Autonomous Robotic Vehicle Team
University of Michigan

mARVin 2.0

Submitted: May 15th, 2025

Team Captain: Sydney Belt (sydbelt@umich.edu)

Faculty Advisors: Xiaoxiao Du (xiaodu@umich.edu) & Damen Provost(provostd@umich.edu)

Statement of Integrity: The design and engineering of mARVin 2.0 has been significant and equivalent to what might be awarded credit in a senior design course.

Team Roster (*Lead, **Assistant Lead)					
Executive	Sydney Belt	sydbelt@umich.edu	**Navigation & Sensors**	Maaz Hussain*	maazh@umich.edu
	Arnav Shah	arnshah@umich.edu		Yamato Miura*	yjmiura@umich.edu
	Taylor Nguyen	taylorng@umich.edu		John Rose**	johnrose@umich.edu
	Chris Erndteman	chrisern@umich.edu		Ethan Hardy	hardyem@umich.edu
Business	Anshul Mohanty*	manshul@umich.edu		Ryan Liao	ryanliao@umich.edu
	Connor Pang*	pangc@umich.edu		Sunny Xu	sunnyx@umich.edu
Platform	Chloe Akombi*	cjakombi@umich.edu		Helena Sieh	helenasi@umich.edu
	Cara Blashill**	blashill@umich.edu		Natasha Sieh	nsieh@umich.edu
	Drew Boughton	drbought@umich.edu		Caitlyn Trievel	ctrievel@umich.edu
	Yuri Carnino	ycarnino@umich.edu		Erika Chen	erikachn@umich.edu
	Hazel Tao	hazeltao@umich.edu		Serena Saleh	serenasa@umich.edu
	Aidan Deacon	agdeacon@umich.edu		Darren Cleeman	dcleeman@umich.edu
Embedded Systems	Brinda Kapani*	bkapani@umich.edu		Pranav Pabba	ppabba@umich.edu
	Katherine Shih**	katshih@umich.edu		Julian Whittaker	juliwhit@umich.edu
	Eric Bi	ericbi@umich.edu		Kris Terdprisant	kristerd@umich.edu
	Sneha Das	snehadas@umich.edu		Anna Novak	annanova@umich.edu
	Ruey Day	rueyday@umich.edu	**Computer Vision**	Matthew Gawthrop*	mgawthro@umich.edu
	Liam Domegan	ldomegan@umich.edu		Ryan Beaudoin**	rbeaudoi@umich.edu
	Eric Barbieri	ericbarb@umich.edu		Maya Echtinaw	mayaecht@umich.edu
	Layth Abdelkarim	laythabd@umich.edu		Pranav Mallela	pmallela@umich.edu
	Yi Keen Lim	yklim@umich.edu		Amanda Juvera	ajuvera@umich.edu
	Bany Huang	bany@umich.edu		Edison Zhou	edisonz@umich.edu

1. INTRODUCTION

The Autonomous Robotic Vehicle team at the University of Michigan is a student-led project team with a focus on building a competitive robot for the IGVC. We prioritize innovation and the development of our members while contributing to the progress of the broader robotics community.

1.1. *Team Organization*

ARV is entirely student-run, led by an executive board that generally guides the team towards our mission statement and oversees our five distinct subteams. Platform designs, builds, and maintains the vehicle chassis. Computer Vision develops machine learning algorithms for obstacle detection and avoidance. Navigation and Sensors directs vehicle motion based on path planning and sensor fusion. Embedded Systems controls vehicle hardware and execution of software commands. Business secures funding and establishes relationships with corporate sponsors.

1.2. *Design Process and Assumptions*

We follow a cycle of ideation, design, implementation, and testing throughout the duration of the year. The driving forces behind this year's vehicle were: 1) improve upon the reliability and performance of last year's design, 2) develop custom infrastructure for complex algorithms as part of our first entry into Self Drive, and 3) implement robust testing and analysis mechanisms for hardware and software. We develop modularly to allow for rapid prototyping and progressive integration. Our design is intended to operate in real-world environments in complex driving scenarios, so we strive to avoid assumptions about ideal conditions and seamless sim to real transitions.

2. SYSTEM ARCHITECTURE

2.1. *Major Software Components*

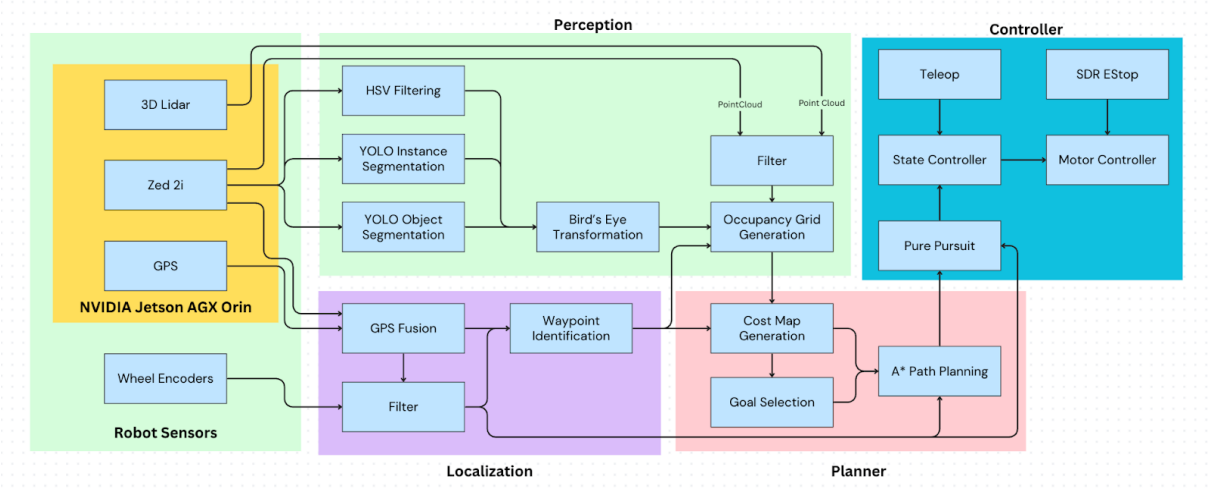


Figure 1. Major software components

Robot Sensors: The robot utilizes a suite of sensors, including 3D Lidar, Zed 2i stereo camera, GPS, and wheel encoders, all interfaced via an NVIDIA Jetson AGX Orin and a Lenovo Legion Laptop for distributed processing.

Perception: This module processes raw sensor data to understand the environment. Techniques like HSV filtering, YOLO-based object and instance segmentation, and bird's-eye transformation are applied to camera data to generate an occupancy grid, while LiDAR data is filtered to the camera overlay for use in No Man's Land.

Localization: GPS data is fused with encoder and IMU data and filtered to estimate the robot's position. Waypoint identification uses this localization to support path planning.

Planner: The planning module receives inputs from both the perception and localization modules. It generates a cost map of the environment, selects goals, and computes paths using the A* algorithm.

Controller: The controller executes the planned path using a Pure Pursuit algorithm for motion tracking. It handles teleoperation inputs, emergency stop controls (SDR E-stop), and sends commands to the motor through the state and motor controllers.

2.2. *Major Power & Electrical Components*

As shown in 2, our electrical system is managed by a centralized computer, which sends velocity commands to motor controllers and receives encoder feedback via USB. It communicates with the remote E-stop using a software-defined radio and with the safety light over a serial connection. The motors are powered by a 26 V rail, an additional 12 V rail is provided by a voltage regulator.

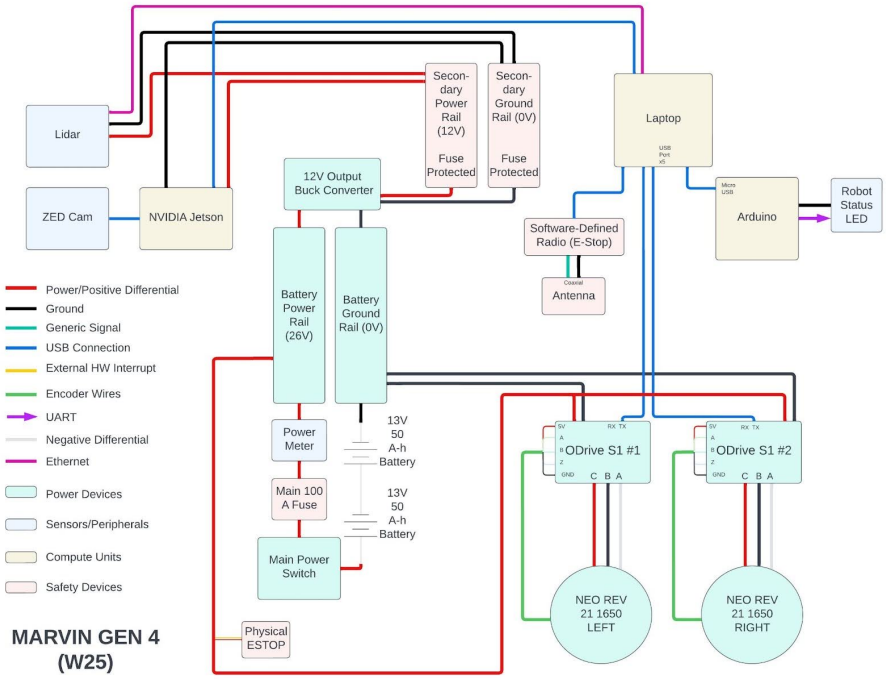


Figure 2. Electronics and Power Diagram

2.3. *Safety Devices*

Safe operation of mARVin 2.0 is a key part of the design. The main power from the batteries is supplied through a visible, accessible circuit breaker mounted on the outside of the robot. A 100 A fuse protects the power rails and downstream devices from catastrophic errors. A fuse corresponding to the current consumption of each peripheral device protects the 12 V power rail. To ensure that no safety issues arise during a run, a remote software-defined E-stop is attached to the laptop, and a physical E-stop is connected to the main power circuit on the robot.

3. EFFECTIVE INNOVATIONS

3.1. *Software Architecture*

Our software architecture is focused on efficiency, modularity, adaptability, and performance. We accomplish this by building custom debugging tools and system-specific frameworks, developing self-supervised machine learning pipelines to address data constraints, and making hardware decisions that directly support our system while eliminating redundant computation.

3.2. *Dashboard GUI*

The dashboard allows for remote monitoring and control of the robot. The dashboard is split into 2 main components: Controller Mode and Live Feed. The GUI status light lets the user know the robot is ready to run. The controller mode has 4 options: Disabled, Autonomous, Teleop, and Start self drive. Teleop mode is for the convenience of testing within the team. Autonomous mode would be used in Autonav to start the code for GPS waypoint and laneline following. The start-self drive option launch sthe self-drive software. The "EMERGENCY STOP" is a manual button to stop all action of the robot; this is an additional layer of E-stop that is separate from on-robot E-stop and doesn't interfere with the wireless E-stop. The Live Feed component shows the real-time camera feed to the user and real time updated data such as IMU data (roll, pitch, and yaw) and GPS location (longitude, latitude, altitude).

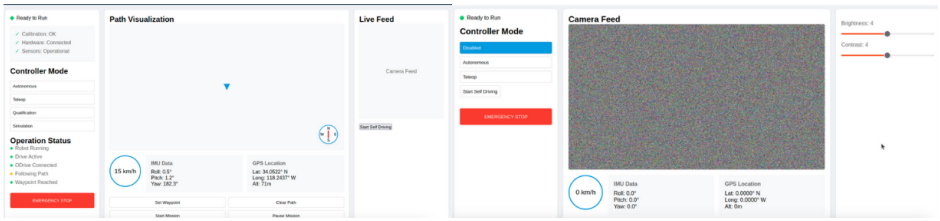


Figure 3. Side by Side of Two Dashboard Views

3.2.1. *Custom Navigation Infrastructure*

In prior years, the team's navigation system was built upon Nav2, a common Robot Operating System 2 (ROS2) based navigation framework. However, Nav2's rigid plugin based architecture prevented us from integrating custom software that goes beyond the scope of the predefined-Nav2. For example, Nav2 does not infrastructure for our custom goal selection software. Additionally, Nav2 is built upon SLAM-toolbox, a localization package that only supports LaserScan based input (i.e., LiDAR) while our perception when lane-following is entirely camera based. We have chosen to forgo Nav2 and build a custom navigation interface called $nav\_infrastructure$, that provides a preformant and easily adaptable alternative to Nav2. This software infrastructure allows us to process and send data throughout the software system with minimal overhead and a high degree of flexibility.

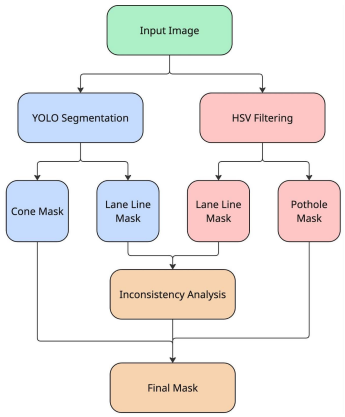
3.2.2. *Perception System*

This year, our team moved to a modular perception system that combines advanced machine learning algorithms with classical computer vision techniques. We choose to innovate our approach to introducing state-of-the-art methods while considering input from known technologies. We use a custom trained You Only Look Once (YOLO) machine learning segmentation model for both lane lines and cones. To add redundancy we also use HSV filtering to gather a second set of masks for lane lines and potholes. Finally, we constantly scan for inconsistencies in both HSV and YOLO lane line masks and choose to use one or both, effectively adding a safety net for both models in their respective areas of struggle. The logic flow and impact of this innovation can be found in 4a and 4b.

Mechanically, the Zed 2i camera mount is the highest vehicle element and facilitates the vision transforms necessary to calculate obstacle distances. As such, the mount must allow for fine adjustments of camera

angle before vehicle operation and stability during operation. A previous iteration of this mount involved a 3D printed planetary gearbox and a crank, pictured in 5a, which provided stability and adjustability, but also created camera blind spots and many failure points due to complexity of the gearbox.

These issues were resolved through design and 3D printing of an innovative rack-and-pinion tilt mechanism, pictured in 5b. This simpler mechanism uses a rack, a straight set of gear teeth controllable by a circular pinion gear, to angle and then fix the camera holder using a pivot and movable shaft collars. This maintains adjustability and stability of the camera while eliminating blind spots and greatly reducing failure points.

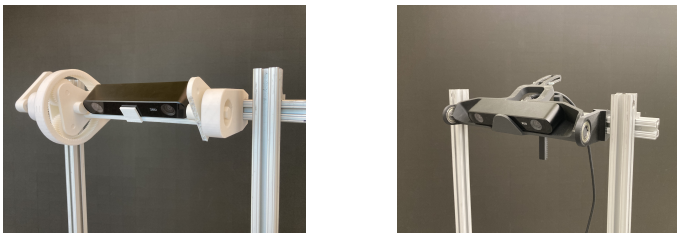


(a) Logic Flow for Object Perception

Method		Accuracy	Description
YOLO	**HSV Filtering**		
✓		89%	High accuracy, but can create non-existent lane lines.
	✓	94%	Lower accuracy but more robust, and reduces comuptational complexity with simple classical approach.
✓	✓	98%+	Fusion improves accuracy significantly.

(b) Comparison of YOLO and HSV Filtering Methods

Figure 4. Hybrid Perception Pipeline Performance



(a) Planetary Gearbox Mount (b) Rack-and-Pinion Mount

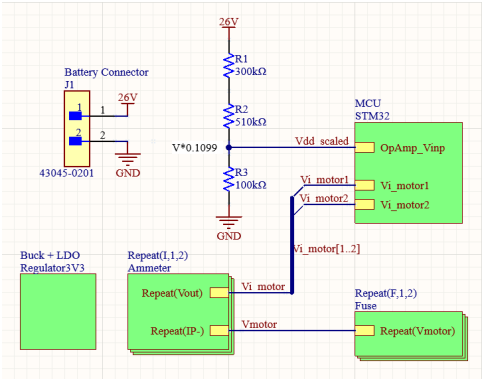
Figure 5. Old and New Zed 2i Camera Mount Tilt Mechanisms

3.3. *Testing and Debugging*

To understand vehicle performance, support integration efforts, and identify points of failure, we developed robust methods to track and display various metrics in different operational modes.

3.3.1. *Embedded Test Bench & Power Distribution Panel PCB*

To streamline testing and prevent potential issues before implementing changes on the robot, we developed a dedicated testbench for the robot. Our setup features an ODrive motor controller connected to a NEO motor, which enables us to connect the ODrive to a computer for programming, testing, and debugging. This setup has been vital for one of our biggest changes this year: removing the ODrives' dependence on the Nucleo STM32 MicroROS framework. Now, instead of running ODrives through the STM32, which then communicates with the computer via USB and MicroROS, we control the ODrive directly over USB using a ROS2 integrated Python script. This change allows us to take full advantage of the computer's processing power for handling ROS2 messages, allowing us to publish encoder readings at 50 Hz without noticeable performance impact, whereas the STM32 MicroROS setup introduced latency in motor control when publishing at just over 10 Hz. It also avoids blocking other STM32 system functionalities, as running MicroROS under Real-Time Operating System (RTOS) triggers constant interrupts, which previously impacted the



(a) Excerpt from PCB Schematic

Color	State
Flashing Blue	Autonomous
Solid Blue	Teleop
Red	ODrive Error
Green	Quals Mode
Cyan	Ramp
Yellow	Cones Area
Magenta	No Man's Land

(b) LED Color State Indicators

Figure 6. Embedded Stack Innovations

performance of other code on the STM32. Additionally, the Python API for the ODrive offers more features and flexibility, and using USB provides more robust signals compared to running UART lines over distance.

To improve power reliability and diagnostics, we are currently developing a power distribution panel with safety features, an integrated display, and permanent storage. This panel will be placed between the batteries and the motor controllers, replacing the current 100 A fuse for the motor system. The PCB will implement independent fuses for each motor system with associated fuse-blown indicators, a toggleable OLED display to show a live plot of current and voltage measurements, and a MicroSD card to store these measurements. An STM32L4 low-power microcontroller will control the measurement functions. An excerpt from the PCB schematic is shown in 6a.

3.3.2. *State Indication: LEDs*

To help with debugging, we use an RGB LED strip with 30 individually addressable LEDs. We use different colors to indicate different states of the robot and errors. The LED strip transitions from solid blue to flashing blue when the robot enters autonomous mode. The other colors are helpful for troubleshooting errors depending on the robot's state. A summary of the colors and corresponding states can be found in 6b. The LED strip is connected to an Arduino MEGA, which contains a script that changes the color of the LEDS based on the ROS message received.

4. MECHANICAL DESIGN

4.1. *Design Overview*

mARVin 2.0 was designed and manufactured by the Platform subteam. The vehicle measures 30.5 inches in width, 44.5 inches in length, and 67 inches in height. The design process involved making a detailed CAD in SolidWorks (see 7a), prototyping key mechanical systems, and fabricating components.

Drill presses, bandsaws, and waterjets were used to fabricate the aluminum square tubing, brackets, and gearbox casings that make up the vehicle frame while 3D printers were used to print the camera mount and weatherproofing parts. Major parts not manufactured by the team are the 8-inch wheels and gearbox gears.

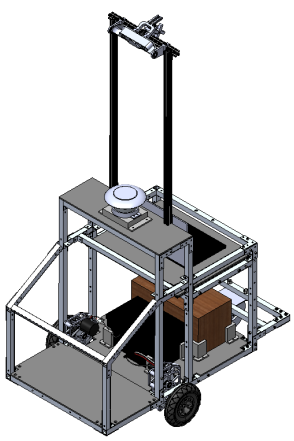
4.2. *Significant Mechanical Components*

The drivetrain consists of two custom-designed single-motor gearboxes, seen in 7b, made from team-fabricated aluminum housings and purchased gears. Each gearbox transmits a 98:3 torque increase from the Neo brushless motors, a ratio determined based on the motor torque-speed curves and the calculated output torque necessary to traverse the ramp. The output shaft of each gearbox connects below the vehicle frame to 8-inch concrete-specific wheels in order to provide the necessary height to drive over the peak of the ramp.

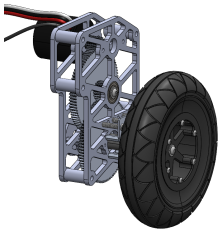
The other significant mechanical component is the Zed 2i rack-and-pinion camera mount, pictured in 7c. Details for this component have been introduced as an innovation in Section 3.2.2.

4.3. *Frame, Structure, and Suspension*

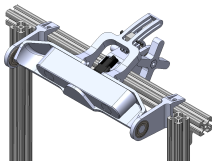
The vehicle structure, pictured in 7a, has sections for electronics, loads, and laptop accessibility. The back of the vehicle houses the batteries and payload to maintain the center of mass behind the wheels, thus



(a) Full mARVin 2.0 CAD



(b) Gearbox CAD



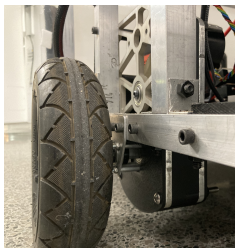
(c) Camera Mount CAD

Figure 7. CAD Models of Major Mechanical Components

mitigating vehicle tipping. The payload is also secured behind the batteries for ease of access. The laptop is placed on a shelf above the loads and at seating height for user convenience. The front of the vehicle is used to house the embedded systems due to proximity to the batteries and two Neo brushless motors powering the driving gearboxes. The roof of the front section is also sloped to avoid obscuring the Zed 2i camera's field-of-view. A suspension system was not necessary for mARVin 2.0 due to stability under low vehicle speed and relative smoothness of the asphalt course.

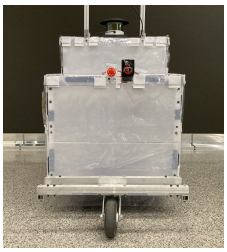
4.4. *Weather Proofing*

The dry layer of weatherproofing consists of thin polyethylene sheets cut to fit the vehicle profile, preventing large debris from entering the interior. 3D printed coverings were also designed to directly interface with the bottoms of each gearbox to keep debris out of the gears, pictured in 8a. The wet layer of weatherproofing is a clear vinyl covering, shown in 8b and 8c, that protects against light precipitation. This cover serves to seal gaps in the polyethylene cover while also providing access to the power and manual stop.



(a) Gearbox Cover

(b) Front Vinyl



(c) Back Vinyl

Figure 8. Dry and Wet Vehicle Weatherproofing Measures

5. ELECTRONIC AND POWER DESIGN

5.1. *Significant Power and Electronic Components*

The electronic and power design was implemented by the Embedded Systems subteam. The team incorporated two Odrive S1s and a STM32 Nucleo board to control various parts of the robot. In this year's embedded system design, we focused on simplifying the previous year's setup to enhance efficiency and reliability. This stripped-down approach provided a solid foundation, and we are now incrementally rebuilding with improvements, such as a custom PCB power distribution panel and the exploration of a new motor.

5.2. *Power Distribution System*

The system is powered by two 13 V batteries connected in series, meeting the minimum 12 V requirement for each ODrive S1 motor controller. Additionally, we implemented two separate power rails, one delivering 26 V and another 12 V for powering the LiDAR. Each ODrive S1 controller is wired to a NEO brushless motor, which has built in Hall sensors for measuring odometry.

Empirical power consumption of the robot is around 30 W at idle and 120 W when operating under load. From aforementioned observations, it is derived that the robot will have a battery life of around 10 hours. The batteries will recharge to full within 2 hours.

5.3. *Electronic Suite*

A wide range of electronics are deployed on the vehicle, including computers, a GNSS, and motors. We use the TP-Link AC1200 Wireless Gigabit Access Point to facilitate communication between the robot's ROS messages and an external computer. The access point is connected to the ethernet switch, which is connected to the laptop on the robot, and allows an external device to receive the laptop's ROS data. This is essential for the self-drive qualification functional tests to view the robot's camera feed from a separate device.

5.4. *Mechanical and Wireless E-Stop*

A critical refinement in our wiring ensures that when either E-stop is pressed, only the ODrives shut off while the software remains operational. This significantly reduces software recalibration time and improves overall system efficiency. The mechanical E-stop directly cuts power to the motor system, while the wireless software-defined radio (SDR) sends a 0 m/s velocity command to the motor controllers that overrides all other velocity commands but does not power off the software components. Pressing the mechanical and wireless E-stop buttons again will allow the robot to resume where it left off. The remote E-stop has a measured operational range of 100 ft.

5.5. *PID Tuning*

To achieve the desired speed and acceleration while keeping the current draw within safe limits, we leverage the ODrive's PI control loop, which regulates torque based on a target speed. We use the ODrive GUI for tuning these parameters, as it allows us to visualize both the target and actual rotational speeds (in RPS). Tuning begins with a low proportional gain, which we gradually increase until we achieve a small steady-state error. During this process, we carefully monitor current draw, since a higher proportional gain leads to a faster initial response, and consequently, to a higher peak current. The GUI helps us track real-time current consumption to ensure the ODrive draws less than 10 A, even during peak acceleration from a standstill when given a direct velocity command corresponding to 4 mph on the wheels. To further safeguard against excessive current draw, we configured both the soft and hard current limits on the ODrive. These parameters ensure that the controller will shut off a motor if its current exceeds 40% of the main fuse's rated capacity.

Once the proportional gain in our PI controller is tuned for safe operation and minimal steady-state error, we incrementally increase the integral gain to eliminate steady-state error. Meanwhile, we aim to avoid any overshoot or oscillation and maintain a smooth, non-aggressive initial response.

6. SOFTWARE SYSTEMS

6.1. *Perception Pipeline*

6.1.1. *Multi-Sensor Perception Strategy*

Our perception sensors include a ZED2i camera and a Veloydne VLP-16 LiDAR sensor. Due to the sophistication of our Computer Vision algorithms and the range of environmental conditions we can expect in the competition environment, we are able to accurately detect all necessary obstacles, lane lines, potholes, and traffic cones with the camera alone. We suppress the LiDAR detections in environments segmented by continuous lanes to prevent redundant computation. In cases with only sparse 3D obstacles (no-man's land), we use the LiDAR to directly produce an occupancy grid of the scene.

6.1.2. *Sensor Processing and Fusion*

The data preprocessing and sensor fusion framework includes automation of the initialization and configuration of the uBlox GPS, ZED camera, and LiDAR through launch scripts, ensuring proper setup for GNSS fusion. When lane following, the GPS and ZED launch scripts uses ROS nodes to fuse GNSS data with positional tracking information to obtain a precise robot position. Another node coverts latitude/longitude to robot map coordinates. This is because the robot map coordinates given by the computer vision team are locally based, this conversion transforms this data to a world coordinate frame. In No Man's Land, a LiDAR launch script retrieves PointCloud2 data from the LiDAR, contributing to obstacle detection and environmental mapping for dense point clouds, which is then integrated with navigation for mapping and path planning via an occupancy grid of the scene.

6.1.3. *Lane and Drivable Area Detection*

The perception flow begins with input from a ZED stereo camera. Based on the part of each competition the robot is in, we select the correct combination of segmentation techniques to give us the best results (details in Sections 9A and 9B). All computation for the computer vision stack runs onboard the NVIDIA Jetson Orin, where we use TensorRT to accelerate inference of our ML models and ensure real-time performance. The output from the segmentation models is then transformed using a Bird's Eye View (BEV) transformation matrix calculated from the camera's intrinsics (optical center and focal length) and extrinsics (height and angle). This creates a top-down occupancy grid where each pixel represents a fixed real-world distance (0.05 meters). The occupancy grid is then published using ROS, where it can be used for goal selection and path planning on a separate onboard laptop (except with self-drive, more info 9B).

6.2. *Goal Selection and World Representation*

6.2.1. *Transition away from SLAM*

This year's we avoid Simultaneous Localization and Mapping (SLAM) due to significant integration challenges and redundant computation. Our computer-vision is currently able to detect lane-lines, potholes, and cones within a range of 2.5m (after transformations). Since the area outside of this range is undetectable initial testing showed that all of the SLAM computation, as well as the path-planning computation, done outside of the range of 10m was subsequently recalculated once the robot progressed and saw more obstacles. Therefore, a significant amount of computation was wasted with this method.

Given these inefficiencies mARVin 2.0 switches to an entirely local world representation, where localization and planning is done entirely within the cameras current frame. Using our odometry we transform the path from the camera's point of view to real world coordinates and use that for motion control. This transform gives orientation relative to the given GPS goals, and the goal selection algorithm and planning is biased to choosing goals/paths towards the goals. This approach significantly decreases the search space when path planning, and eliminates computation outside the camera range.

Last year, we primarily implemented localization by using the IMU through Extended Kalman Filters (EKF). Last year, we ran IMU EKF on the BNO055 sensor; however, the sensor resolution is at 0.488 mg, whereas for the IMU sensor inside the ZED2i camera has a resolution of 0.244 mg. With Robot Localization we would be able to fuse more precise sensors with the ZED2i leading to our decision to retrieve odometry data via visual odometry.

6.2.2. *Dynamic Waypoint Selection and Goal Adjustment*

The dynamic goal selection algorithm provides intermediate goals between the given GPS waypoints between the lane lines and obstacles. The output of the goal selection algorithm is to choose an optimal point within each frame of computer vision to guide the path plan.

First, an inflation layer is applied to the occupancy grid provided by CV to ensure a safe distance is maintained from any obstacles. This also fills in any uncertain points that may exist in the CV output. To save computation, this layer is used both to determine a goal and also in path planning.

The second component is the algorithm responsible for choosing the point to path plan towards. The considerations are maximizing clearance of obstacles and maximizing progress towards the GPS waypoint within the lanes.

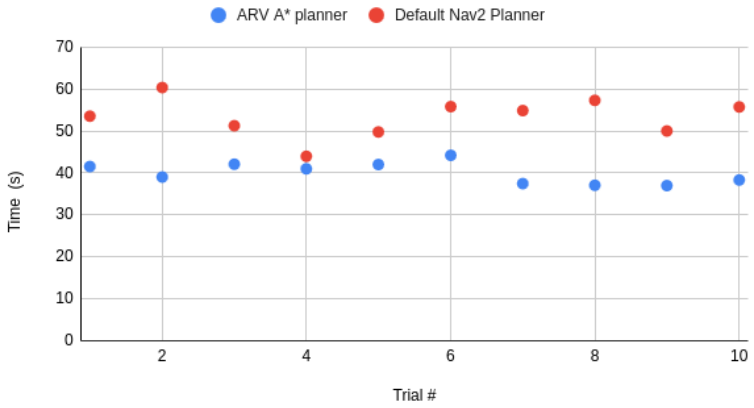


Figure 9. ARV A* Planner vs Nav2 Default Planner

Distance from the robot and distance from the edges are also considered to incentivize further distances traveled. We run this algorithm on the points only within the drivable area detected from the lane lines and obstacles. The best point calculated becomes the new goal and is forwarded to the path planner.

6.2.3. *Operating Modes and Mode Switching*

The robot's operating modes are: Autonomous, Teleop, Ramp No Man's Land. If the current mode is Autonomous control input is from the path planner. In Ramp mode the path is straighted in order to stay on the ramp. The states are changed when the robot's GPS is within range of the competition's given GPS coordinates. If no GPS coordinate is given, the software will automatically switch to a lane-following mode, where the goal that is selected is soley based on making progress through the drivable area. In teleop mode control is taken from a human driver, this is triggered by a button on the controller.

6.2.4. *Influence of Past Trajectories and Obstacles*

As mentioned, our current design uses localized occupancy grid to increase computation speed. Past trajectories are used for normalization in the current frame such that the planner knows what angle the camera frame is relative to the starting position and GPS waypoints.

6.3. *Path Planning*

Using our custom navigation infrastructure, the custom A* path planning algorithm efficiently computes the optimal trajectory (path) to the goal selected by the goal selection algorithm. The custom infrastructure creates a pipeline for a portion of the data calculated by the goal selection algorithm (i.e obstacle inflation) to be used by A* to save computation. The algorithm uses a custom A* implementation, using the euclidean distance as a heuristic function (which in our testing had identical or faster execution times compared to other common heuristics such as Manhattan distance). The custom navigataion infrastructure, excludes unneeded path planning processes such as Nav2's costmap layering. These strategies ensure both speed and resource efficiency. We see about a 25% speed improvement over the default planner Nav2 provides in our simulation testing.

6.4. *Path Execution*

6.4.1. *Trajectory Execution and Control*

An adaptive pure pursuit algorithm is used for trajectory execution. In this algorithm a close point on the path is chosen, and the linear and angular velocities to reach that point (called the lookahead point) is calculated. A spline smoothing algorithm is used to smooth the given trajectory. Linear interpolation between the points on the path gives more precise lookahead. The range of points is limited to be a tunable distance away. Additionally, points along the path that are too far back from the current lookahead point are rejected. This results in a smooth path execution.

6.4.2. *Real-Time Replanning & Path Failure Handling*

Error detection is done both in goal selection and in path planning to be certain of safe traversal. In goal selection, if the cost of every cell within the current field of view is too high (detected no free space), the system will wait a few seconds and retry with a new camera frame in case of a temporary camera error. Path planning will do the same thing if a safe path is not found. If either algorithm fails a second time, the robot will attempt to reverse a set amount of distance (0.25 m) and retry. The planner will also re-plan if the pure pursuit algorithm fails in the middle of trajectory following, or if no lookahead points are found. The timing of the system is tuned such that there is the minimum amount of time between path planning while also allowing the computer vision system to be up to date. Logs are generated and separated by sub-systems for easy debugging.

7. CYBERSECURITY ANALYSIS USING RMF

To address cybersecurity in the custom Remote E-stop system, we follow the NIST Risk Management Framework (RMF) by first identifying and modeling potential threats. The top threats are signal spoofing, which mimics a real signal, and jamming, which prevents a real signal from being identified. These are assessed as high-impact threats due to the safety-critical nature of the E-stop, where a malicious or missed signal could result in physical harm or equipment damage. Our current implementation enhances cybersecurity by using an SDR dongle to receive wireless button press signals, which are processed through a low-pass filter to reduce noise, followed by matched filtering and peak detection to reliably identify valid transmissions. This multi-stage signal processing pipeline shown in 10 improves the robustness of signal recognition and helps filter out random noise or unintentional interference, reducing the likelihood of false activations.

In future iterations, we plan to significantly strengthen the E-stop security by implementing a custom transmitter that sends digitally encoded messages using a proprietary modulation scheme. These messages will include built-in error detection such as checksum to prevent spoofing and tampering. Additionally, we will incorporate frequency hopping to mitigate the risk of signal jamming and eavesdropping by dynamically changing the transmission frequency according to a shared hopping sequence. Together, these enhancements will create multiple layers of protection that improve the system's security by making it harder to intercept, fake, or block signals. We'll test these defenses thoroughly using simulated attacks to make sure the system stays reliable and secure.

To secure our robot dashboard, we enabled HTTPS encryption, validated all HTTP and WebSocket inputs, and required authentication tokens on Socket.IO connections to prevent unauthorized control and data exposure.

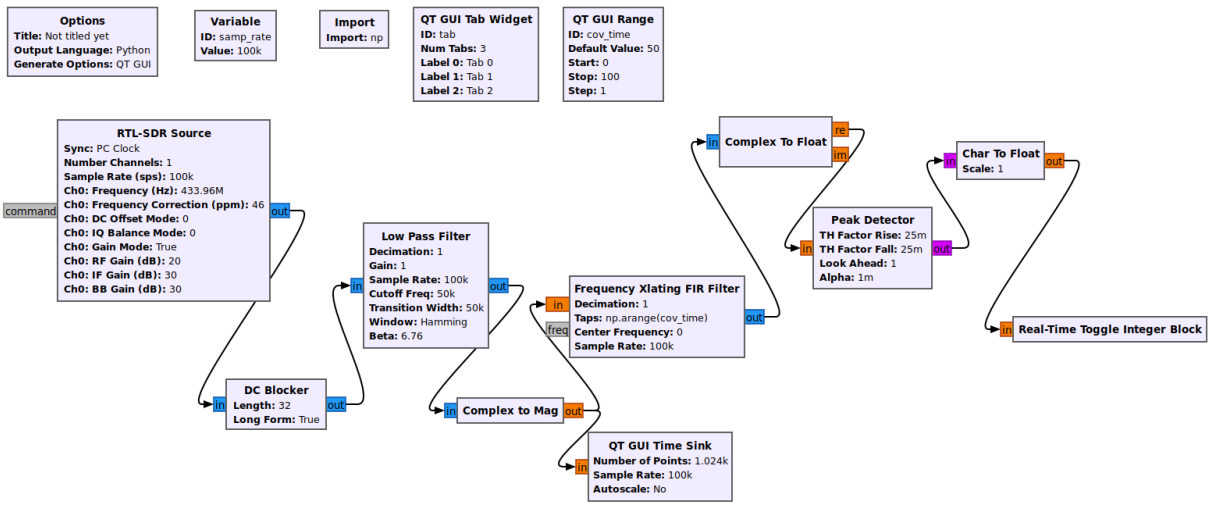


Figure 10. SDR E-Stop Flowgraph in GNU Radio

8. COMPLETE VEHICLE ANALYSIS

8.1. *Lessons Learned*

One lesson learned during system integration is the importance of continuously deploying software onto hardware. For example, while testing the local planner, we found its performance is heavily affected by real-world factors such as friction, wheel resistance, and actuator response time. These factors required several iterations of testing and adjustment to achieve reliable path-following.

8.2. *Safety, Reliability, and Durability Considerations*

To ensure system reliability and durability, we designed the electrical architecture to be highly modular. Each subsystem, including safety light, E-stop and motor controller, can be replaced by a different implementation without modifying other parts of the system. This modularity simplifies repairs and upgrades during development and competition. To ensure controller robustness, we tested the robot repeatedly on different road surfaces. The E-stop was tested both indoor and outdoor under various conditions. One issue we encountered was a delay in remote E-stop activation caused by GNU Radio processing speed. To resolve this, we iterated on the embedded Python block within GNU Radio, optimizing the algorithm efficiency to ensure fast and reliable response.

8.3. *Key Hardware Failures and Strategies*

Hardware failure points include exceeding user-specified current or voltage thresholds, which can trigger ODrive S1 errors and shut down the motors. Blown fuses may cut power to peripheral electronics, and accidental short circuits can damage electrical components. To mitigate these risks, the ODrive S1s are tuned to operate within safe current limits, and brake resistors are configured to handle voltage fluctuations that might otherwise cause errors. LED indicators on the power rails allow for quick identification of blown fuses. To prevent short circuits, all wiring is separated, securely tied in place, and protected with heat shrink tubing at terminal connections.

8.4. *Predicted Performance*

Speed: Combined PI tuning allows our robot to accelerate to over 1.5 mph in under 0.5 s while achieving a top speed of 2 m/s (4.47 mph).

Ramp Climbing Ability: Uphill performance met expectations because of integral control. Downhill speed was slightly aggressive and can be improved with software compensation.

Reaction Times: E-stop response times met expectations: physical E-stops reacted in under 100 ms and remote E-stops in under 300 ms.

GPS for Waypoint Navigation and Localization: We expect the maximum total error possible for our GPS to be 1.5 m, but in our testing, we see a maximum error of 0.5 m in about 95% of cases.

Battery Life: Battery is expected to last 10 hours; tests showed consistent operation for at least 8 hours under load.

Distance at Which Obstacles are Detected: Obstacles are detected within the 120 degree wide angle field of view of the camera. This leads to obstacles being detected anywhere from 0.3-20 m in front of the vehicle. However, only the obstacles within the first 2.5 m are kept after the BEV transform and used in navigation.

Vehicle Dealing With Complex Obstacles: We have built our architecture to allow for waypoints to be defined both close and far away from the robot based on the situation the robot is in. Throughout testing, this has shown to help the robot to move in tight spaces and gain different angles around close obstacles.

Identifying and Addressing Failure Points and Modes: When testing in simulation, the navigation system identified all cases where the given field was not traversal-able (i.e there was no viable path to

goal). With multiple failure checks (in goal selection, path planing, path execution), at least one of these checks was always triggered. In real world scenarios, tuning of the inflation and cost threshold parameters will affect how well we identify failure points. In addressing failure modes, our recovery logic mostly consists of backtracking and waiting, this would prove effective for errors such as temporary vision blockages, but it may fail in more catastrophic scenarios.

Accuracy of Arrival at Navigation Waypoints: Navigation way-points are expected to be reached about 90% of the time. Way points were only not reached in specific circumstances including: obstacles in the blind spot of the robot right under the nose, less than 0.1 m of clearance between the robot and the obstacles, significant error in GPS. When reaching a waypoint we expect to be less than 0.5 m from the actual waypoint.

8.5. *Software Version Control and Testing*

Effective testing and debugging are crucial for developing a reliable autonomous navigation system. Our team employs a combination of simulation, hardware-in-the-loop testing, and specialized visualization tools to analyze system performance and rapidly identify issues.

With all the changes we made to the computer vision stack this year, testing and verification became a major focus as we wanted to make sure every design decision was backed by data. One of the most significant updates was adding redundancy by combining HSV filtering and YOLO segmentation as two independent ways to mask lane lines. To get started, we built a custom HSV Python library that allowed us to tune parameters and visualize segmentation in real time. This made it easy to quickly adjust settings, run tests, and directly compare HSV filtering against YOLO. We measured the accuracy of each approach using recordings from our own test courses, which we set up in campus parking lots with white tape. To ensure consistency in our evaluations, we designed a dedicated testing suite that was only ever used for validation. The suite included footage from different cameras and a variety of weather and pavement conditions to test robustness. Each frame was hand-labeled to serve as ground truth, and we used a simple pixel-to-pixel accuracy metric to evaluate performance. Our results showed that combining the HSV and YOLO masks led to a 4% increase in overall accuracy compared to using YOLO alone. That improvement validated our decision to integrate both methods and confirmed the value of adding classical computer vision techniques as a backup to ML-based segmentation.

Comprehensive edge test cases were developed to validate the robustness of the planning implementation across various dynamic scenarios. Our implementation successfully passed all test cases with optimal paths. Path visualization, using the tool provided by nav-visualization (see below), confirmed accurate navigation and demonstrated real-time responsiveness of the planner in complex environments.

8.6. *Simulation*

A critical tool in the team's testing and debugging workflow is the nav visualization package, developed to provide graphical insights into the software stack. This suite allows developers to visually monitor and validate the core components of the navigation system. Costmap Representation: The visualizer loads and displays costmaps used by the planning algorithms. Grids can be from real world ROS data or from custom grids drawn by developers using the interactive feature. It renders free space, obstacles (e.g., detected lane lines, cones from the perception pipeline), and intermediate cost areas. This allows for immediate verification and debugging.

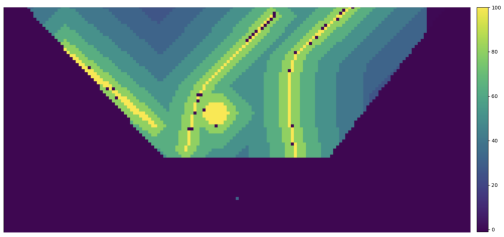
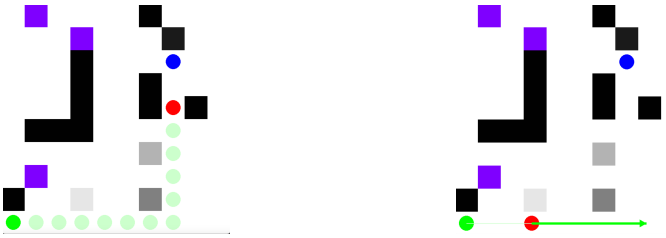


Figure 11. Cost Heatmap for Simulated CV Output

Path Planning Visualization: This component focuses on global planning. It interfaces with the nav infrastructure and displays the calculated global path (e.g., the output from our A* planner) overlaid onto the costmap. This enables developers to validate the planner's logic and assess path optimality and feasibility before execution. This also allows us to observe how the planner handles different environmental features and constraints. This component also simulates and displays the real-time execution of the path. This is done by subscribing to the path-execution code (e.g., Adapative Pure Pursuit) and simualting the robot's continuous motion (x, y, theta) by integrating velocity commands over small time steps. The robot's current position, orientation, and its historical trajectory are displayed directly on the costmap. This real-time view is invaluable for diagnosing controller performance, identifying path-following errors (like oscillations or deviations), and debugging obstacle-avoidance maneuvers.

Utility: The package also includes a custom interactive tools for quickly creating or modifying costmap files, facilitating the setup of specific test scenarios.

Together, these visualization tools form an essential part of our development cycle, enabling rapid debugging and testing against different environmental conditions. This software is designed to be quickly and easily installed cross-platform, especially easy for noivce user. The image below shows an example output from a simulation visualizing the path generated by the A* planner.



(a) Path (b) Local

Figure 12. Planner Visualizations

9. UNIQUE AUTONAV FEATURES

During the autonav competition we use both HSV filter and a YOLO model to mitigate the effects of hallucinations (YOLO downfall) and shadows (HSV downfall) on the robot. Having found that our YOLO model has much more consistent accuracy with the traffic barrels we are detecting those solely with YOLO.

A different strategy is used in "No Man's Land," where ML inference is disabled entirely (given bad results in the sparse detection environment), and the system relies solely on HSV filtering. Before the ramp comes into view, the robot detects cones to guide itself forward. Once lane lines appear, the system artificially projects them into the occupancy grid, aligning the robot properly with the ramp. Path planning and path execution remain mostly the same, the main difference is our aforementioned goal selection algorithm is done only in Autonav. In self drive we use a uniqnue technique to define waypoints. The dashboard also has separate buttons to launch the Autonav or self-drive ROS nodes with their respective parameters.

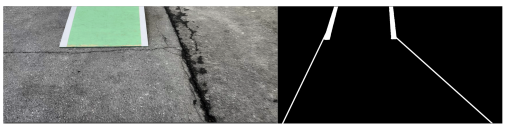


Figure 13. Artificial Lanes for Ramp Alignment

10. UNIQUE SELF DRIVE FEATURES

By designing the software stack with modular publishers that perform one task at a time, the architecture allows us to adjust the flow of information based on the competitions. The largest shift being that we first define waypoints within the ZED camera frame to utilize the wide angle, partially helpful with intersection maneuvers. Then the waypoints are transformed using the same perspective transform we apply to our occupancy grid.

We detect the yellow and white lane lines using HSV filtering and use a custom trained YOLO model for stop signs and obstacle classes. Throughout the course we take full advantage of the yellow center dashed lines to know which lane we are in as well as where to align the robot on a turn.

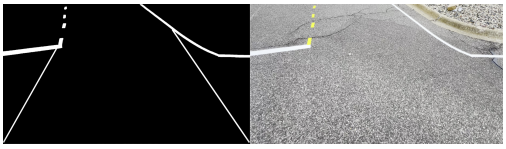


Figure 14. Artificial Lanes for Turns

Additionally, we combat fake stop signs by filtering and reading the text with a lightweight optical character recognition plugin to verify we are performing the correct actions throughout the course.

11. INITIAL PERFORMANCE ASSESSMENTS

Individual hardware components on the vehicle such as the motor controller, wheel encoders, GPS, IMU, Camera, wireless and physical E-stops have been tested and work as expected. We utilized a tele-op controller to test the motor control, and RViz to visualize the sensor data collected.

Software components have also been tested individually to ensure each produces the correct output. The computer vision pipeline, which takes frames from the onboard camera and produces a 2-D occupancy grid of obstacles, accurately identifies obstacles within 1 inch of error.

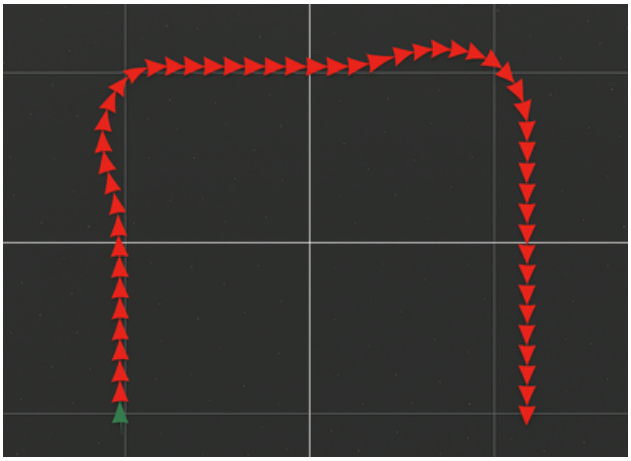


Figure 15. Path Planning and Execution Test

In a simulated course using our visualization software, with a final waypoint approximately 50 meters away from the starting position we saw an average of 39.978 seconds completion time. After tuning, all simulated runs reached the goal waypoint within 0.5 meters.

The navigation stack, including velocity controller, waypoint generation, and path planner, has been tested in both simulation and real-world conditions using ROS bags and live sensor data. During real-world tests, the controller can follow a given path with less than 10 cm of deviation at the largest deviation point, as shown in 15. The path planner reliably generates feasible and efficient paths, with a goal reach success rate above 84%.