Virginia Polytechnic Institute and State University

VT CRO - Bowser

Submitted May 15, 2025

Team Member	Email
Joe Bekiranov	jb5@vt.edu
Wyatt Dillon	wyattdillon13@vt.edu
Tom Drinkwater	tjadrinkwater@vt.edu
Yianni Fifis	yianni24@vt.edu
Tim Fish	timothyfish@vt.edu
Cassie Freedlander	cfreed@vt.edu
Sree Motukuri	smotuku@vt.edu
Sam Ramey	sramey02@vt.edu
Tommy Savory	tsavory@vt.edu
John Shebey (Team Captain)	jshebey@vt.edu

Advisors/Mentors
- Marie Paretti, PhD, Professor of Engineering Education at Virginia Tech
- Matt Nowinski, Visiting Professor of Practice at Virginia Tech
- Sanjana Bharadwaj, Software Engineer at Torc Robotics
- Hope Ugalde, Software Engineer at Torc Robotics

Statement of Integrity
I hereby state the engineering design and development of Bowser was completed by the current student team and is equivalent to a senior design capstone course at Virginia Tech.

[illegible]

Matt Nowinski, mcnowins@vt.edu

2. Team Organization

2.1 Introduction

The Competitive Robotics Organization (CRO) at Virginia Tech is excited to introduce Bowser to this upcoming AutoNav Intelligent Ground Vehicle Competition (IGVC). Bowser has many upgrades, and a brand-new design compared to previous teams at Virginia Tech.

2.2 Organization

The team consists of 10 students from different engineering disciplines: Joe Bekiranov (computer engineering), Wyatt Dillon (electrical engineering), Tom Drinkwater (mechanical engineering), Yianni Fifis (mechanical engineering), Tim Fish (computer engineering), Cassie Freedlander (mechanical engineering), Sree Motukuri (mechanical engineering), Sam Ramey (computer engineering), Tommy Savory (computer engineering), and John Shebey (computer engineering). Bowser was broken into mechanical, electrical, and software subsystem. Each subsystem had a designated chief engineer that led weekly sub team meetings. Additionally, two juniors (Sree and Cassie) joined in February and worked on creating the driving mechanism for the Bowser plushie. Figure 1 shows a team organization chart.

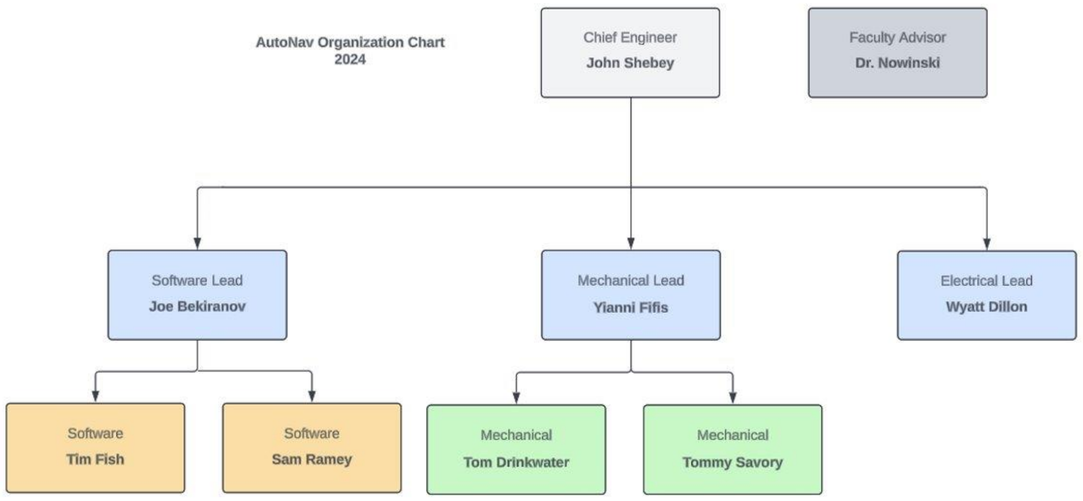


Figure 1: Team Organization Chart

The team also had weekly general meetings with Dr. Nowinski to give sub team updates, and make sure everyone understood the current state of Bowser. Additionally, the team meets twice a week in the lab to get work done and collaborate with other teammates. Each team member put in about 6 hours of work a week (not including meetings) for two 15-week semesters. This means the team has dedicated about 1500 total hours to the project.

2.3 Design Process

To develop a fully autonomous vehicle for the IGVC, the team used a waterfall approach with 5 major design phases as follows:

Phase 1 Research and Planning: Looked at previous years IGVC team designs and created requirements

Phase 2 Prototyping Initial Design: Built a prototype and made a bill of materials (BOM) once initial designs were made

Phase 3 Subsystem Design and Development: Each sub team designed and built their subsystem

Phase 4 Subsystem Integration: Integrated each subsystem of the robot and ensured that the sensors could communicate with each other and control the chassis of the vehicle.

Phase 5 Testing and Validation: Tested the entire functionality of the robot and using a test track

3. System Architecture

Table 1: Significant Components

Quantity	Component	Identification
2	24VDC Brushless Motor	Mechanical
1	24VDC Lithium-Ion Phosphate Battery	Electrical
3	DC-DC Down Converters (19V, 12V and 5V)	Electrical
1	Motor Controller	Electrical
1	Nvidia Jetson Orin Nano	Electrical/Software
1	Arduino Uno	Electrical/Software
1	Stack-On (Safety) Light	Electrical
1	Novatel GPS	Electrical/Software
1	SICK LiDAR	Electrical/Software
1	Zed2i Camera	Software

The main safety features of the electrical system are the DC current breakers, finger-safe wire terminals, as well as neat and organized wire routing. This protects not only the devices on the robot but the people operating the robot as well

The software team designed and implemented autonomous control for the robot using Robot Operating System (ROS). The software architecture consists of the following components:

- **Perception:** The ZED camera and the SICK LiDAR provide the robot with vision and use this data to make decisions
- **Navigation:** Nav2 is used to take the LiDAR and camera data and generate a cost map showing where nearby obstacles are
- **Path Planning:** Uses a path planning algorithm like A* to find the most efficient path for the robot to take. This is done by using the cost map data and GPS waypoints
- **Control:** Takes the desired path and converts it into motor speeds so the robot can move to the desired location

A full system diagram can be found below in Figure 2.

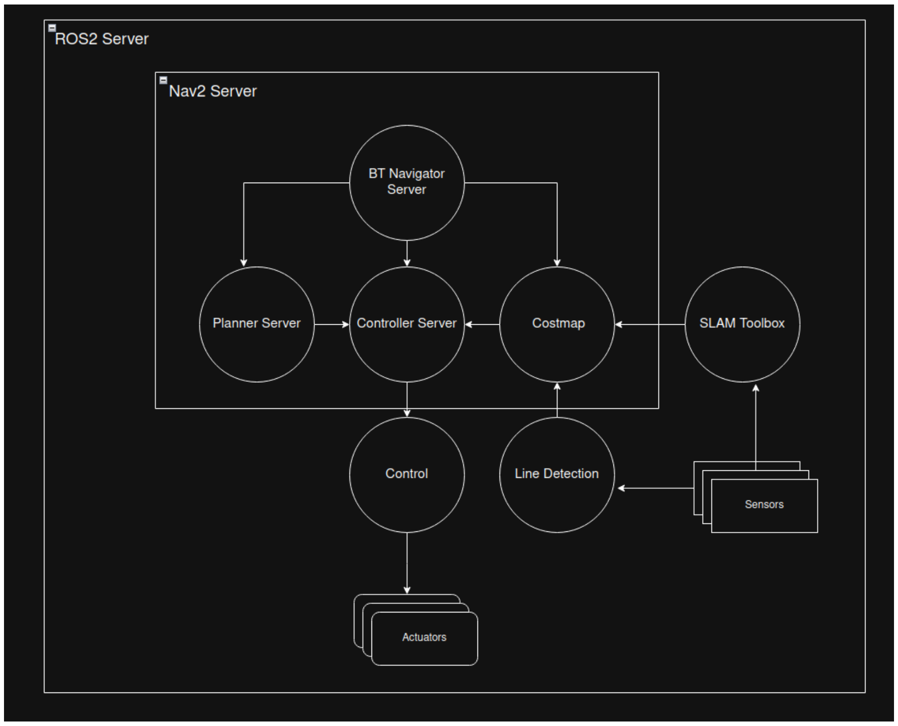


Figure 2: Software system design diagram

The interconnection of the electrical and software systems is done through the use of serial communication interfaces both the Jetson and an auxiliary Arduino. The system diagram for electrical-software integration can be found in Figure 3.

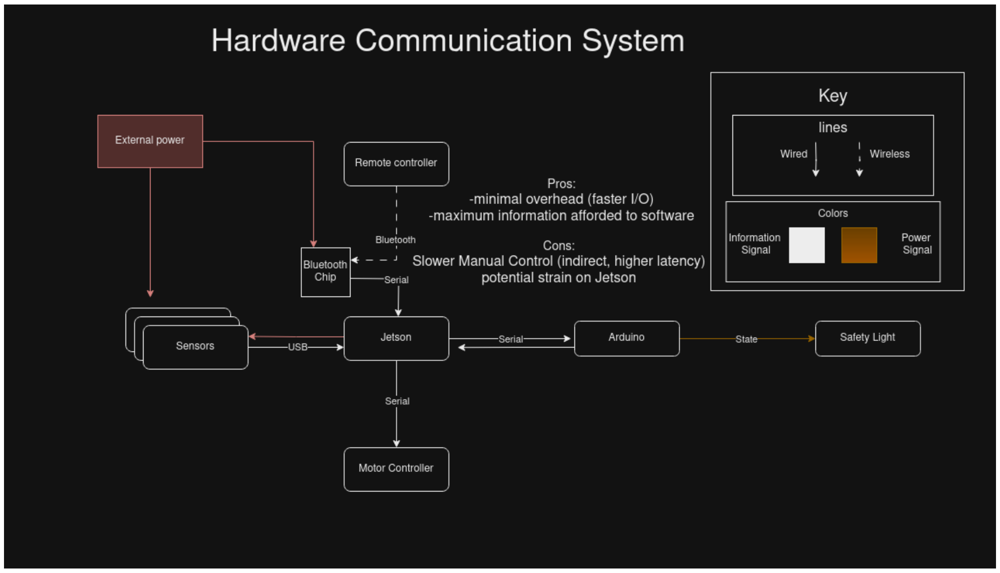


Figure 3: Hardware-Software communication system

4. Innovations

4.1 Software

The software team's key innovation is the line detection system, which uses per-point GPU accelerated custom line detection for quickly and robustly classifying line pixels, instead of attempting to parameterize an entire line.

This innovation was arrived at due to the lack of rotational and scalar invariance found in most line detection methods. Deep learning methods were researched, and while some segmentation architectures seemed promising, they were often weak to the intense rotational transformations that occur frequently in IGVC due to the nature of the course. This made pre-trained models brittle, and without sufficient data, fine tuning was difficult. Therefore, literature was searched, and the CERIAS characteristic was discovered. The per-point line detection fit our use case particularly well as it could be easily integrated into the costmap paradigm through a series of frame transformations from the camera.

4.2 Mechanical

The mechanical team completely redesigned Bowser this year. The prominent innovation is the drive train. Bowser's drive consists of two independently driven motors at the center axis and two free spinning caster wheels; one at the fore and aft of the robot.

The drive and component selection allows for Bowser to turn within its own footprint while mitigating the majority of the scrubbing friction experienced in turning. The driven wheels are mounted directly onto the frame of Bowser. The caster wheels are mounted on a pinned control arm which is supported by an air spring at the free end. This allows vertical deflection that permits the driven wheels to maintain contact with the ground while the robot is driving on, over, and off the ramp obstacle.

5. Mechanical Design

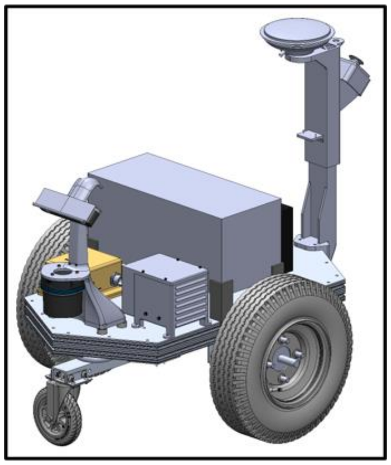


Figure 4: Final 3D Model of Bowser

5.1 Chassis Overview

The chassis serves as the robot's load-bearing framework, supporting all components. The team chose a hexagonal frame made from T-Slotted aluminum extrusion for its strength, light weight, and modularity, allowing for design flexibility. Custom aluminum 6061-T6 angled brackets were created to maintain the robot's shape and reinforce corners. The aluminum 6061-T6 baseplate acts as both a shaping template and the mounting surface for electrical components.

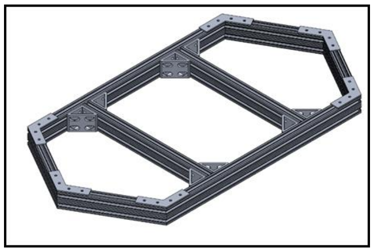


Figure 5: Chassis without base plate mounted

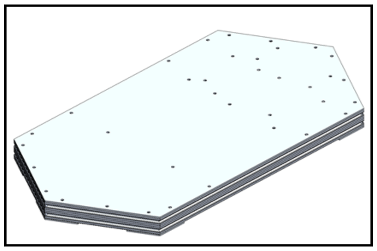


Figure 6: Frame with base plate mounted.

5.2 Power Train Overview

Bowser's power train was designed to accelerate the robot and to support the major vertical loading. The power train is driven by two 24v DC brushless motors which each directly drive a 16-inch wheel. The 16-inch wheel was selected to raise the ride height of the robot such that there was optimal space to fit a suspension system for the caster wheels. The wheel size also balances factors such as speed, acceleration, force, and traction.

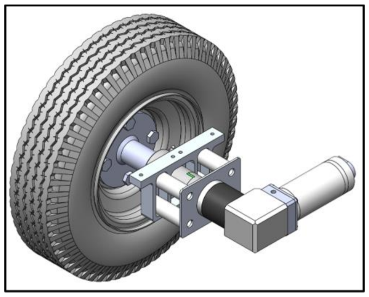


Figure 7: Power train subassembly

5.3 Caster Wheel Suspension System

The caster wheel suspension system is designed to support vertical loads, provide longitudinal stability, absorb ramp impacts, and dampen oscillations without affecting the robot's turning radius. It features a pinned control arm with an 8-inch caster wheel at each end of the robot to enable a zero-turn radius. The pinned design restricts lateral and longitudinal movement while allowing rotation. An upgraded air spring was added to maintain consistent stroke length, provide reliable restoring force, and improve stability with its larger contact base.

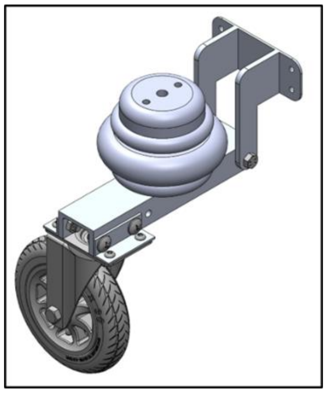


Figure 8: Caster Wheel Subassembly

5.4 Weatherproofing

To meet competition requirements for light rain conditions, the team designed protective housings for all sensitive electrical components. Each component was evaluated for potential water entry points, and custom 3D-printed housings were developed based on this analysis and the robot's design constraints. The housings were tested through simulated rain using a spray bottle and paper towels in multiple trials, confirming that all components remained operational and protected from water damage.

5.5 Cooling Methods

The mechanical team was responsible for ensuring all electrical components had proper cooling measures. The Nvidia Jetson and Arduino were the only components that required an independent cooling system. All other components were either in the open or did not require cooling. The housing for the Nvidia Jetson and Arduino featured internal pass throughs, vents, and a fan mounted on the bottom to ensure proper airflow was achieved. Additionally, the Nvidia Jetson is fitted with its own direct fan. The housing is shown in Figure 9.

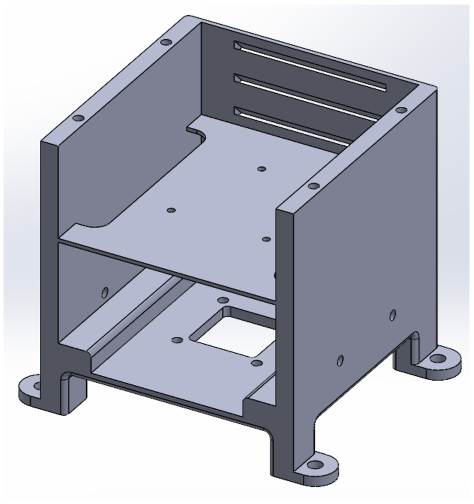


Figure 9: Nvidia Jetson and Arduino Housing without lid

6. Electrical and Power Design

6.1 Electrical Overview

The electrical team designed and set up all the power wiring for the LiDAR, GPS, Arduino, Jetson, safety light, E-stop, motors/motor controller, as well as all the necessary communication connections. The wiring was kept neat organized on the robot by using DIN rail, terminals, and wire harnessing. Additionally, circuit breakers and industry-standard electrical connectors were implemented to help protect equipment and people from electrical hazards. Figure 9 shows a complete wiring schematic of the power system.

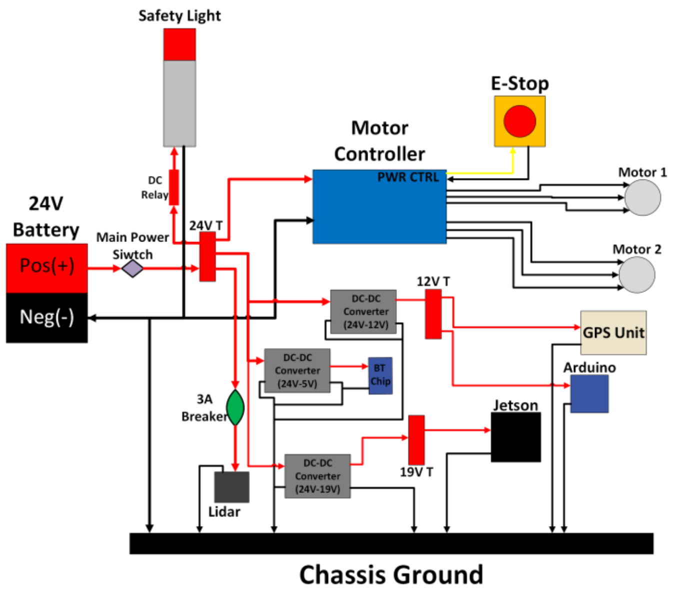


Figure 10: Circuit Wiring Diagram

6.2 Battery Life and Runtime

The power for the robot is distributed from the main 24VDC battery through the use of voltage terminals and industry standard wiring techniques. The battery in use can handle up to 25 Amp-hours which means the robot's maximum current draw must be under 25A. The robot running at maximum speed(5mph) draws roughly 13 Amps from the battery continuously which allows for a runtime of about 1.92 hours. Fortunately, the team has 2 batteries on hand which charge to full power within an hour, which will allow a dead battery to be easily swapped out and recharged before the next dies.

6.3 Mechanical and Wireless Emergency Stops

The mechanical E-stop requirement is met by installing a bright red button on the rear post of the robot which stops all commands from being sent to the motor controller which can be seen in Figure 10. However, the wireless E-Stop requirement is met by installing an HC-05 Bluetooth chip to the rear post that will be connected to a laptop. This laptop will be ready to send an ASCII signal that will perform the same action on the motor controller that the physical E-Stop does.

7. Software System Design

7.1 Line Detection Architecture

Lines in the IGVC are a special challenge compared to traditional autonomous lane detection due to the greater invariance necessary to solve the problem. The robot must navigate through tight obstacle tracks, which requires it to rotate almost past orthogonal to the lane line. This means that not only must our algorithm spot lines in a way that is fully rotationally invariant, it also cannot rely on image clues for line orientation, since the image of a single orthogonal line could equivalently be the left or right line. [3] suggests that parameterization, segmentation, and anchor point methods are leading research-based methods for classifying a lane line. Anchor based methods were too rigid for our use, since they could not handle high rotational invariance without great modification.

Furthermore, challenges are created in parameterization, since parametric functional forms for lines can be weak to certain line orientations ($y = mx + b$ cannot handle 'up'). Segmentation is promising and a considerable alternative but lack of training data and concern for our limited compute pushed us away. The team researched solutions in literature and eventually came upon [4].

7.1.1 Implementation

The CERIAS spatial characteristic line detection algorithm takes in an image and runs a set of processing steps independently on each pixel in the image. For each pixel (i, j), the intensity of all angles and orientation aligned pixels for angle θ and direction l is considered, $L_{(i,j)}(\theta, l)$. For each orientation and l, the standard deviation of this pixel line is

$$\sigma_{(i,j)}(\theta,l) = \sqrt{\frac{1}{N_{L_{(i,j)}}(\theta,l)-1}\sum_{(m,n)\in L_{(i,j)}(\theta,l)}(f(m,n)-\overline{f}_{L_{(i,j)}(\theta,l)})^2},$$

where

$$\overline{f}_{L_{(i,j)}(\theta,l)} = \frac{1}{N_{L_{(i,j)}}(\theta,l)-1}\sum_{(m,n)\in L_{(i,j)}(\theta,l)} f(m,n)$$

is the average intensity of the line. In the algorithm, this standard deviation measure is then used to accurately classify pixels on lines of uniform intensity of lines of width at least l. The result of the algorithm is a list of coordinates that are expected to be on a line. The algorithm was first implemented in python and tested on CPU. The result of the algorithm on a real image from the course is seen below in Figure 11.

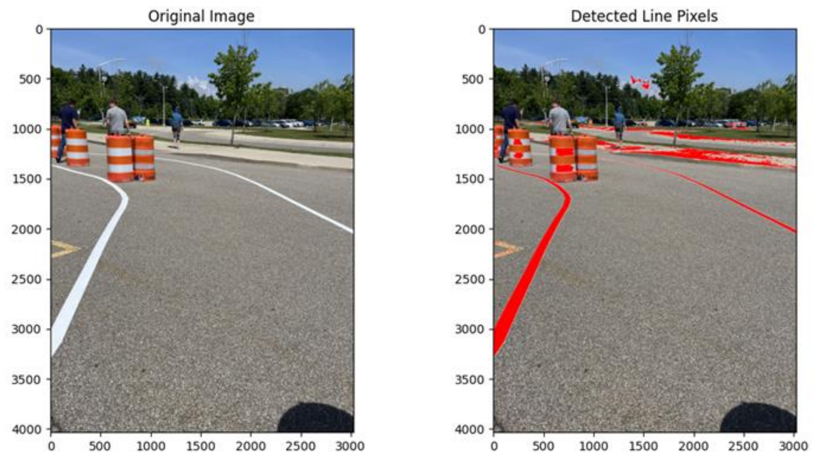


Figure 11: Line Detection Algorithm Result

7.1.2 Optimization

The team optimizes the algorithm for our hardware by implementing it as a custom CUDA kernel. The kernel is written such that blocks are warp optimized, and we utilize one thread for every pixel. The intensity standard deviation along all θ can be estimated using a fixed size square window. We pre-calculate the intensity integral image and its square on the GPU for every image using the NVIDIA NPP library, then utilize the integral image in the kernel to estimate mean and mean square values for the standard deviation in $\mathcal{O}(1)$ time.

Critical sections of the algorithms were timed and logged. The algorithm was run on the 3000 x 4000 image above in fig 2. (~12 million pixels) The results of the benchmarking can be seen below in Table 2.

Table 2: Line Detection Benchmark

Hardware	**Language**	**Execution Time (Seconds)**
CPU	Python	39.37
GPU	CUDA/C++	1.6 (24x speedup)

7.1.3 Sensor Fusion Application

This algorithm is nice because it has general purpose and could be functionally switched out for deep methods in use cases that have more data and compute available. A benefit of a general-purpose algorithm is ease of use in sensor fusion. The algorithm is shown on a camera image, but in fact works for any spatially local image. We make use of this by converting the Multiscan point cloud to a 2d image in the ground plane (x, y), and using the reflection intensity instead of pixel gray level. This leverages the 3d nature of the point cloud data over the camera, allowing us to take an effective slice of the ground and ignore all potential matches above ground in the camera image. With the speed-up from the GPU, we can run both the camera and LiDAR image through the kernel concurrently. The point data for each sensor is returned, and we fuse the point lists using a Bayesian Belief Network. This network utilizes per-sensor confidence in the form of sensor reported confidence or externally pre-computed precision and recall values to weight the incoming point lists. Due to the independence of these confidences, the network could easily handle more sensor types, so long as they could be usefully converted to a point list. The weighted point lists are averaged, and a static threshold is applied to determine binary line points. This fusion solves the problem of the noise detected in the algorithm output shown above.

Finally, the fused point list is written directly to a custom NAV2 cost-map layer plugin, skipping past the need for encoding the line into parametric form. This process allows us to process insane sensor throughput and classify lines with maximum information at very high frequencies, while still giving plenty of compute to the rest of the system.

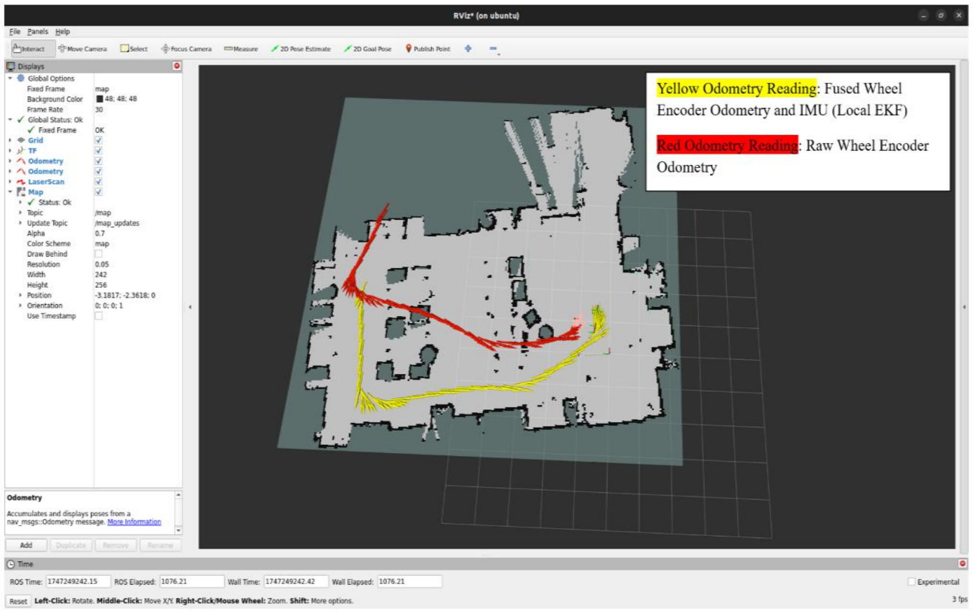
7.2 Object Detection and Environmental Representation

Obstacle detection in this system is functionally tied to environmental mapping. Since the robot is tasked with avoiding obstacles, this system uses mapping software and costmap layer tools to detect objects with the LiDAR. This system uses the Slam Toolbox library in conjunction with the Nav2 navigation library to accomplish this. Slam Toolbox is an open-source localization and mapping library that provides useful tools such as a costmap interface for environment representation, a behavior tree interface for simple and robust behavior control, and controllers and planner servers for centralized global and local planning. For obstacle detection, the team leverages SLAM and the Nav2 Voxel costmap layer plugin to create an accurate depiction of the environment and enable navigation to handle avoidance.

7.3.1 Mapping, Localization and Control

Our system combines real-time sensor data with historical data to create and continuously update a dynamic world model, which represents the environment around the robot. The robot collects current scene data through its LiDAR (SICK Multiscan 3D LiDAR), ZED2i Camera, and wheel encoders. The LiDAR provides detailed 2D scans and 3D point clouds, essential for detecting obstacles and mapping the surroundings, while the wheel encoders provide odometry data to track the robot's movement. Where the LiDAR fails, we utilize our ZED2i camera for lane line detection to ensure the robot stays in the bounds of the course.

To improve the accuracy of the robot's position, the wheel odometry and IMU data are integrated using the Extended Kalman Filter (EKF) from the robot_localization package. This process filters and combines these noisy sensor inputs to produce a more reliable estimate of the robot's position and orientation. The filtered odometry serves as the robot's pose estimate, providing a foundation for both localization and mapping.



Yellow Odometry Reading: Fused Wheel Encoder Odometry and IMU (Local EKF)

Red Odometry Reading: Raw Wheel Encoder Odometry

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
TF	✓
Odometry	✓
Odometry	✓
LaserScan	✓
Map	✓
✓ Status: Ok	
Topic	/map
Update Topic	/map_updates
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	242
Height	256
Position	-3.1817; -2.3618; 0
Orientation	0; 0; 0; 1
Use Timestamp	☐

Odometry

Accumulates and displays poses from a nav_msgs::Odometry message. More Information

ROS Time: 1747249242.15 ROS Elapsed: 1076.21 Wall Time: 1747249242.42 Wall Elapsed: 1076.21

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

Figure 12: Map Using LiDAR, Encoder, and IMU Data

The filtered data, along with real-time LiDAR scans, is then fed into the slam_toolbox for Simultaneous Localization and Mapping (SLAM). SLAM continuously updates the map of the environment by integrating new sensor data with previously collected data. This process allows the system to refine the map over time, creating a more accurate and comprehensive representation of the robot's surroundings as it moves.

Once the map is generated, it is passed into the Nav2 stack, which creates global and local costmaps used for path planning. These costmaps represent the robot's environment, accounting for obstacles and other factors that influence navigation. When a goal is set, Nav2 generates a path to the destination, and velocity commands are sent via the /cmd_vel topic to the controller node, which executes the path by controlling the robot's motion.

Throughout this process, the robot continuously updates its world model by combining new sensor data with older data. This allows the system to refine both the robot's map and its position in real-time. The integration of current and historical data ensures the robot can maintain an up-to-date understanding of its environment, supporting obstacle avoidance, precise localization, and efficient path planning. By continuously updating and refining its world model, the robot can make real-time adjustments, navigate effectively, and follow the planned path while avoiding obstacles and considering its environment.

Within the ROS system, control is directly derived from the created trajectory in Nav2. When the Nav2 controller takes in a path from the planner, it creates the necessary linear and angular velocities to achieve its path. Due to the pub sub architecture of ROS, these velocities can easily be subscribed to and manipulated to find the necessary left and right wheel speeds. Calculations are done based on both the angular and linear velocities, as well as the length of the wheelbase. The calculated left and right wheel speeds are fed directly into the motor controller. This gives the robot the correct and intended motion from the Nav2 controller.

7.3.2 Path generation and Waypoint Following

The system keeps track of waypoints and uses a global plan to navigate towards those waypoints. However, direct navigation is ill-advised as the robot would attempt to escape the lines at most points in the course. Thus, a low priority is assigned to global goals, and local goals are instead given higher priority. Local goals are created by a path generator module that occasionally generates a look-ahead point to navigate to. This is used to maintain the robot's 'forward' behavior that is expected by the course. The look-ahead points are overridden by new goals on a timer, which is important because the actual global trajectory of the robot over the course is roughly circular, and not linear. By resolving a lot of small linear look-ahead goals, small changes in local heading due to course curvature are unproblematic.

The system will eventually find a waypoint using the global plan. As the system finds its first waypoint, it changes into 'waypoint mode' to account for the landscape changes. Look-ahead points are disabled, and the robot can now directly navigate to the next waypoint directly. Local planning is still used for collision avoidance. The robot is able to navigate to the waypoints by using GPS data fused in through a Navsat GPS node, which converts GPS coordinates to the meter scale against our robot's base frame. In doing so, we can then transform the GPS coordinates to the map (world) frame and place the waypoints in the environment so they can be incorporated into planning. 'Waypoint mode' is exited upon finding the last waypoint, and look-ahead point generation begins again until the end of the course.

9. Final Vehicle Analysis

Table 3: Key Hardware Failures

Failure Point	**Failure Mode(s)**	**Mitigation**	**Resolution(s)**
3D printed component mounts	Cracking, delamination	Reducing the use of 3D printed mounts for structural mounts	Replacing with spare parts brought to competition, Emergency 3D printer
Poor Power Connections	Failure to power up important hardware	Implemented industry standard wiring techniques such as wire terminals and breakers	Use the electrical safety inspection to ensure the connections are stable as well as bring the necessary tools to fix loose connections

9.1 Mechanical Performance Analysis

The mechanical team was able to analytically prove that the robot could climb the ramp. Below are the steps taken to show this analysis. Since there are two motors that each output 130 in-lbs of torque, by Equation 1, the total driving force for two motors is 32.5 pounds.

$$Force(lbs) = Torque(in \cdot lbs) \div Radius(in) \qquad (1)$$

The total weight of Bowser with the payload is 135 pounds. The team was able to determine that the robot will be able to drive up the ramp obstacle by finding the maximum slope that the robot can climb by Equation 2.

$$F = mg \cdot \sin(\theta) \implies 32.5 = 135 \cdot \sin(\theta) \implies \theta \approx 14° \qquad (2)$$

The ramp obstacle will be a maximum of 15% grade. A percentage grade is the percentage of distance traveled horizontally to vertically. Therefore, we can estimate that the 14° slope is a 24.7% grade by Equation 3.

$$\%Grade = \tan(\theta) \cdot 100 \implies \%Grade = \tan(14) \cdot 100 \implies \%Grade = 24.7\% \qquad (3)$$

By this analysis, Bowser will successfully climb the ramp obstacle even after factoring in mechanical losses, friction, and other methods of error. By dividing the theoretical maximum grade by the competition grade, the factor of safety is 1.64. Furthermore, the team physically tested this by using a digital inclinometer and driving up a 10-degree incline. Bowser passed this real-world test successfully driving up and down the ramp.

9.2 Integration Lessons and Failure Modes

During the integration process we learned many lessons. The biggest and most recurrent lesson was to start simple and use modularity. If functionality is developed too far without testing, the failure modes compound and debugging becomes extremely difficult. Isolating tests for the smallest possible connection points allows for consistent fallback-style testing and failure mode isolation and speeds up both development and testing.

Safety, reliability, and durability were top considerations in both design and testing for the team. Safety was made a priority during development by integrating the safety systems first, and in testing by ensuring the safety systems were operational at all times (both e-stops, safety light). Reliability and durability were tested by strength of printed components and addressed by trading weight for more stable components throughout testing where necessary.

9.3 Software Project Management

Software testing was performed using segmented subsystem sub system tests through a variety of methods. Testing harnesses were created through the NAV2 Commander API, which allows an interface to communicate with a live NAV server using a low-boilerplate python script. Tests were run on each of the detection, planning, controller, and actuation components individually, and diagnostics were collected via logs, ROS server inspection, and Rviz. Version control was maintained through GitHub, with regular commits on team-specific branches. Bugs were tracked through a ticket system on Notion; every bug has a ticket a comment thread with details and discussion about the bug.

9.4 SIL Virtual Environment Simulation

Our team chose to utilize Gazebo for simulation due to its robust real-world physics engine, support for highly customizable environments, and seamless integration with ROS. To replicate the testing conditions of previous competitions, we constructed a simulated course based on video footage and images from prior years. This included the placement of obstacles, potholes, lane markings, and other key environmental features. To ensure thorough testing and validation of our software systems, we also integrated a detailed model of our robot into the simulation. This model accurately reflects the physical dimensions and sensor placements of the real-world robot. The simulation proved especially valuable in the early stages of development, as our team had undertaken a complete hardware redesign. Without access to a physical prototype during that period, the Gazebo environment allowed the software team to continue developing, testing, and refining subsystems in parallel with ongoing hardware development.

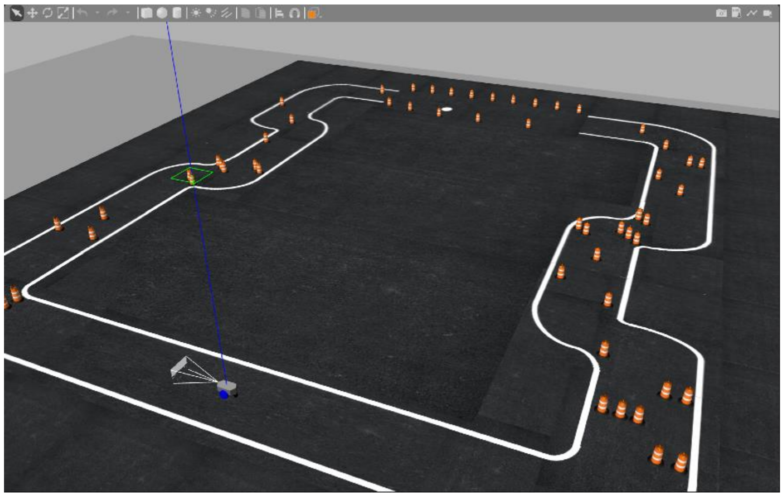


Figure 13: Simulated IGVC Course

10. Unique Software, Sensors, and Controls

The sensors and controls tailored for Autonav include the wheel setup and the line detection software. The design of the wheels allows for an in-footprint turn radius. This is specifically designed for this competition, where dead ends and tight switchbacks require robust movement to ensure that complex navigation can be performed efficiently. The line detection software is also specifically designed for this course. This software was researched specifically to fit the constraints of IGVC. It is specifically designed to be invariant to both scale and rotation, as well as account for line thickness and arbitrary curvature.

11. Performance Assessment

As of submitting this report, Bowser is nearing completion. The robot’s chassis and electronics are assembled and integrated with the vehicle. The software components are functional on their own but are still being tested and implemented on the vehicle. Although there is more software integration work to be done, we believe Bowser will be ready and fully operational by competition.

12. References

[1] “Isaac ROS Common” *Github.io*, 2024. https://nvidia-isaac-ros.github.io/repositories_and_packages/isaac_ros_common/index.html

[2] “Jetson Orin Nano Developer Kit User Guide - Hardware Specs,” *NVIDIA Developer*. https://developer.nvidia.com/embedded/learn/jetson-orin-nano-devkit-user-guide/hardware_spec.html

[3] “LanePtrNet: Revisiting Lane Detection as Point Voting and Grouping on Curves.” Accessed: Feb. 25, 2025. [Online]. Available: https://arxiv.org/html/2403.05155v1

[4] S. Liu, C. F. Babbs, and E. J. Delp, “Line Detection Using A Spatial Characteristic Model”.