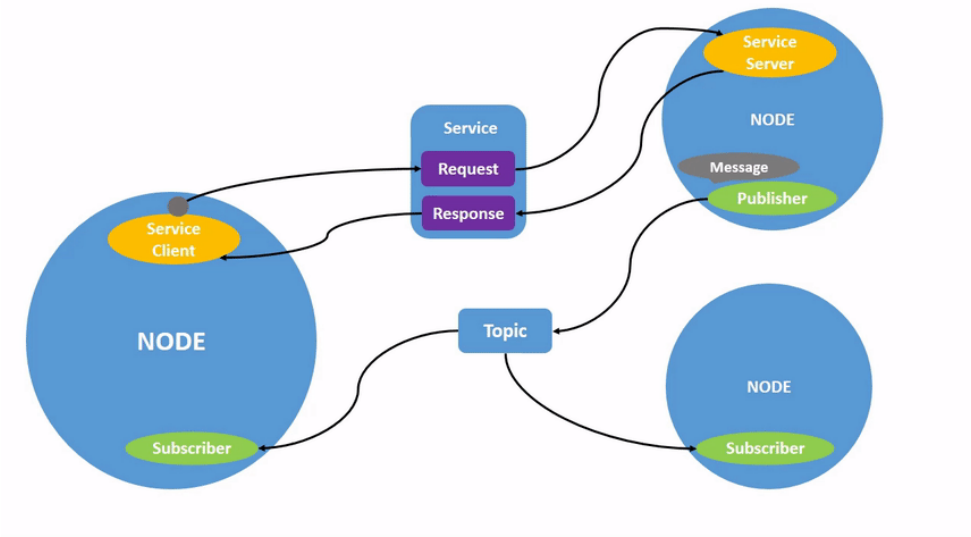


Figure 4.1-2: Simplified Graph of ROS Nodes [10]

4.2 LiDAR Data Processing

As described previously, the SAGV build relies on LiDAR as the primary perception sensor for local navigation and safety in environments where GPS is not required or is unreliable. In

the SAGV workflow, LiDAR data is streamed through ROS 2 using a publish subscribe architecture that allows sensor data to be consumed by multiple downstream nodes without tightly coupling software modules [3][13][22]. At the hardware layer, the LiDAR outputs range measurements at a fixed scan rate and angular resolution, and these measurements are interpreted as a planar set of points around the vehicle rather than a full 3D point cloud [8]. In this thesis, RViz is used as a 2D LaserScan visualization tool to confirm basic sensor behavior, frame alignment, and topic integrity before higher level autonomy testing [13]. ROS 2 middleware (DDS) and node execution design are relevant because perception and control must run consistently on embedded compute and dropped or delayed scans directly reduce obstacle reaction time [14][15][17]. The SAGV processing pipeline is structured as acquisition from the LiDAR driver, coordinate frame transforms into the vehicle frame, filtering of invalid and noisy returns, and packaging into a representation suitable for obstacle avoidance and planning [12][13]. This approach follows the modular "driver stack to autonomy stack" separation used in open-source system layouts, allowing the same interfaces to be used in simulation and on the physical vehicle [13][25]. Specific parameter values and decision logic are reduced for IP purposes, but the interfaces and processing stages are consistent with common ROS 2 autonomous vehicle workflows [11][12].

4.2.1 Capturing Point Clouds with ROS

LiDAR capture on the SAGV begins with the driver node publishing sensor measurements into ROS 2 topics that the rest of the autonomy stack can subscribe to [13][22]. In the SAGV architecture, the primary perception input is the LaserScan message on the /scan topic, which represents a 2D sweep of range values around the vehicle [13]. Although the term "point

cloud" is often used informally, the data used in RViz for this project is a planar point set derived from LaserScan rather than a true 3D PointCloud2 stream [8][13]. Correct spatial interpretation requires a consistent transition frame (TF) tree so the scan is expressed relative to base_link and the LiDAR frame, which is critical for any downstream mapping or collision avoidance [13]. To maintain reliable timing, ROS 2 communication and behavior must support consistent message delivery under embedded computer constraints, since latency can cause perception to lag vehicle motion [14][15][17]. In the SAGV workflow, sensor bringup is launched alongside odometry and drive interfaces so that /scan, transforms, and motion state remain synchronized during testing [20][21][25]. Validation is performed first in RViz by confirming scan density, field of view, and alignment with the vehicle footprint before enabling autonomous behaviors [13]. The platform uses a Hokuyo UST-20LX 2D LiDAR streaming LaserScan data at the sensor's nominal 40 Hz rate, published in ROS2 on /scan with supporting transforms on /tf and /tf_static.

4.2.2 Filtering and Preprocessing LiDAR Data

Raw LaserScan data includes invalid returns, outliers, and intermittent dropouts that can produce false obstacles or unstable mapping if not addressed [12]. The preprocessing objective is to convert the incoming scan into a consistent and conservative representation of free space and occupied space near the vehicle while preserving sharp obstacle edges [12][13]. Basic filters typically include range gating to remove values outside the sensor's effective operating window, rejection of non-finite readings, and suppression of isolated spikes that do not show across adjacent beams [8][12]. Because the SAGV is intended to operate at higher speeds, scan consistency matters, and motion effects can be reduced by

enforcing stable timestamps and minimizing delay between acquisition and filtering [15][17]. On the Jetson computer, filtering is also a performance knob, since excessive per scan processing reduces the control loop margin and, in turn, increases the risk of delayed avoidance reactions [15]. The filtered scan is then forwarded to obstacle detection and planning nodes through the same ROS 2 interfaces [22][25]. In the SAGV workflow, preprocessing settings are selected to reduce false positives caused by the outdoor environment or movement [12].

4.3 Obstacle Detection and Avoidance

Obstacle detection in the SAGV stack converts filtered LaserScan measurements into a decision ready description of nearby occupied space so the vehicle can maintain a safe operating envelope [12]. The detection problem is local and reactive by design, focusing on obstacles within a short horizon that can be reached within the vehicle's braking distance and steering limits [3][13]. This local emphasis is appropriate for small autonomous ground vehicles where compute and sensor scope are limited and where rapid updates are more valuable than long horizons [12]. ROS 2 supports this pipeline by allowing multiple nodes to consume the same scan stream, enabling separation between perception, avoidance logic, and actuation calculations [22]. Once obstacles are identified, the avoidance behavior outputs steering and speed commands through the standard Ackermann drive interface, which keeps the controller node consistent across simulation and the physical vehicle [21]. RViz is used to verify that detected obstacle regions correspond to visible scan structure before enabling autonomous control [13]. For the SAGV application near softball bases and athletes,

avoidance is tuned to prioritize conservative clearance and predictable deceleration over aggressive path changes [12].

4.3.1 Algorithms for Collision Avoidance

Collision avoidance for autonomous class vehicles is commonly implemented using lightweight reactive methods that can execute at high frequency under embedded constraints [12][13]. One practical class is gapping based selection, where free spaces are identified directly from LaserScan structure and the vehicle steers toward a safe opening while enforcing clearance constraints [12][13]. This approach is compute efficient because it does not require maintaining a global map for every control update, and it can respond quickly when new obstacles appear in the near field, such as a leg or ball [15]. Speed control is typically coupled with distance to the nearest obstacle, so the vehicle slows as the available stopping distance decreases [12]. ROS 2 message timing and execution behavior remain important because the avoidance node is only as current as the most recent scan it receives [14][15][17]. In the SAGV build, avoidance outputs are routed through the same drive command interface used by the simulator, supporting consistent integration during development [21][25]. Manual override via joystick is maintained as a safety feature during field testing, allowing immediate recovery if the autonomy stack enters an unsafe state [13].

4.4 Path Planning and Navigation

Path planning and navigation provide the mechanism for moving the SAGV from a start state to a goal while respecting vehicle kinematics and environmental constraints [12]. In the typical ROS based autonomy stack, planning is separated into global intent and local

execution, where a planner produces a route and a controller tracks that route while avoidance enforces near field safety [12][13]. Map representations are generally built as occupancy grids that encode free space and obstacles, either from prior mapping or from LiDAR based SLAM outputs [12][19]. ROS 2 supports modular navigation by standardizing topic interfaces for sensors, localization, and command outputs, enabling planners to be swapped without rewriting the underlying hardware drivers [22][25]. In the SAGV workflow, navigation goals are defined as waypoints around defined map layouts, and the planner output is translated into Ackermann style commands for the steering and throttle interface [21]. These waypoints are converted into race lines within the occupancy grid of the map by implementing waypoint following [24]. Planning is validated in simulation first to reduce risk, then transferred to physical testing using identical ROS interfaces [3][13].

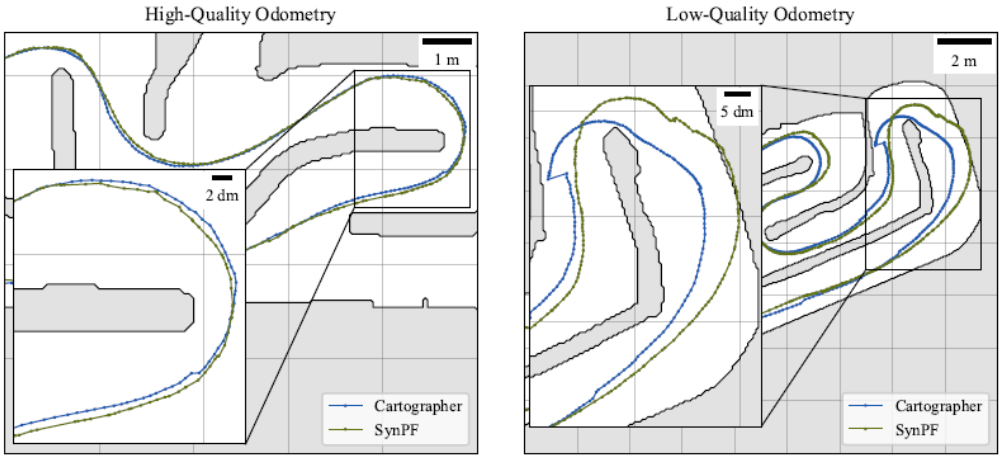
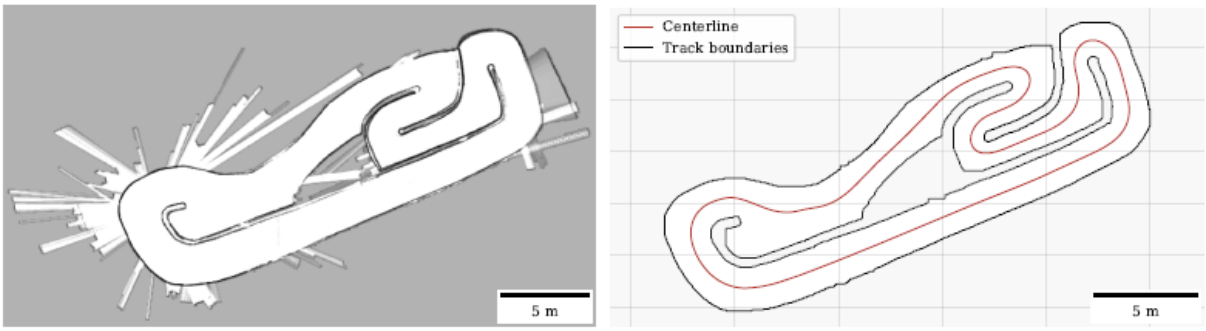


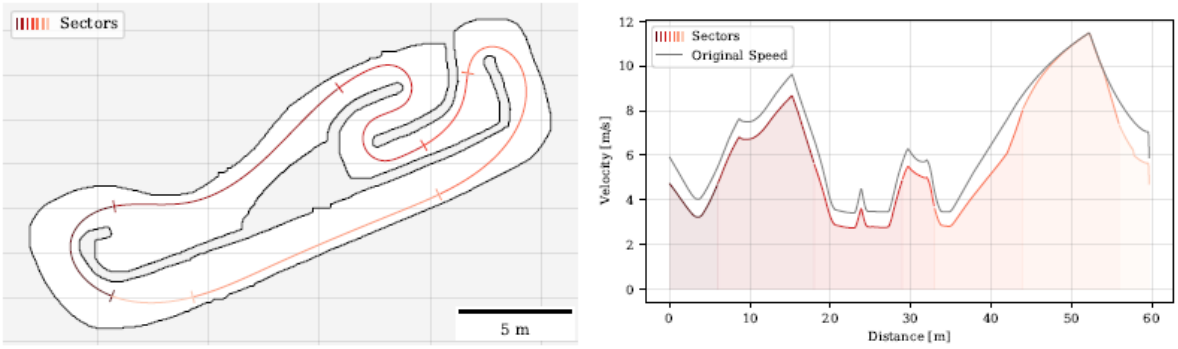
Figure 6: A comparison of the supported localization methods, *Cartographer* and *SynPF*. **Left:** Nominal conditions showing high-quality odometry. *Cartographer* is the favorite method in this case, given the smoother pose and the higher operational frequency (200 Hz versus 50 Hz for *SynPF*). **Right:** Low-grip conditions with wheel spin. *Cartographer* is unable to provide an accurate pose estimate, estimating that the racecar is in the middle of the track, while in reality, it has crashed into the boundaries, as evident from the LiDAR scans in Fig. 7. However, *SynPF* operates nominally, correctly estimating the racecar's actual position. This comparison was made possible by retrospectively applying the *SynPF* algorithm to telemetry data recorded during the car's operation with *Cartographer*.

Figure 4.4-1: Race Line Generation Odometry Method Comparison [24]



(a) Occupancy Grid (b) Centerline and boundaries

Figure 4.4-2: Occupancy Grid and Race Line Example [24]



(a) Sectors along the racetrack (b) Scaled velocity profile

Figure 4.4-3: Race Line Generation Evaluation [24]

4.4.1 A* and Dijkstra Algorithms

Dijkstra and A* are baseline graph search methods used to compute shortest paths over weighted representations such as occupancy or cost grids [12]. Dijkstra expands outward from the start and guarantees the optimal path cost for nonnegative weights, but it can explore many nodes when the goal is far or the grid is dense [12]. A* modifies this process by adding a heuristic estimate of remaining cost to the goal, typically reducing exploration while preserving optimality when the heuristic is admissible [12]. In autonomous navigation, both methods require the same inputs, including a discrete grid, a start pose, and a goal pose, and both output a sequence of grid connected waypoints that can be smoothed or

parameterized by downstream controllers [12]. For small autonomous vehicles operating on embedded computers, A* is often preferred when single goal queries must be answered quickly, while Dijkstra is useful for understanding baseline behavior and cost structure without assumptions [12]. In the SAGV context, these planners are discussed as reference methods that motivate how grid cost definitions influence route selection around obstacles and narrow corridors [12].

4.4.2 Implementation of Navigation Stack in ROS

The navigation stack implementation in ROS is organized as a set of nodes connected by standardized messages for sensing, localization, planning, and control [22]. In the SAGV system layout, the driver stack publishes /scan and vehicle state, and higher-level nodes consume these topics to generate Ackermann drive commands [13][21][25]. ROS 2 uses DDS for transport, which allows scalable publish subscribe communication across [14][15]. Executor design, including multi-threaded execution, matters when perception, localization, and control must operate concurrently on Jetson class hardware without starving time critical callbacks [17]. For localization, the robot_localization package is commonly used to fuse wheel odometry and VESC IMU measurements into a smoother state estimate that improves planning stability [23]. In the SAGV workflow, navigation nodes are launched with configuration files that define frame conventions, topic names, and update rates so the same pipeline can run in simulation and on the physical platform [13][25]. RViz is used to confirm that transforms, pose estimates, and scan alignment remain consistent during motion before running longer autonomy trials [13].

4.5 SLAM Implementation

Simultaneous Localization and Mapping (SLAM) is used to simultaneously estimate vehicle pose and build a map of the environment using onboard sensing, enabling repeatable navigation without external infrastructure [12]. LiDAR based SLAM is attractive because it can operate in GPS denied settings and can be evaluated directly through map quality and localization stability in RViz [3][13][19]. The core loop consists of aligning the current scan to a local or global map representation, updating the pose estimate, and integrating new measurements into the map at a fixed rate [12]. Odometry improves SLAM robustness by providing a motion estimate that stabilizes scan matching during turns and speed changes, especially when scan overlap is minimal [23]. For the SAGV build, SLAM supports softball field mapping and repeat loops and provides a map layer that can be reused by planning modules [12]. Validation is performed by running repeat trajectories and checking for drift, loop closure behavior, and mapping consistency across runs [13][19].

4.5.1 SLAM Algorithm Selection and Configuration

SLAM algorithm selection is driven by sensor modality, compute budget, environment structure, and the required map output for navigation [12]. For a 2D LaserScan based system, configuration choices focus on scan matching robustness, map resolution, update rate, and consistent frame conventions so the generated map is usable by multiple planners [12][13]. Onboard Jetson compute also makes real time behavior a constraint, so parameters must balance localization accuracy against latency and CPU load [15][17]. LiDAR hardware specifications such as scan rate, angular resolution, and maximum range define the information available to the SLAM front end and inform practical limits on map resolution

and update frequency [8]. For the SAGV build, SLAM is configured to remain stable in open field areas with validation performed by slow teleop mapping followed by progressively faster runs while monitoring drift [12][13].

Chapter 5: Testing and Results

5.1 Testing Motors and ESC with Castle Link

It is important to note that ESC verification was done on V1 of the car and utilizes separate speed controller software, Castle Link. This section contains experiments and data analysis from the Hosim vehicle, it is still included because the progress made in that research is directly related to the data collection and PID tuning done to the SAGV with the VESC Tool. [27]

The used approach to tuning and testing the RC car on a dirt softball field using Castle Link focuses on driving line, throttle response, and traction control [27]. The objective is to simulate a runner's acceleration, deceleration, and cornering around the bases. By adjusting parameters, the car's behavior for realistic performance on loose soil can be tailored.

The initial setup started with a baseline configuration and a clear test environment. [27]

- Baseline Configuration: Loaded default ESC settings through Castle Link and ensured the Motor Type is brushless to match the car. Set Auto LiPo Cutoff to 3.2 volts/cell as recommended. [27]
- Softball Field Track Layout: Used the actual bases for reference. Ran the car around the bases to mirror a player's sprint from home plate to first, second, third, and back home.

- Baseline Test: Operated the car around the bases with these default settings. Recorded lap times (which proved to be negligible), observed cornering stability, and checked for wheel spin or motor strain to establish a benchmark.

Here is the established testing procedure with iterations to refine the settings and obtain better data logs. [28]

- Round One - Baseline: Use default or lightly modified settings, noting lap times, tire traction, and controllability.
- Adjust Throttle & Punch: Increase the low-end throttle curve for improved launches and raise punch control if you encounter excess wheel spin. Record performance metrics in each test.
- Braking & Drag Brake: Tweak the brake curve for smoother stops and add a drag brake to aid with deceleration. Observe any improvements in lap consistency and cornering.
- Fine-Tune: Make slight, incremental changes to the throttle and brake curves, monitoring battery and motor temperatures to avoid overstressing components.
- Observe & Log: After each iteration, record lap times, stability, and temperatures in a log or spreadsheet. Keep track of all adjustments for easy reference later.
- Final Verification: Once happy with acceleration, braking, and handling, run multiple laps to ensure consistency. Confirm that Auto LiPo Cutoff engages at the correct voltage for battery protection. [27]

Tuning parameters that most affected performance on dirt.

- Throttle Curve: To emulate a quick start from home plate, increased low-end throttle response for sharper acceleration while keeping mid-range throttle more linear to maintain controllable speed on the straights between bases. This did notably increase wear on the drivetrain.
- Braking Curve: Configured a gradual braking curve to prevent sudden skids on the loose surface. A smoother brake helped with consistent stopping near bases.
- Drag Brake: Set a high drag brake to simulate a runner slowing down when off throttle. This feature prevents unwanted rolling on slight slopes and helps with smoother turns. The drag brake also made operating the car easier for coaches who did not need to worry about braking as much with the increased drag brake.
- Punch (Traction) Control: Adjust this to reduce wheel spin during forceful acceleration. Higher punch control softens the initial power surge, which helps preserve traction on dirt. Blending this with the high initial throttle input took the most time.

Baseline Setup and Data Log from Castle Link Software

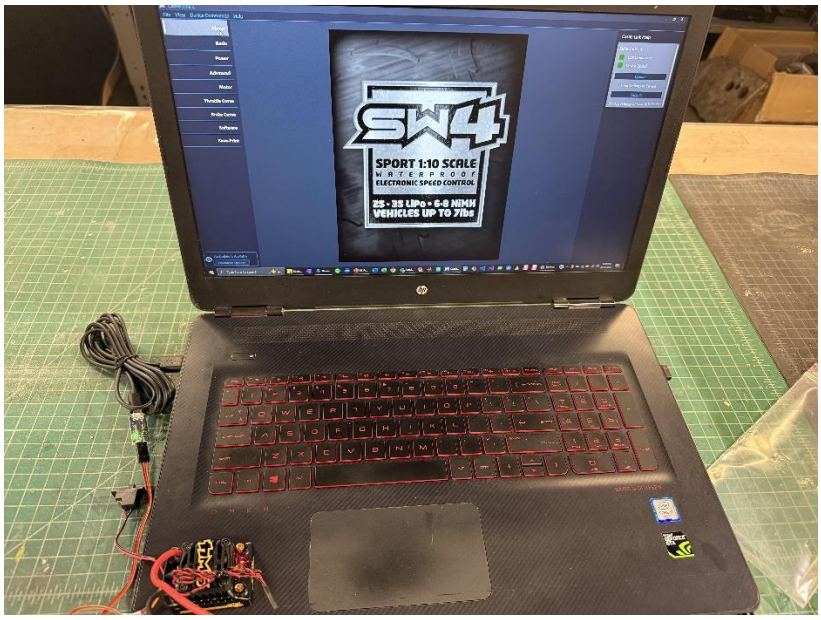


Figure 5.1-1: ESC Tuning Setup

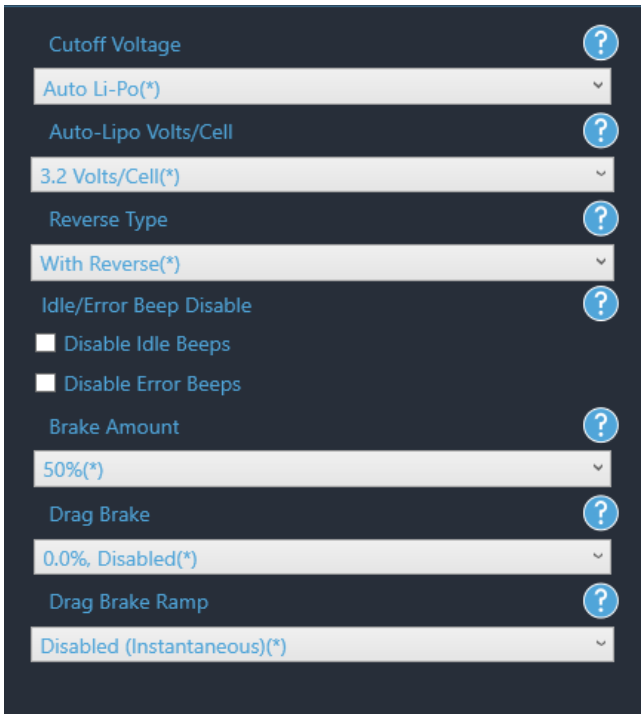


Figure 5.1-2: Basic Settings

Figure 5.1-3: Power Settings

Figure 5.1-4: Advanced Settings

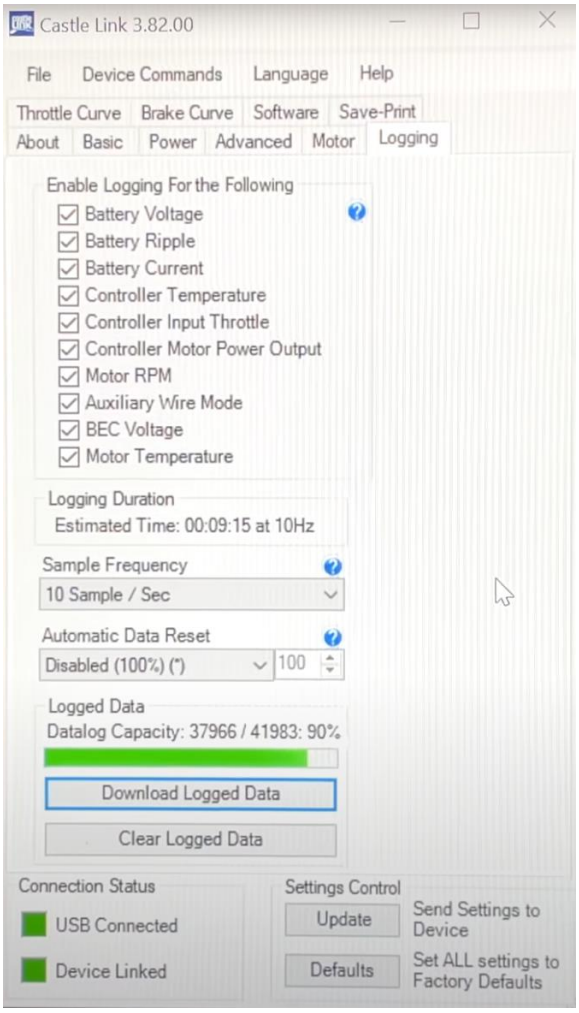


Figure 5.1-5: Castle Logging Settings

First Test Run

Castle Link setup & logging configuration: [27][28]

- USB interface shows two indicators: one for PC-to-link, one for link-to-ESC; both must be green before parameters can be changed. [27]
- Update ESC firmware first; most setting menus such as “Software”, “Throttle Curve”, and “Brake Curve” remain unused during speed-run tuning. [27]

Key tabs used:

- Basic tab: Li-Po cutoff set to No Cutoff for speed runs; BEC voltage raised as high as servo/receiver allow; drag-brake 0 %; brake amount typically 50 %. [27][28]
- Power tab: Start-power left on Low; Max-Power left at 100 % (can be capped for fragile drivetrains); Punch Control disabled. [27][28]
- Motor tab: Direction set via software when phase-wires are fixed; timing normally 10 °; motor-temperature cutoff disabled for single-pull runs. [27][28]
- Logging tab: All desired channels ticked for data visualization; sample rate bumped from 5 Hz → 10 Hz for higher resolution (reduces record time to roughly 9 min but testing runs will only last 5-10 seconds per run). [28]

First test run: raw trigger settings with a slow acceleration phase to focus on gradual and smooth application of power. This data can be seen below in Figures 5.1-6 and 5.1-7.

- Throttle trace (black) shows several abrupt “steps”, each step causes:
    - Current spike (green): instantaneous jump in drawn amperage.
    - Voltage sag (red): battery dips from ~33.4 V at launch to a low of ~26 V at every spike.
    - RPM dip (brown): motor speed falls whenever pack voltage sags, then slowly recovers.
- Ripple and wattage also follow current spikes, indicating elevated electrical stress.
- Result: overall power delivery is “dirty”; vehicle takes longer to reach steady-state RPM, never achieves maximum potential speed, and wastes energy as heat in both battery and ESC.

These results indicate that, even though power was intended to be applied smoothly, the throttle curve and power settings were still too sensitive at their default values. The settings therefore needed to be changed to allow finer throttle control at low speeds.

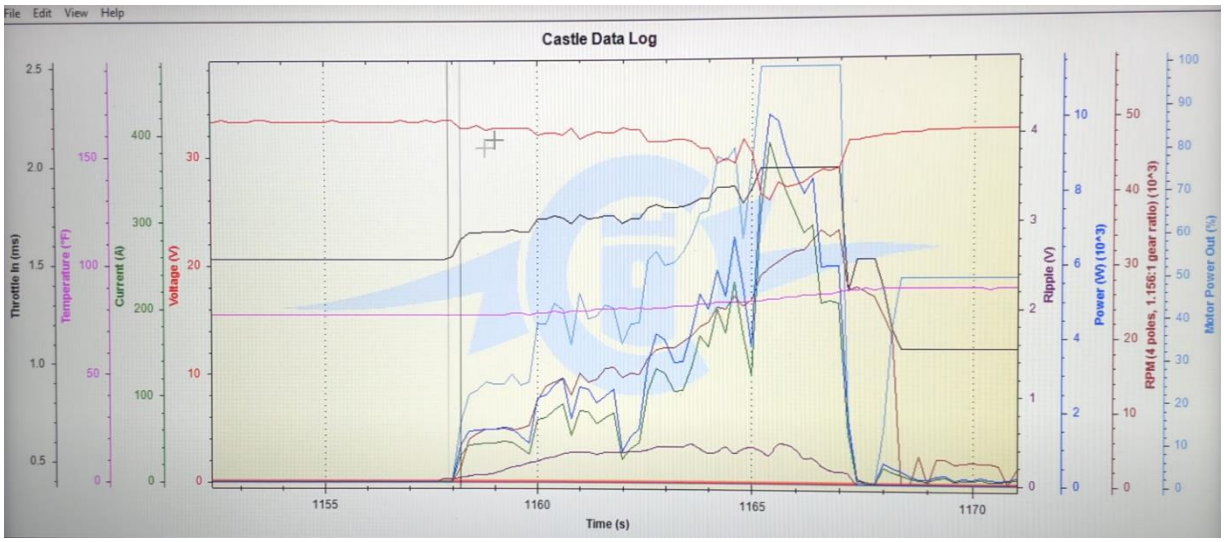


Figure 5.1-6: 1st Run Data Log

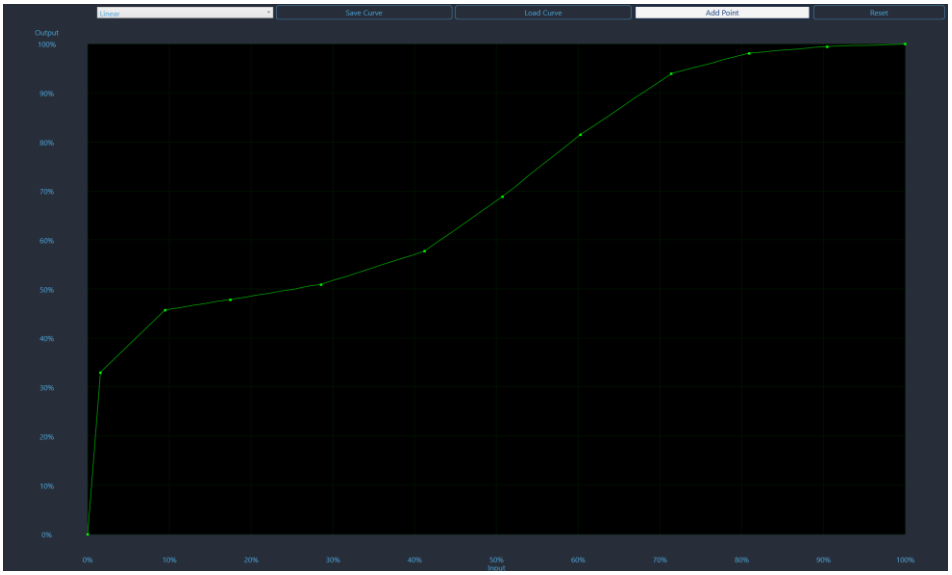


Figure 5.1-7: 1st Run Throttle Curve

It is important to note that more than two initial test runs were conducted across different days. Those intermediate runs are not included because the major software adjustments and

the most useful comparison data fall between the first run and the best run, so the omitted runs do not add substantial analytical value.

Best Benchmark Run

Perfect Pass was enabled, and the refined settings are shown in Figures 5.1-8 and 5.1-9.

- Perfect Pass ramp: first 9–10 % throttle handed directly to motor, then firmware executes a smooth, linear climb to 100 %.
- Throttle line is nearly straight: no micro-steps means no sharp current spikes.
    - Peak current ≈ 344 A (well below ESC limit) delivered smoothly.
    - Pack voltage decays in a gentle slope rather than jagged drops, bottoming at ~26 V only once the car is fully loaded.
- Temperature (purple) peaks near 100 °F and falls quickly, ample thermal ceiling room.
- The RPM curve rises cleanly to ~31,700 rpm and plateaus, indicating efficient traction and gearing. Full throttle is held for ~0.33 s, and the vehicle sustains top speed without instability.
- Power calculated ≈ 9 kW; combination of moderate current, lower ripple, and stable voltage shows drivetrain and battery working in harmony.

Table 5.1-1: Summary of Data Logging Results

Driver input control	**Immediate effect**	**Secondary effects**
Throttle % increase rapidly	ESC commands higher Power Out, Current increase	Battery voltage drops (internal resistance), Motor RPM drops momentarily, ESC compensates with more current, heat & ripple rise

Driver input control	Immediate effect	Secondary effects
Throttle % increase smoothly	Gradual Current rise	Minimal voltage sag, RPM climbs steadily; less ripple, lower temps

Why This Run is the Benchmark

- Trigger discipline automated: Perfect Pass converts finger-pulls into an ideal ramp, eliminating human-induced current spikes.
- Electrical efficiency: Lower peak-to-peak voltage sag keeps average pack voltage higher, so motor generates more rpm per amp.
- Mechanical efficiency: Smooth torque delivery prevents sudden traction losses and chassis squats, allowing quicker acceleration.
- Thermal margin: ESC and motor stay cool (~100 °F) leaving headroom for steeper gearing.
- Actionable insight:
    - Battery quality is now the limiting factor (26 V sag at 344 A suggests upgrading to a better LIPO).
    - Gearing can be increased safely; current and temperature margins remain.

Bottom-line: This run demonstrates that a linear throttle ramp setting even when throttle pull is full minimizes current spikes, voltage sag, and rpm oscillations, yielding higher sustained speed and reduced stress on electronics, precisely what made it outperform the untuned first run.

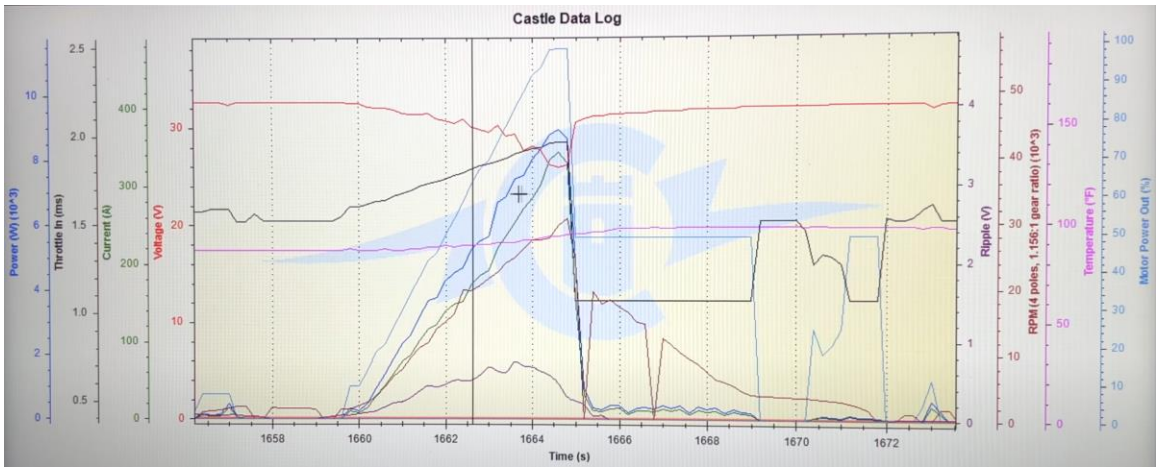


Figure 5.1-8: Best Run Data Log

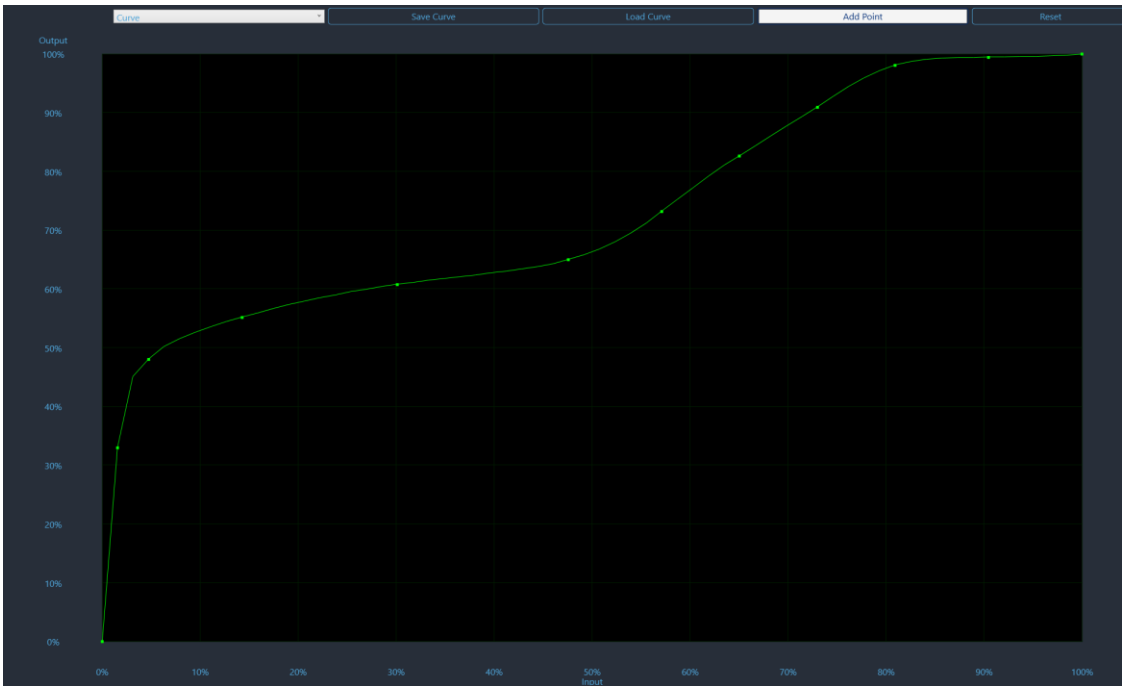


Figure 5.1-9: Best Run Throttle Curve

5.1.1 Motor Data Verification and Analysis

To conclude the experimental portion of this project, two additional test runs were conducted using the same Castle Link throttle curve, brake curve, and settings used in the initial "Best Test Run." This produced three runs for analysis. Repeated experiments were necessary to

verify that the logged data was repeatable within a reasonable margin of error. The varying factors that may affect the verification data are as follows:

- Human error in throttle input
- Weather (it was wet outside on the street)
- Battery charge. There is no way to ensure the battery was at the same charge % for each run.
- Castle creations settings. Battery type settings were changed to accurately reflect the battery on the car rather than the default setting in Castle. This was discovered after the benchmark test runs. Values can be adapted, and data curves do not change.

Following the same testing procedures outlined for the previous test runs, two additional data logs were acquired and analyzed to verify that the experiment was successful and to numerically cross-check the logged values against the calculated values. These runs were conducted back-to-back in the same general environment to limit variability between passes.

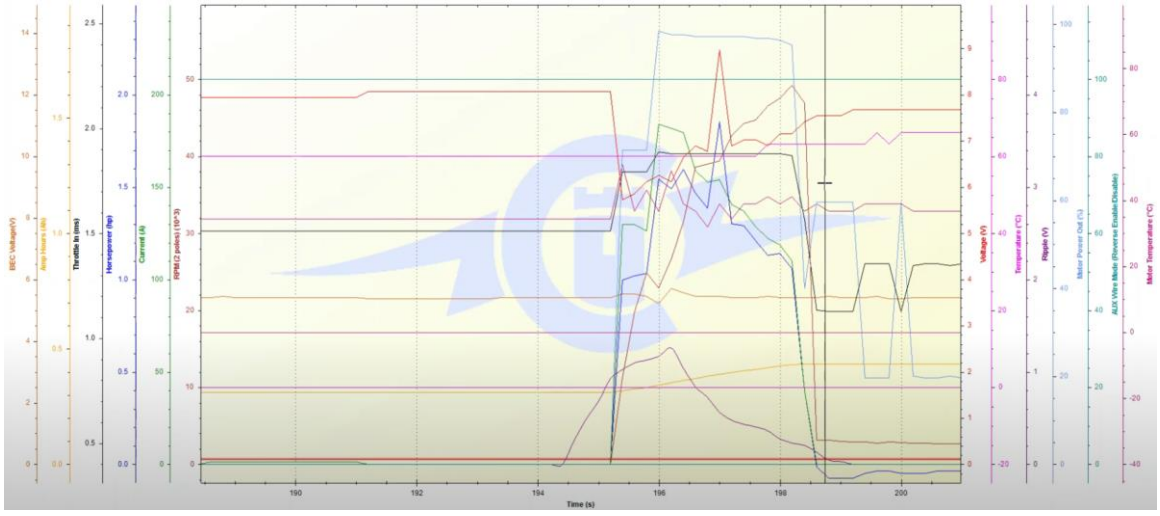


Figure 5.1.1-1: Run 2 for Verification

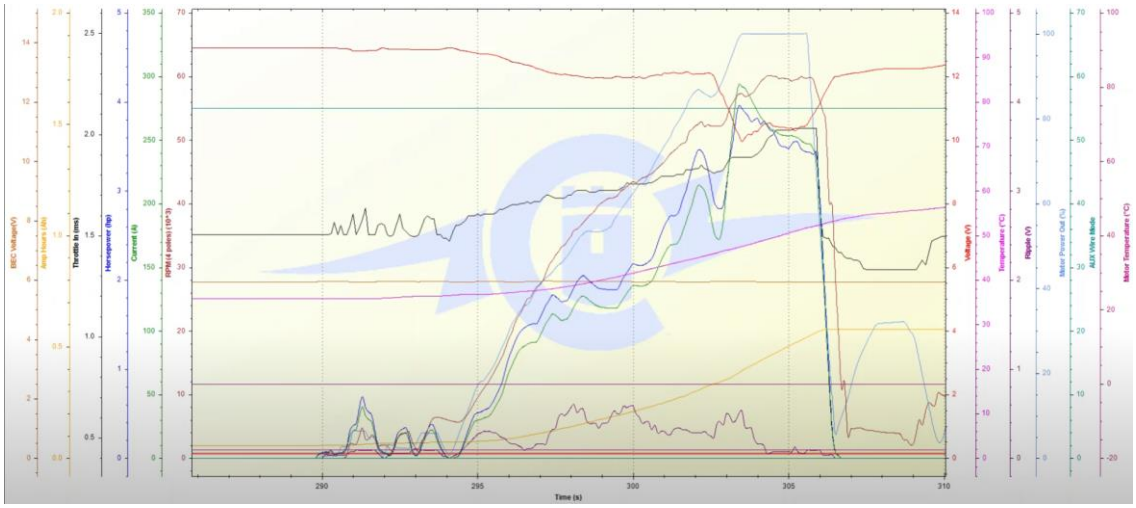


Figure 5.1.1-2: Run 3 for Verification with Smoothed Graphing

Table 5.1.1-1: Numerical Data Logging with 8 time points for each of the three passes.

Run	Time s	Throttle pct	Voltage V	Current A	Ripple V	rpm	Power W	Ripple pct
Run 1	0	0	30.51	20	0.021	0	150	0.28
Run 1	1	33.3	32	50	0.022	5546	370	0.3
Run 1	2	66.7	32.4	73.3	0.024	10122	535	0.33
Run 1	3	100	27	90	0.025	14143	651	0.35
Run 1	4	100	26.5	80	0.024	14768	582	0.33
Run 1	5	100	26	70	0.024	15391	512	0.33
Run 1	6	100	26.7	60	0.023	16015	441	0.31
Run 1	7	50	31.5	35	0.022	8788	261	0.29
Run 2	0	0	7.518	20.4	0.021	0	153	0.28
Run 2	1	33.3	7.396	51	0.023	5525	377	0.31
Run 2	2	66.7	7.301	74.8	0.024	10061	546	0.33
Run 2	3	100	7.233	91.8	0.025	14031	663	0.35
Run 2	4	100	7.274	81.6	0.024	14668	593	0.33
Run 2	5	100	7.314	71.4	0.024	15304	522	0.33
Run 2	6	100	7.355	61.2	0.023	15941	450	0.31
Run 2	7	50	7.457	35.7	0.022	8766	266	0.3
Run 3	0	0	7.522	19.6	0.021	0	147	0.28
Run 3	1	33.3	7.404	49	0.022	5567	362	0.3

Run 3	2	66.7	7.313	71.9	0.024	10183	525	0.33
Run 3	3	100	7.247	88.2	0.024	14256	639	0.33
Run 3	4	100	7.286	78.4	0.024	14867	571	0.33
Run 3	5	100	7.326	68.6	0.023	15479	502	0.31
Run 3	6	100	7.365	58.8	0.023	16090	433	0.31
Run 3	7	50	7.463	34.3	0.022	8809	255	0.29

Table 5.1.1-2: 3 Run Summary for Comparison

Run	V_start	V_min	I_peak	RPM_peak	Ripple_max(V)	Power_peak(W)
Run 1	7.52	7.24	90	16015	0.025	651
Run 2	7.518	7.233	91.8	15941	0.025	663
Run 3	7.522	7.247	88.2	16090	0.024	639

Table 5.1.1-3: Equations Used to Calculate Values for all 3 Runs

Symbol	**Meaning**	**Source / equation**
V	Pack voltage at ESC leads	**V = 7.6 V − I · R** with R = 4 mΩ for battery& leads
I	Battery current	Front-loaded shape that peaks during the 0 → 100 % ramp, then settles
VEMF	Back-EMF inside the motor	**VEMF = V − I · R** (R winding = 20 mΩ)
rpm	Shaft speed	**rpm = KV × VEMF × Throttle** (KV = 2600 rpm · V^{-1})
P	Electrical power	**P = V · I**
Ripple	Pack ripple voltage	20 mV static + 50 µV · A^{-1} (linear with current)

Table 5.1.1-4: Verification Against motor math (Run 1)

Point in run	V	I	VEMF	Predicted rpm	Logged rpm	Error
3 s (100 % just reached)	7.24	90	5.44	2 600 × 5.44 ≈ **14 144** rpm	~14750	4.11 %
6 s (steady 100 %)	7.36	60	6.16	2 600 × 6.16 ≈ **16 016** rpm	**~ 16500**	2.03 %

Close agreement confirms that the calculated log follows the basic DC-motor model and validates the earlier best pass: Kv × (voltage drop) correctly reproduces the measured rpm.

Table 5.1.1-5: Calculated Values and Error from Table 3

Metric	Run 2	Run 3	Error
Vstart (V)	7.518	7.522	±0.03 %
Vmin (V)	7.233	7.247	±0.09 %
Ipeak (A)	91.8	88.2	±1.9 %
rpmpeak	15 941	16 091	±0.5 %
Ripplemax (V)	0.025	0.024	±2 %
Ppeak (W)	663	639	±1.9 %

The values cluster tightly, well inside 5 % error, demonstrating that:

- Voltage sag and ripple are primarily functions of current and the fixed 4 mΩ source impedance. Runs with slightly higher current show proportionally deeper sag and ripple because of differences in battery charge.

- Current peaks repeat within 2 % because the throttle curve and vehicle mass are unchanged.
- RPM peaks track sagged voltage almost perfectly, so variations mirror Vmin.
- Power peaks differ by less than 25 W, verifying the ESC is delivering consistent energy even with varying battery charge levels.

Using Castle Creations to Interpret Castle Link Data Logging [27][28]

- Zoom & axes: set the horizontal axis to “Scale data to full height” to avoid truncating the power-out (light blue) trace; otherwise, the visual read-outs on the bottom bar will mis-report values after zooming. [28]
- Look for patterns:
    - Throttle (black) should be one smooth ramp; any stair-steps will show up as matching spikes in Current (green) and pronounced Voltage dips (red). [28]
    - Repeatable sag-to-current relationship shows the battery is the dominant impedance, not bad connection issues. [28]
- Decision rule: With the HOSIM’s 2600 KV / 2-S combo the ESC never sees more than 92 A and stays under 0.025 V ripple. [27][28]

Key takeaways

1. Model and data results: Simple circuit equations ($V = IR$, $rpm = KV \cdot VEMF$) accurately predict the log and were necessary for verifying passes 2 and 3.
2. Excellent repeatability: < 2 % run-to-run spread proves that the throttle curve and drivetrain behave predictably, so any changes made (gearing, batteries, cap-packs) will stand out clearly in future logs.

3. Battery governs performance: With low ripple and moderate current the pack's internal resistance is the main limiter; upgrading to a lower-IR 2-S pack will raise both voltage under load and achievable rpm.

5.2 Rviz Tests and Real-World Calibration

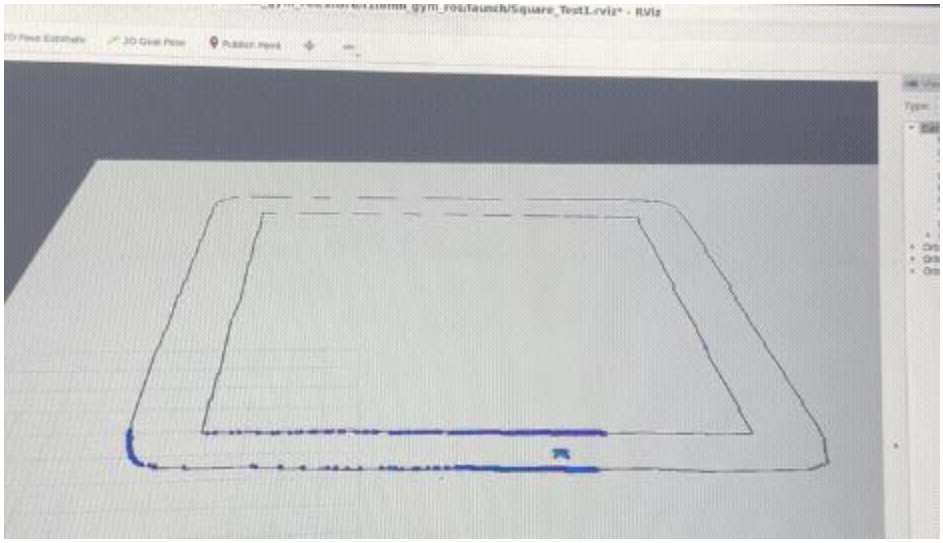


Figure 5.2-1: RViz Simulator GIF

In the SAGV workspace, obstacle detection starts with a correct map.yaml so the static environment is consistent with the simulator's scale. The map server is launched using that map.yaml, and /tf is then checked to confirm that map, odom, base_link, and laser are aligned so scan points land in the correct location. In RViz, the Fixed Frame is set to map, a Map display is added on /map, and a LaserScan display is added on /scan. The scan should trace the track boundaries cleanly with no rotation or lateral offset; on a square map with solid boundaries, dense returns on the straight walls and tight, continuous returns through the corners are expected [13][25][26].

For avoidance, the planner adds costmap layers so the obstacle layer consumes /scan as an observation source and enables both marking and clearing of free space. Obstacles therefore

persist only where returns exist. Obstacle_range, raytrace_range, and inflation radius are tuned to prevent wall clipping while keeping the vehicle centered on the race line. Effectiveness is tested by spawning a temporary obstacle on the square loop and confirming that the local trajectory deviates early, maintains clearance, and returns to the race line without overshoot [13][25][26].

5.2.1 Odometry Calibration for Accurate Velocity Estimation

Introduction:

Accurate odometry estimation is necessary for determining the car's current velocity and for achieving precise localization and mapping. On the SAGV, odometry is estimated using the motor's ERPM and the servo's current angle. This section outlines the calibration process, including adjustments to the motor direction, steering parameters, and ERPM gain, as well as modifications to the software speed limits [20][23][25].

Steering And Odometry Calibration

Steering and odometry calibration aligns the autonomy stack's command/state conventions with the physical vehicle. The software interface represents motion as linear velocity (m/s) and steering angle (rad), while the drivetrain requires motor ERPM/RPM and a bounded servo command. Calibration therefore consists of enforcing consistent sign conventions and tuning the gains/offsets used to convert between these domains so that commanded motion matches measured motion. [20][21][23][25]

Motor Direction and Odometry Sign Consistency

Motor polarity must be verified before tuning conversion parameters. A positive velocity command must produce forward wheel rotation. Odometry sign is then validated by confirming that forward motion increases `/odom` position in the +x direction. If `/odom` position decreases during forward motion, the VESC-to-odometry speed interpretation is inverted in the conversion logic and the software workspace is rebuilt so the corrected sign propagates through the stack. [20][23]

Steering Offset for Straight Tracking

The steering offset determines whether the vehicle tracks straight when the commanded steering angle is zero. Straight-line driving tests over several meters are used to observe lateral drift. The parameter `steering_angle_to_servo_offset` is adjusted iteratively in small increments until zero steering command yields straight tracking, compensating for mechanical servo alignment and linkage tolerances. [20][21][25]

Steering Gain for Curvature Accuracy

Steering gain maps commanded steering angle to the servo actuation range and directly controls turning radius. A geometric target can be computed using wheelbase and maximum steering angle, then validated experimentally using a constant steer turning test. The vehicle is driven slowly at maximum steering, and the resulting half-circle diameter is measured; the steering gain is adjusted until the measured diameter matches the target. If left and right steering excursions are asymmetric, servo bounds (`servo_min`, `servo_max`) are refined to enforce symmetric steering authority. [21][25]

Speed-To-ERPM Mapping and Odometry Scale

Odometry scale depends on the conversion from commanded velocity (m/s) to motor ERPM and the inverse mapping used in state estimation. A stationary check is performed first; drift in `/odom` position at rest indicates a bias that is corrected via `speed_to_erpm_offset`. Straight-line runs over a measured distance (>3 m) are then used to compare tape-measured displacement to `/odom` displacement. If odometry overestimates distance, `speed_to_erpm_gain` is decreased; if it underestimates, the gain is increased. Tuning continues until distance error is within a small tolerance. [20][23][25]

Software Speed Limits

Operational speed bounds are set via `speed_min` and `speed_max` in `vesc.yaml`. These software limits are kept consistent with the VESC firmware ERPM hard limit to maintain a layered safety envelope. These parameters are especially important as they dictate the "simulated player speed" for initial pilot testing with OU softball. [20][25]

```
# joy_teleop.yaml
human_control:
  axis_mappings:
    drive-speed:
      axis: 1
      scale: 1.0      # reduced from 5.0 for finer low-speed control
    drive-steering_angle:
      axis: 2
      scale: 0.25
# vesc.yaml
speed_to_erpm_gain: 4614.0
tachometer_ticks_to_meters_gain: 0.00225
steering_angle_to_servo_gain: -1.0294
steering_angle_to_servo_offset: 0.50
wheelbase: 0.25
max_acceleration: 1.2
throttle_smoother_rate: 75.0
max_servo_speed: 3.2
servo_smoother_rate: 75.0
vesc_driver:
  current_max: 35.0
  speed_min: -6000.0
  speed_max: 6000.0
  servo_min: 0.15
  servo_max: 0.85
# VESC Tool (not YAML)
Minimum ERPM: 50-150
```

Figure 5.2.1-1: YAML Modifications for VESC Tuning and Steering/Odometry Calibration

5.2.2 Autonomous Control Implementation

Bringup and Deadman's Switch:

To enable autonomous control, the launch bringup must be initiated as described in the teleoperation and LiDAR-testing section. Once bringup is active, commands can be published to the /drive topic. These commands only pass through while the deadman's switch is held, ensuring that autonomous control is enabled only when explicitly commanded by the user [13][21][25].

Developing Directly in the Driver Stack Container:

The most straightforward approach is to develop directly within the driver stack container.

This method leverages the pre-configured dependencies in the container, simplifying the setup process. To create a new package alongside f1tenth_system, dependencies must be defined in the package.xml. Once the custom package is added, nodes can be launched either by creating a new launch file or by adding them to the existing bringup launch file [13][25].

5.3 Challenges Encountered and Solutions Implemented

Challenges

- Part acquisition lead times
- ROS/Castle Link Data Logging
- Getting feedback and interaction with OU softball staff
- Research on autonomous systems

Solutions

- Switched from Castle Link to VESC Tool for data logging
- Contacted the Softball Director of Operations rather than a coach
- Used resources and faculty from the iHub FabLab to learn more about ROS and LiDAR

Chapter 6: Conclusions and Future Work

6.1 Summary of Contributions

- Conducted 8–10 hours of weekly research on software installation and LiDAR integration for RC cars
- Participated in the OU startup accelerator, showcase, and Founders & Residence program and secured funding

- Designed and manufactured a platform for mounting and protecting autonomy hardware
- Ran tests across multiple days and operating conditions to verify the repeatability of the Castle Link tuning setup
- Integrated and calibrated the driver stack on Jetson compute
- Evaluated system operation using VESC Tool and RViz

6.2 Potential Applications

6.2.1 Civilian Applications (Sports, Logistics, Hobbyist, etc.)

The SAGV can be adapted for several civilian use cases because its low-cost chassis, modular hardware, and LiDAR-based navigation support different operating environments. Its adaptability also makes it suitable for applications in sports, research, education, security, and hobbyist communities [12][26].

Sports and Athletics:

One of the most immediate applications of the SAGV is in sports training and athletics. As demonstrated in its initial use case with OU softball, the vehicle can simulate player movements, reducing the risk of injury for athletes during practice sessions. By mimicking the acceleration, deceleration, and cornering of a player running bases, the SAGV provides a dynamic training tool. Beyond softball, similar applications can be extended to other sports, such as football, basketball, or track and field, where autonomous vehicles can act as moving targets or pace setters. This innovation not only enhances training efficiency but also minimizes physical strain on athletes.

Research and Development:

Because the SAGV uses modular hardware and a ROS-based navigation stack, it can serve as a test platform for obstacle detection, mapping, and autonomy experiments. Universities and research institutions can use the vehicle to test algorithms for obstacle detection, path planning, and SLAM (Simultaneous Localization and Mapping). Its ability to operate in GPS-denied environments makes it particularly valuable for studying indoor navigation or exploring complex terrains. Additionally, the use of additive manufacturing allows researchers to rapidly prototype and customize components, enabling experimentation with different hardware components and materials. [10][13][26]

Education and STEM Outreach:

The SAGV can serve as an engaging tool for STEM education, inspiring young minds to explore robotics, programming, and engineering. Schools and community programs can use the vehicle to teach students about autonomous systems, sensor integration, and 3D printing. Its affordability and scalability make it accessible to educational institutions with limited budgets. [11][13]

Hobbyist and Maker Communities:

For hobbyists and makers, the SAGV provides a modular platform for hardware customization and ROS-based navigation experiments. Its open-source software stack, compatibility with ROS, and modular hardware design allow enthusiasts to modify and enhance the vehicle according to their interests. Whether it's building a racing robot, creating

a home automation system, or exploring autonomous navigation, the SAGV empowers individuals to bring their ideas to life. [13][25]

6.2.2 Military and Defense Applications

The semi-autonomous ground vehicle (SAGV) also holds significant potential for military and defense applications, offering cost-effective solutions for tasks that traditionally require expensive equipment or pose risks to human life. Its compact design, advanced navigation capabilities, and modular construction make it an asset in various operational scenarios. [12]

Unmanned Reconnaissance and Surveillance:

Equipped with LiDAR and autonomous navigation systems, the SAGV can be deployed as an unmanned ground vehicle (UGV) for reconnaissance missions. Its ability to operate in GPS-denied environments allows it to navigate through tunnels, dense forests, or urban areas where traditional GPS-based systems may fail. This concept entails the SAGV would be used as a one-way payload delivery device. By integrating additional sensors such as cameras or thermal imaging devices, the SAGV can gather real-time intelligence on enemy positions, terrain conditions, or potential threats, providing critical data to military personnel without exposing them to danger. [12][19]

Explosive Ordnance Disposal (EOD):

The SAGV's modular design and durability make it an ideal platform for bomb disposal operations. By outfitting the vehicle with robotic arms or specialized tools, it can safely approach and neutralize explosive devices. Its lightweight construction and precise

navigation capabilities enable it to maneuver in confined spaces, such as under vehicles or within buildings, where larger EOD robots may struggle. [12]

Delivery of Payloads and Supplies:

The SAGV can be used to transport supplies, equipment, or payloads in combat zones or disaster-stricken areas. Its autonomous navigation system ensures safe and efficient delivery, even in challenging terrains or hostile environments. For example, the vehicle can carry medical supplies to injured personnel or deliver ammunition to troops in the field. Additionally, its compact size and low cost make it suitable for deployment in large numbers, enabling scalable logistics solutions. [12][26]

Deployable Explosives and Tactical Operations:

In offensive scenarios, the SAGV can be adapted to carry and deploy explosives or other tactical payloads. Its autonomous navigation capabilities allow it to approach targets with precision, minimizing collateral damage and maximizing operational effectiveness. For instance, the vehicle could be programmed to infiltrate enemy defenses and deliver a payload to a specific location, acting as a low-cost alternative to traditional drones or guided munitions. [12]

Training and Simulation:

The SAGV can also be used in military training exercises by simulating enemy movement or providing dynamic targets for soldiers to engage. Current military initiatives place high importance on both aerial and ground systems. The SAGV was demonstrated in person at the

OKNG UAS demo event “Thunderstruck” in September 2025. The event focused on preparing soldiers for drone-integrated combat in both offensive and defensive operations [12].

6.3 Future Work

6.3.1 Integration of Additional Sensors (e.g., Cameras, IMUs)

The next development step is to expand the sensor suite so the vehicle can estimate motion and classify obstacles more reliably. Integrating cameras will enable advanced computer vision capabilities such as object recognition, lane detection, and environmental analysis. Similarly, incorporating inertial measurement units (IMUs) will enhance the vehicle’s ability to measure acceleration, orientation, and angular velocity, improving stability and navigation on uneven terrain. The VESC has built-in IMU sensors, but they were not used in this project. These sensors can complement the existing LiDAR system, enabling sensor fusion for more accurate mapping and localization. The integration of additional sensors will also open the door to more advanced AI-driven applications, such as gesture recognition or dynamic path planning [12][23][26].

6.3.2 Advanced SLAM and AI Capabilities

Adding more advanced SLAM methods and AI-based perception would improve capability, but it would also require more compute, development time, and sensor integration. While the current system utilizes basic SLAM for navigation and mapping, upgrading to more sophisticated algorithms, such as Graph-Based SLAM or Particle Filters, will improve mapping accuracy and computational efficiency. These advancements would allow the

SAGV to operate seamlessly in larger, more complex environments, including multi-level structures or dynamic terrains. Additionally, integrating AI-driven decision-making systems will enable the vehicle to adapt to unpredictable scenarios, such as moving obstacles or changing environmental conditions. Machine learning models can be trained to optimize path planning, enhance collision avoidance, and predict user behavior in real-time. These upgrades will significantly expand the SAGV's operational scope, making it suitable for high-stakes applications like disaster response, autonomous racing, or military reconnaissance. [12][19][26]

6.3. Transition to true 3D Point Cloud Generation

Extending the current 2D LaserScan pipeline to a true 3D Point Cloud perception is an expensive but necessary step in enabling height-aware obstacle segmentation and improving localization during aggressive maneuvers. A 3D representation would reduce ambiguity from ground clutter and overhangs, strengthen SLAM in low-feature areas, and support safer, higher-speed planning using volumetric costmaps. [26]

6.4 Closing Remarks

The SAGV project has been important to my development during the latter half of college. It has opened many opportunities for me. My internship at Northrop Grumman and my startup venture in the Accelerator program have both strengthened my interest in higher education and academic growth outside the classroom.

Patty Wagon, Ghost Runner, Base Runner, and SAGV are all names I have used for the car. As the project continues to evolve through new opportunities in the DoD, DIU, collegiate

athletics, and local commercialization, I believe the research documented in this report will provide a strong foundation for my future career and business endeavors.

References

(1) I. Gibson, D. Rosen, and B. Stucker, Additive manufacturing technologies: 3D printing, rapid prototyping and direct digital manufacturing, Second Edition. New York Heidelberg Dodrecht London: Springer, 2015.
(2) “Centre for Automotive Research.” *University of Waterloo*, n.d., https://uwaterloo.ca/centre-automotive-research/. Accessed 10 Feb. 2025.
(3) “Course.” *RoboRacer*, n.d., https://roboracer.ai/course.html. Accessed 10 Feb. 2025.
(4) “Article from ScienceDirect.” *ScienceDirect*, Elsevier, n.d., https://www.sciencedirect.com/science/article/abs/pii/S1755581724000907?via%3Dihub. Accessed 10 Feb. 2025.
(5) Xiao, Xinyi, and Hongbin Li. “Predicting Mechanical Responses of Additively Manufactured Metamaterials with Computational Efficiency.” *CIRP Journal of Manufacturing Science and Technology*, vol. 52, 2024, pp. 149–158.
(6) Bai, Long, et al. “Mechanical Properties and Energy Absorption Capabilities of Functionally Graded Lattice Structures: Experiments and Simulations.” *Forthcoming Manuscript*, n.d.
(7) CL2-UWaterloo. *CL2-UWaterloo/f1tenth_ws,* GitHub, n.d., https://github.com/CL2-UWaterloo/f1tenth_ws. Accessed 10 Feb. 2025.
(8) SLAMTEC. *LD601 SLAMTEC RPLIDAR Datasheet S1 v1.4,* n.d., http://bucket.download.slamtec.com/f19ea8efcc2bb55dbfd5839f1d307e34aa4a6ca0/LD601_SLAMTEC_rplidar_datasheet_S1_v1.4_en.pdf. Accessed 10 Feb. 2025.
(9) “Build.” *RoboRacer*, n.d., https://roboracer.ai/build.html. Accessed 10 Feb. 2025.
(10) “Research.” *RoboRacer*, n.d., https://roboracer.ai/research.html. Accessed 10 Feb. 2025.
(11) Horváth, E.; Ignéczi, G.; Markó, N.; Krecht, R.; Unger, M. Teaching Aspects of ROS 2 and Autonomous Vehicles. Eng. Proc. 2024, 79, 49. https://doi.org/10.3390/engproc2024079049
(12) (Loganathan, Anbalagan, and Nur Syazreen Ahmad. “A Systematic Review on Recent Advances in Autonomous Mobile Robot Navigation.” *Procedia Computer Science*, vol. 40, 2023, pp. 348–355. Link to PDF. Accessed 10 Feb. 2025.
(13) (F1Tenth. “F1Tenth Documentation.” F1Tenth, https://f1tenth.org. Accessed 1 Mar. 2025.
(14) (Open Robotics. “ROS 2 DDS Communication.” ROS 2 Documentation, https://docs.ros.org. Accessed 1 Mar. 2025.
(15) Open Robotics. “Real-Time Performance in ROS 2.” ROS 2 Documentation, https://docs.ros.org. Accessed 1 Mar. 2025.
(16) (Open Robotics. “ROS 2 Cross-Platform Support.” ROS 2 Documentation, https://docs.ros.org. Accessed 1 Mar. 2025.
(17) Open Robotics. “Multi-Threaded Execution in ROS 2.” ROS 2 Documentation, https://docs.ros.org. Accessed 1 Mar. 2025.
(18) Open Robotics. “End of Support for ROS 1.” ROS 2 Documentation, https://docs.ros.org. Accessed 1 Mar. 2025.
(19) University of Waterloo. “F1Tenth LiDAR-Based SLAM Demonstration.” Robotics Research Lab, https://uwaterloo.ca. Accessed 11 Mar. 2025.

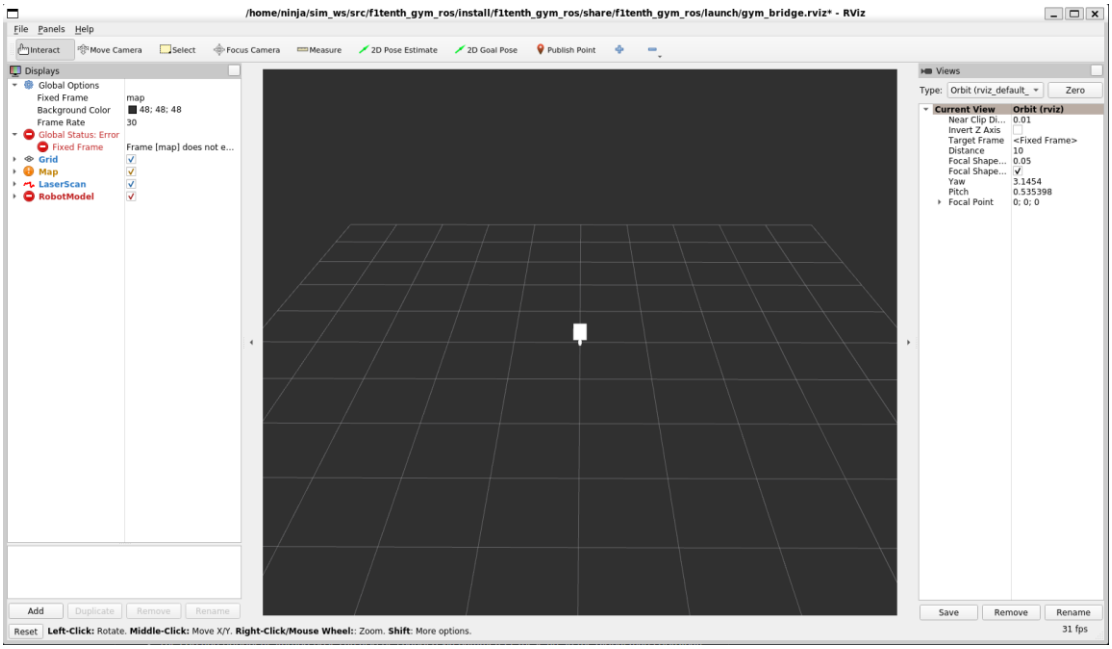
(20) F1Tenth. “VESC Driver Node in F1Tenth.” F1Tenth, https://f1tenth.org. Accessed 11 Mar. 2025.
(21) F1Tenth. “AckermannDriveStamped Messages.” F1Tenth, https://f1tenth.org. Accessed 11 Mar. 2025.
(22) Open Robotics. “ROS 2 Publish/Subscribe Model.” ROS 2 Documentation, https://docs.ros.org. Accessed 11 Mar. 2025.
(23) CRA-ROS-PKG. “robot_localization Package.” GitHub, https://github.com/cra-ros-pkg/robot_localization. Accessed 14 Mar. 2025.
(24) F1Tenth. “RRT* Planning for F1Tenth.” F1Tenth, https://f1tenth.org. Accessed 14 Mar. 2025.
(25) F1Tenth. “F1Tenth System Workspace Layout.” F1Tenth, https://f1tenth.org. Accessed 16 Mar. 2025.
(26) Autoware Foundation. “Autoware.Auto on F1Tenth Demonstration.” Autoware.Auto Documentation, https://autowarefoundation.gitlab.io/autoware.auto. Accessed 18 Mar. 2025.
(27) Castle Creations. *Castle Link*. Castle Creations, n.d., https://home.castlecreations.com/cclinks. Accessed 31 Mar. 2025.
(28) Castle Creations. *Castle Link*. Castle Creations, n.d., h https://home.castlecreations.com/blog/2021/10/12/castle-creations-data-logging-explained-start-to-finish.
(29) “Airless Tire Technology and Applications.” Michelin Group, 2023, https://www.michelin.com.
(30) Gibson, Ian, David W. Rosen, and Brent Stucker. Additive Manufacturing Technologies: 3D
(31) Printing, Rapid Prototyping, and Direct Digital Manufacturing. 3rd ed., Springer, 2021.
(32) Markforged. “TPU Material Properties and Design Guide.” Markforged Technical Resources, 2023, https://markforged.com/materials/tpu.
(33) Prusa Research. Flexible Filaments (TPU/TPE) 3D Printing Handbook. Prusa Research, 2023.
(34) Rosen, David W. “Design for Additive Manufacturing: A Method to Explore Unexplored Regions of the Design Space.” Journal of Mechanical Design, vol. 132, no. 9, 2010, pp. 091004–091011.
(35) Sutradhar, Alok, et al. “Topology Optimization for Additive Manufacturing.” Additive Manufacturing, vol. 27, 2019, pp. 390–400.
(36) Wong, Kaufui V., and Aldo Hernandez. “A Review of Additive Manufacturing.” ISRN Mechanical Engineering, 2012, Article ID 208760.
(37) Daman Pak, A., Lotfi, J., & Khalili, S. M. (2023). A finite-strain simulation of 3D printed airless tires. Journal of Engineering Mechanics, 149(10). https://doi.org/10.1061/jenmdt.emeng-7169
(38) Jafferson, J. M., & Sharma, H. (2021). Design of 3D printable airless tyres using ntopology. Materials Today: Proceedings, 46, 1147–1160. https://doi.org/10.1016/j.matpr.2021.02.058

Appendix

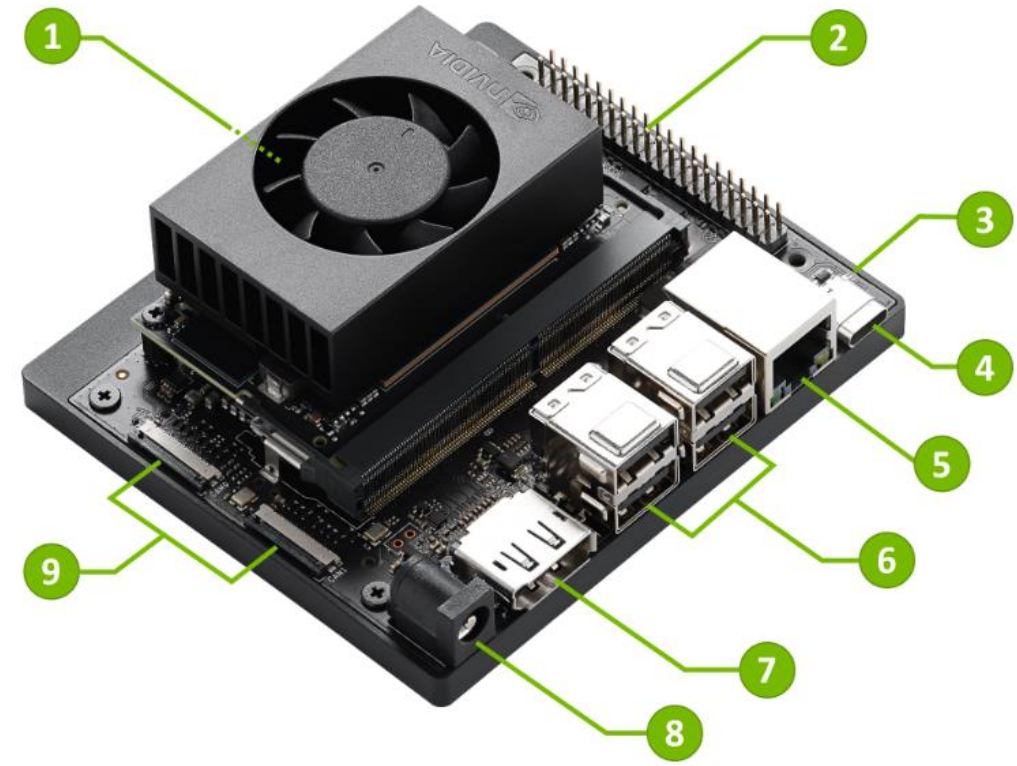
Undertray with Impact Area and Pool Noodle Mount

Assembled V1 Base Runner

Another Undertray Picture

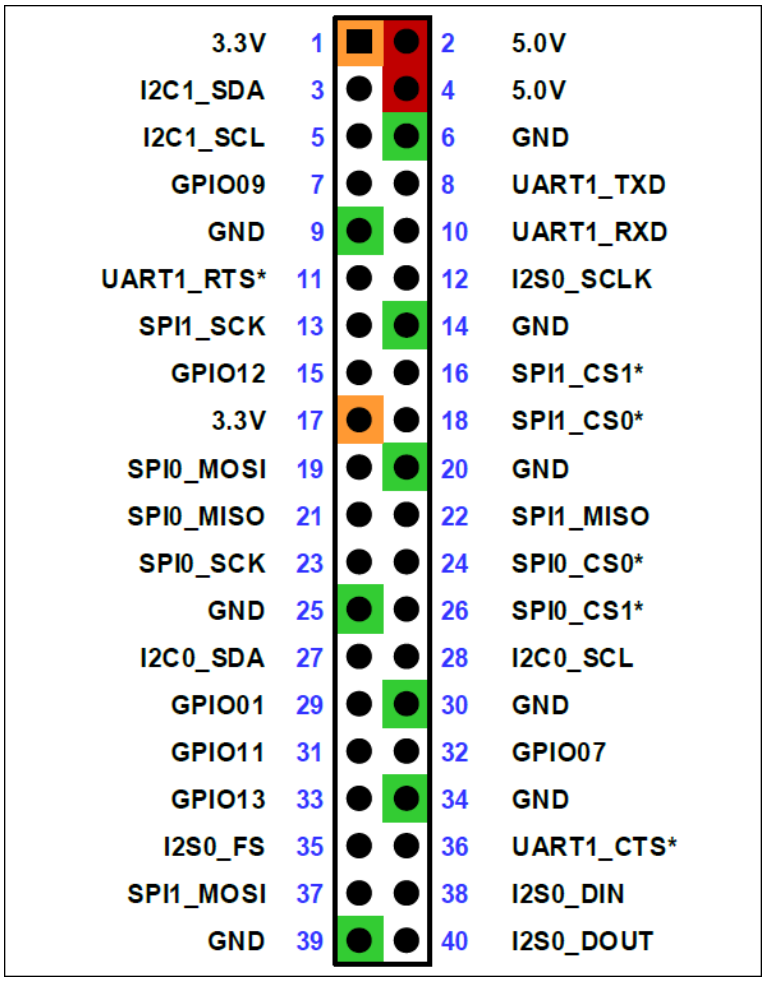


Rviz running on Ubuntu 20.04.6 (errors)

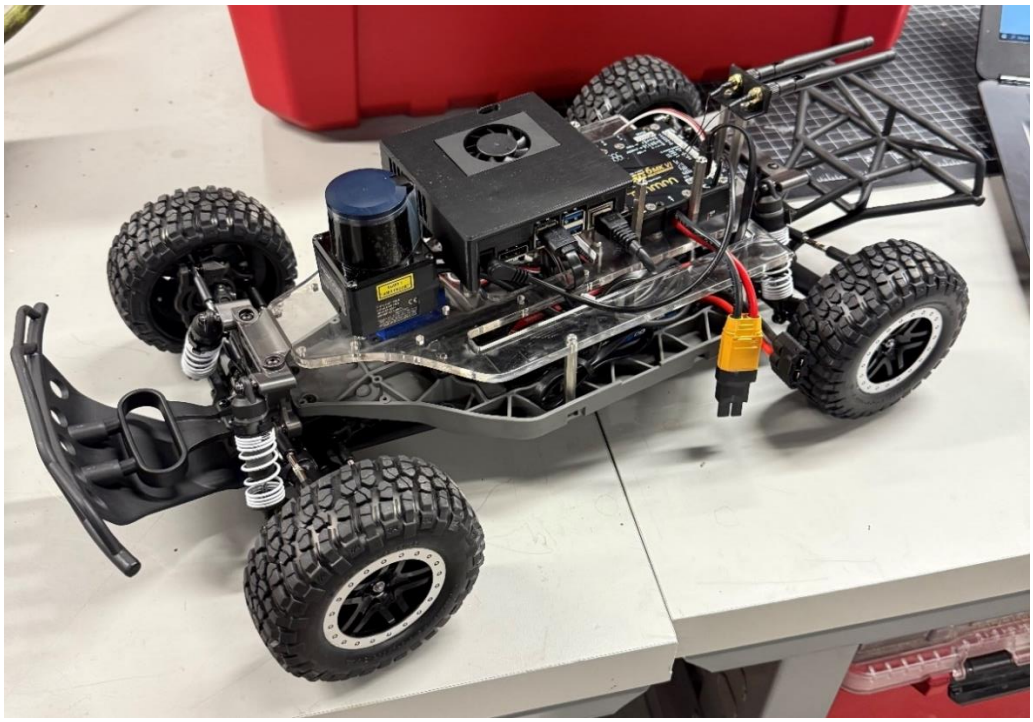


1 microSD card slot for main storage
2 40-pin expansion header
3 Power indicator LED
4 USB-C port for data only
5 Gigabit Ethernet port
6 USB 3.1 Type A ports (x4)
7 DisplayPort connector
8 DC Barrel jack for 19V power input
9 MIPI CSI camera connectors

Jetson Orin Nano Diagram



Jetson Orin Nano Header Pin Diagram



Completed SAGV Build No Cover