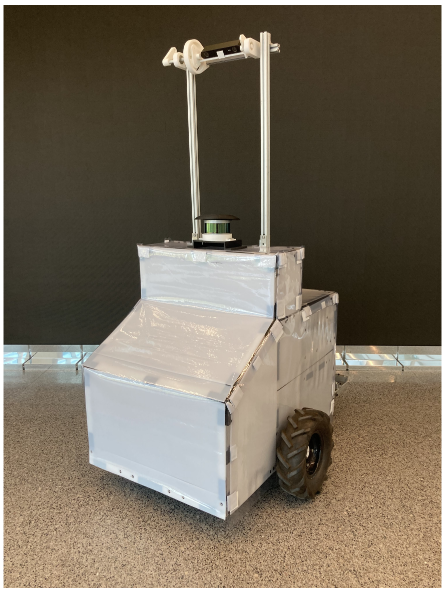
Autonomous Robotic Vehicle
University of Michigan



Submitted: May 15th, 2024
Team Captain: David Welch (dswelch@umich.edu)
Faculty Advisors: Xiaoxiao Du (xiaodu@umich.edu), Damen Provost (provostd@umich.edu)
Statement of Integrity: The design and engineering of the vehicle by ARV has been significant and equivalent to what might be awarded credit in a senior design course.

Xiaoxiao Du

Team Roster

Subteams	**Members**		**Subteams**	**Members**	
Executive	David Welch	dswelch@umich.edu	**Business**	Connor Pang*	pangc@umich.edu
	Jason Ning	zyning@umich.edu	**Embedded Systems**	Joshua Ning*	joshning@umich.edu
	Alan Teng	thtalan@umich.edu		Eric Barbieri**	ericbarb@umich.edu
	Emily Wu	emilyywu@umich.edu		Aakash Bharat	aakashvb@umich.edu
Platform	Drew Boughton*	drbought@umich.edu		Brinda Kapani	bkapani@umich.edu
	Chloe Akombi**	cjakombi@umich.edu		Yuvraj Singh	uvsingh@umich.edu
	Cara Blashill	blashill@umich.edu		Liam Donegan	ldomegan@umich.edu
	Yuri Carnino	ycarnino@umich.edu		Katherine Shih	katshih@umich.edu
	Aidan Deacon	agdeacon@umich.edu		Layth Abdelkarim	laythabd@umich.edu
	Simba Gao	simbagao@umich.edu		Eric Bi	ericbi@umich.edu
Sensors	Kari Naga*	knga@umich.edu		Ruey Day	rueyday@umich.edu
	Annie Li**	anranli@umich.edu		Mahdi Chowdhury	mahdichy@umich.edu
	Yamato Miura	yjmiura@umich.edu	**Computer Vision**	Sydney Belt*	sydbelt@umich.edu
	Erika Chen	erikachn@umich.edu		Liyufei Meng**	liyufeim@umich.edu
Navigation	Chris Erndteman*	chrisern@umich.edu		Arnav Shah	arnshah@umich.edu
	Benjamin Rossano**	brossano@umich.edu		Matthew Gawthrop	mgawthro@umich.edu
	Maaz Hussain	maazh@umich.edu		Taylor Nguyen	taylorng@umich.edu
	John Rose	johnrose@umich.edu		Ryan Beaudoin	rbeaudoi@umich.edu
	Krishna Dihora	kdihora@umich.edu		John Lancaster	lancasjo@umich.edu
	Ryan Liao	ryanliao@umich.edu		Awrod Haghi-Tabrizi	ahaghita@umich.edu
	Ethan Hardy	hardyem@umich.edu		Parsanna Koirala	parsanna@umich.edu
	Aarya Kulshrestha	akulshre@umich.edu		Krishna Sharma	sharmakr@umich.edu
	Ryan Lee	ryalee@umich.edu		Dev Srikar Nimmala	dnimmala@umich.edu
	Oscar Rangel	oscarran@umich.edu		Rohan Raju	rrohan@umich.edu
	Christian Chen	chrc@umich.edu		Nithin Reddy	nkreddy@umich.edu
	Matthew Prince	mattpri@umich.edu		Jonathan McFee	jmcfee@umich.edu
	Gordon Lim	gbtc@umich.edu		Tom Vu	tomvu@umich.edu
	Om Arora-Jain	omaj@umich.edu		Jovan Yap	jovanyap@umich.edu
	Akhil Nair	aknair@umich.edu		Sophie Li	sophieli@umich.edu

*Lead, **Assistant Lead

1. INTRODUCTION

After a year-long development process, our team is delighted to introduce the University of Michigan - Ann Arbor's Autonomous Robotic Vehicle Team's (ARV) new robot – mARVin for the 2024 Intelligent Ground Vehicle Competition. ARV is supported by both campus and corporate sponsors. The team is supported by the University of Michigan Robotics Department. Our corporate sponsors include TE Connectivity, Mcity, Ford, Bose, Ann Arbor SPARK, Aptiv, Siemens, General Motors, Bird, Northrop Grumman, Milwaukee, and Raytheon.

1.1. *Team Organization*

Our team is organized into six subteams: Business, Computer Vision, Navigation, Embedded Systems, Platform, and Sensors. The Business subteam handles sponsor relations, as well as the media/marketing side of the team. The Platform subteam designs and builds the robot chassis to fit within the given design requirements. The Embedded Systems subteam develops the electrical system for the robot; this includes motor control, safety features, the status indicator light, and the power delivery on the robot. The Sensors subteam configures the robot's sensors to compute odometry and Simultaneous Localization and Mapping (SLAM) to produce a map of the surrounding environment. The Computer Vision (CV) subteam develops drivable area detection algorithms using machine learning. The Navigation subteam develops the path planning, GNSS integration, and simulation environment. The leadership team consists of the Team Lead, Operations Director, Engineering Director, and Student Affairs Lead, as well as leads for each subteam. The Team Lead, Operations Director, Student Affairs Lead, and Engineering Director provide the general direction and strategy for the team, and the subteam leads focus on the technical development of their respective subsystems.

1.2. *Design Process and Assumptions*

The design process for the robot follows a system V-Model, Figure 1. The team placed an emphasis on member education, and research-based projects to develop industry-applicable skill sets. Throughout the fall semester, the platform team consolidated design requirements with the other subteams to develop a CAD plan for the robot, while the other subteams used the time for onboarding and algorithm development. We held monthly design review meetings with mentors to discuss design choices and the implementation plan. In the winter semester, simulation testing was done in parallel with robot fabrication and hardware deployment. The final months of the design process consisted of integration testing of software and hardware on the physical robot.

Our project's key design assumption includes minimum wheel slippage for accurate localization, sufficient lighting for the vision system, an unobstructed surrounding environment for accurate GNSS signal, low noise in the radio frequency of the remote E-stop ensuring reliable remote control, and a reliable power source for the onboard electronics/sensors. Our design choices include a platform that facilitates the accessibility of electronics and computing units, a distributed architecture for computation tasks, and a modular approach in both hardware and software. The total cost of the vehicle is around \$10,698, a detailed bill of materials is included in Appendix C.

2. SYSTEM ARCHITECTURE

Power components include two nominal 13 V 50 A-hr $LiFePO_4$ batteries, 12 V high power buck converter, various fuses, power consumption measurement meter, Neo brushless motors and Odrive S1 motor controllers. Key electronic components includes Nvidia Jetson AGX Orin, Razer Blade 15, Velodyne VLP-16 lidar, Adafruit BNO055 IMU, Zed 2i depth camera, Ublox C099-F9P GNSS, optical rotary encoder ISC3004, and STM32 Nucleo-F446ZE. Key safety components include remote and physical E-stop and fuses. Documentation regarding the interconnection between these components can be found in Figure 2.

The software system includes a Zed 2i Stereo Camera and a Jetson AGX Orin for lane line and pothole detection with a machine-learning approach. An MQTT Message Broker to transfer an occupancy grid through ethernet. A simultaneous localization and mapping algorithm (SLAM Toolbox) was employed to localize the robot and detect any obstacles above ground level. An Extended Kalman Filter was used to fuse the encoder and IMU data for a more robust localization. An Occupancy Grid Merger was implemented to overlay the Computer Vision and SLAM outputs into a single map. The GPS node will read the goal coordinates provided by the completion and set destination coordinates for the A* and D* Lite algorithms. With a DWA local planner, the command velocities are sent to the STM32 Nucleo with USB protocol. The STM32 Nucleo processes the incoming velocity messages and performs velocity transform taking into account the gear ratio of the customized gearbox and sends the desired motor RPM to the two Odrive S1 motor controllers. It is also possible to tele-op the robot with a PS4 gaming controller. A state selector is toggled with a push of a button, to take in command velocities from either the local planner or the PS4 controller, a microROS subscriber reads the current state and blinks the status light according to competition rules. An overview of the software system architecture is shown in Figure 3.

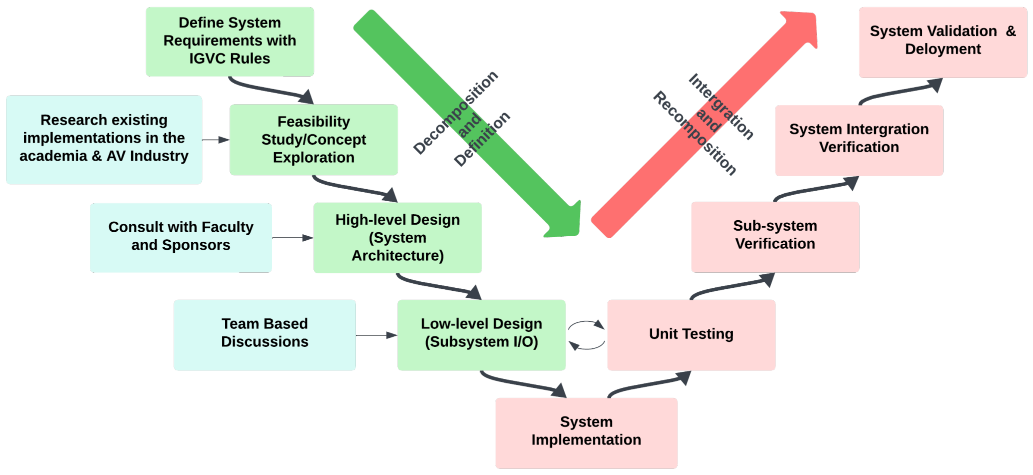

Define System Requirements with IGVC Rules
Research existing implementations in the academia & AV Industry
Feasibility Study/Concept Exploration
Consult with Faculty and Sponsors
High-level Design (System Architecture)
Team Based Discussions
Low-level Design (Subsystem I/O)
Decomposition and Definition
Integration and Recomposition
System Validation & Deloyment
System Intergration Verification
Sub-system Verification
Unit Testing
System Implementation

Figure 1: The ARV Design Process

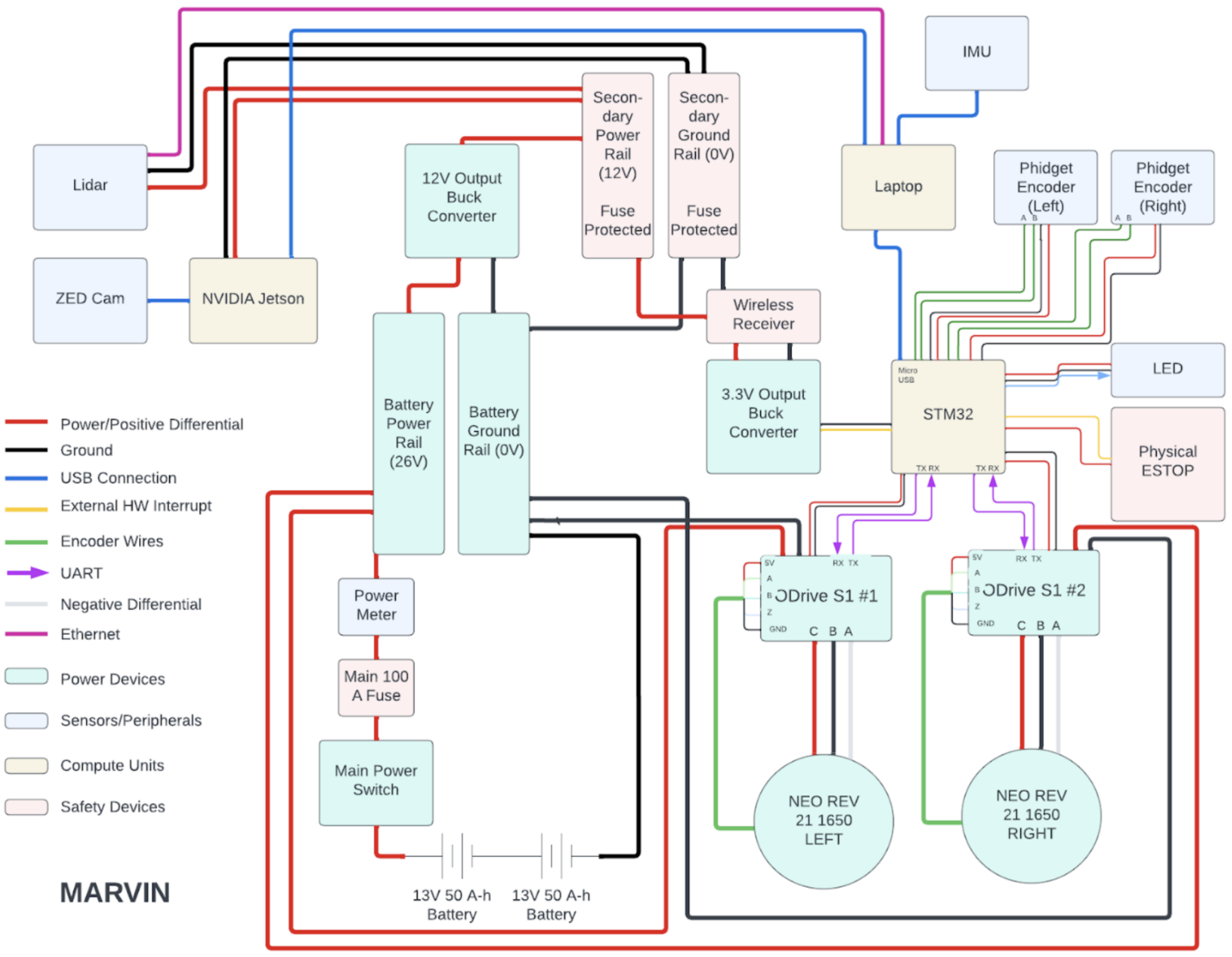


Figure 2: Electronics and Power Diagram

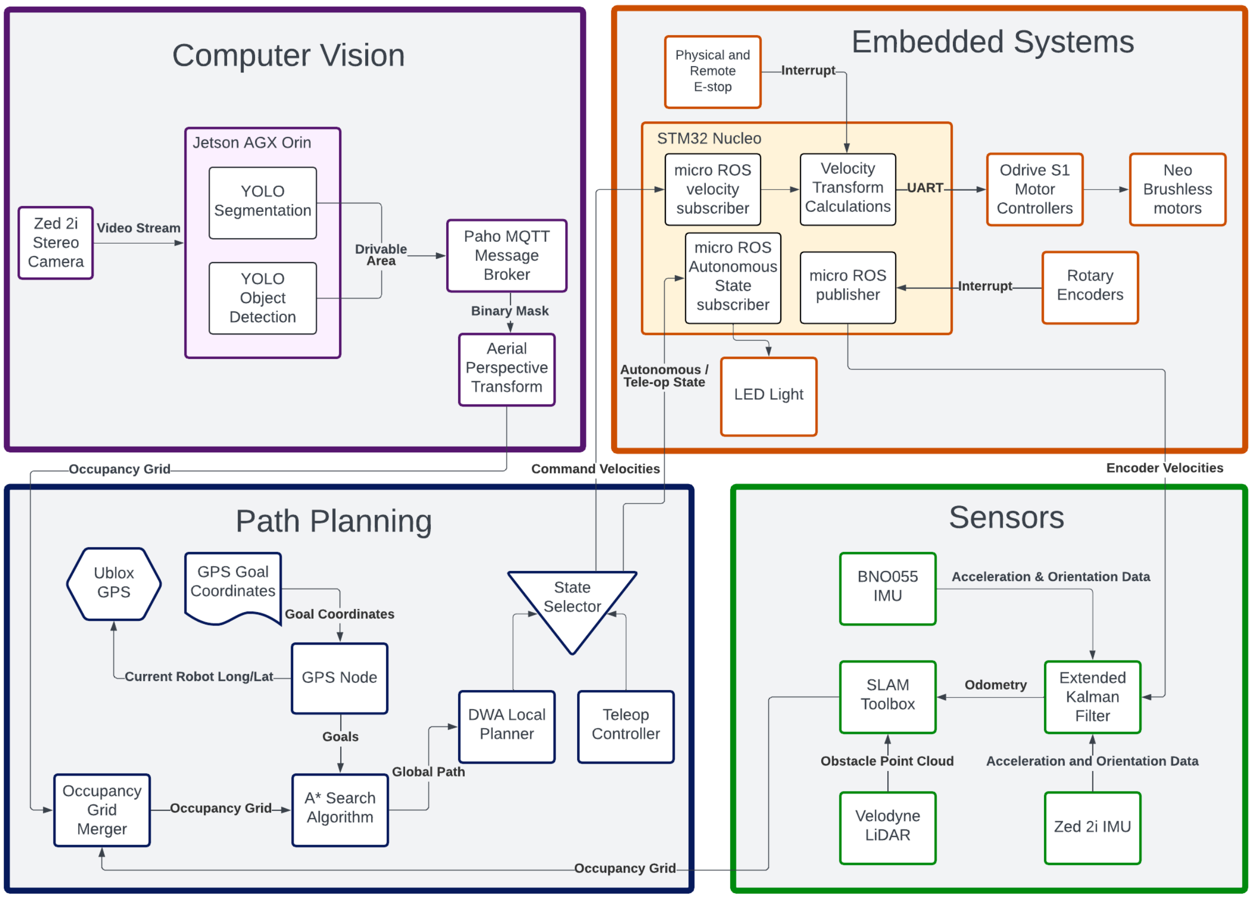


Figure 3: Connected Components

3. INNOVATIONS IN VEHICLE DESIGN

3.1. *Computer Vision: Machine Learning*

Given the complexity of environments for a commercial autonomous vehicle, we introduced a machine-learning approach for drivable area detection. This has improved our performance within the scope of the competition and laid the groundwork for applications with more complex, dynamic scenes. For fast inference and good generalization on small datasets we used a CNN based algorithm named You Only Look Once (YOLO) for our driveable area segmentation and pothole detections. We chose the latest YOLO framework release, YOLOv8 because it provides the best accuracy and has the largest open-source models available for transfer learning. By combining our lane, cone, and pothole detections into one vision framework, we eliminate the need for converting back and forth between 2D image and 3D depth information which was resource-intensive and introduced greater error margins when attempting to render sensor and camera information into a 3D world frame.

3.2. *Platform: Elevated Rotating Camera Mount*

The Zed2i camera is an essential component of mARVin's perception and is best mounted high above the vehicle and at angles that don't change while the vehicle is moving. To meet these requirements, an elevated mount for the Zed2i camera, shown in Figure 4 was designed using a planetary gearbox to facilitate a controlled tilt at the precise middle point of the camera lens. The imperfections in the gear profiles due to 3D printing of the gearbox also provided requisite resistance to perturbations while driving, thus fixing the tilt angle.

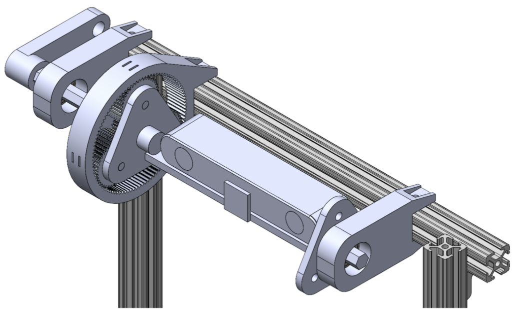


Figure 4: CAD of Zed2i camera on rotating camera mount

3.3. *Embedded Systems: PCB Design*

In an effort to reduce the noise and wire clutter in the embedded system shelf, the team focused on redesigning key electrical components — the 5V buck converter and the Nucleo into custom Printed Circuit Boards (PCBs). The buck converter offered the team an opportunity to learn Altium Designer, the industry standard for ECAD while recreating an important device. After manufacturing and testing, one self-designed buck converter was implemented in the embedded shelf to convert the 12V output of the remote E-stop to 3.3V. The next step was to design a PCB for the STM32 chip that incorporated the existing connections. The schematic included the power, UART, oscillator/clock, and reset/boot circuits that are needed for a functioning STM32 board.

3.4. *Navigation: D* Lite*

A* is a common ideal path-planning algorithm used in robotics. While it is relatively efficient compared to more rudimentary algorithms such as Dijkstra's algorithm, when traversing unknown terrain and discovering new obstacles, A* recomputes the entire path, causing repeated computation that can slow down the system. For this reason, we created a custom global planner using D* Lite. D* Lite is also a heuristic-based optimal path algorithm, but instead of replanning every new path from beginning to end, D*Lite's update step uses past information to generate paths. D*Lite also computes paths from the goal to the current pose instead of the pose to the goal. This is useful when obstacles are more likely to be discovered near the robot rather than the goal (like in this case with road lines). In this case, D*Lite only has to replan on a very small area near the robot, as it keeps the larger path between the goal and the obstacle. While both algorithms generate ideal paths, D*Lite can provide noticeable efficiency gains in scenarios where replanning is frequently needed. In our simulations, we found D*Lite to be around 30.368 % faster in ideal scenarios (see Appendix B).

4. DESCRIPTION OF MECHANICAL DESIGN

The Platform subteam was responsible for designing and manufacturing mARVin. Solidworks was the primary CAD software utilized and tools used for fabrication include drills, bandsaws, 3D printers, a waterjet, and a laser cutter. The total dimensions of mARVin are 34 inches in width, 44 inches in length, and 65.5 inches in height. Figure 6 shows a complete CAD model of mARVin in Solidworks.

4.1. *Significant Mechanical Components*

Key mechanical components of mARVin include its driving gearboxes, powered and passive wheels, and elevated rotating camera mount, all pictured in Figure 5.

The two custom driving gearboxes facilitate mARVin's differential drive by directly powering the driving wheels. Differential drive, completed by a back-mounted freely-rotating castor, was chosen for its advantages in turning and simplicity of design and simulation. Each gearbox is driven by one Neo brushless motor and outputs at a 98:3 gear reduction. This gear reduction was chosen based on the maximum competition speed of the vehicle and verified for ramp traversal.

The elevated rotating camera mount has been introduced as a Platform innovation in Section 3.2. The planetary gearbox that rotates the camera was modified from an existing model while the rest of the mount was custom designed and 3D printed.

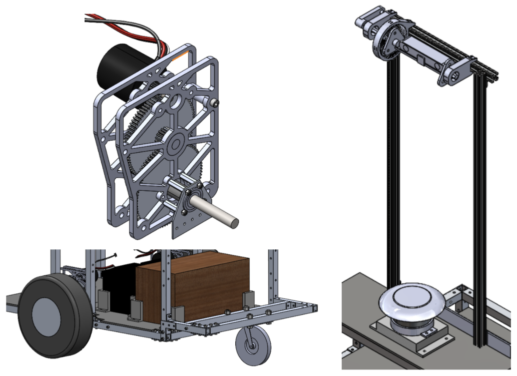


Figure 5: Top left: driving gearbox. Bottom left: powered and passive wheels. Right: elevated rotating camera mount

4.2. *Decision on Structure Design*

The interior structure of mARVin is divided into two sections to separate the electronics from the vehicle loads. The frontmost region, pictured in Figure 6, provides a large and accessible area for electronics to be mounted securely. An angled roof also prevents water stagnation in the event of precipitation. Also pictured in Figure 6 is the battery and payload section, which is situated behind the driving wheel axis to backset the center of mass of mARVin, preventing tipping of the vehicle during operation and transport. Additionally, mounting the payload closest to the rear of the vehicle makes it easily accessible and prevents interaction with electronics or wires during insertion and extraction. Shelving was also installed to elevate the laptop, improving user interaction.

4.3. *Weatherproofing*

To protect electronics against debris during operation, mARVin is completely shielded using thin sheets of polyethylene. This functions as a rigid, yet lightweight, defense mechanism against particulates during operation in nominal weather. Additional coverings for the bottom of the gearboxes have also been printed to protect the gears from accruing or otherwise transmitting outside material.

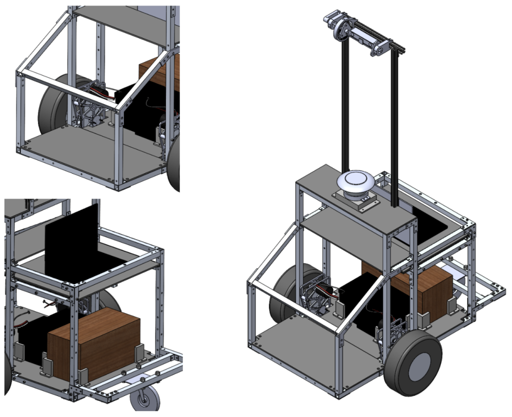


Figure 6: Top left: electronics housing. Bottom left: load housing. Right: isometric view of mARVin

In the event of light precipitation, Figure 7 shows a thin vinyl covering that can be secured to mARVin's exterior, effectively waterproofing the interior.

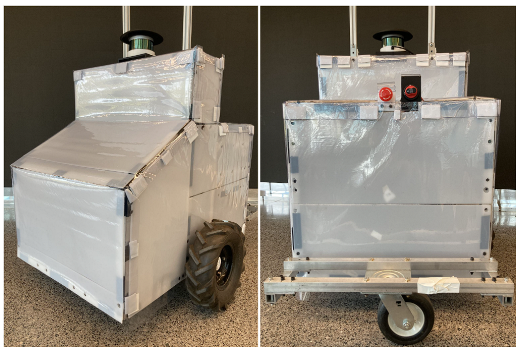


Figure 7: A demonstration of the raincoat protecting the electronics stack while not limiting function

4.4. *Suspension*

Due to low operating speed and minimal disturbances on asphalt, a suspension system was not designed for mARVin.

5. ELECTRONIC AND POWER DESIGN

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. Figure 2 shows a diagram of the electrical and power system.

5.1. *Electronics Suite Description*

A wide range of electronics are deployed on the vehicle, including computers, a GNSS, and motors. These are described further below.

5.2. *NVIDIA Jetson AGX Orin*

The NVIDIA Jetson provides a high-speed discrete GPU suitable for real-time image processing and machine learning acceleration. It pairs particularly well with the ZED camera, as Stereolabs maintains an SDK specifically for the Jetson. The Jetson provides exceptional performance considering its power consumption, form factor, and price. In addition, the included development board has integrated HDMI, Ethernet, USB-A, USB-C, and WiFi to speed up development.

5.3. *Razer Blade 15*

The Razer Blade 15 has an Intel i7 CPU paired with Nvidia RTX 3060 mobile GPU. This combination of hardware enabled the ability to perform point cloud processing, localization and mapping, path planning, and computer vision tasks simultaneously. The laptop form factor is convenient to work with and allows for hours of testing without the need to be powered as shown in Figure 8.

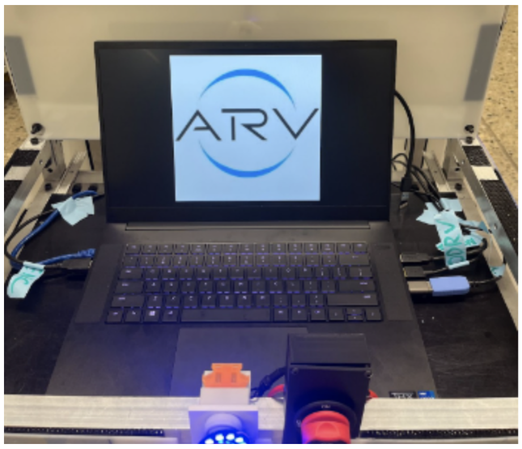


Figure 8: Laptop on the top shelf

5.4. *Velodyne VLP-16*

The Velodyne VLP-16 outputs a 360 degree 3D point cloud with a refresh rate of 10 Hz. The Velodyne has significantly increased range, accuracy, and weatherproofing over the RP-LiDAR we used in previous years and has become an integral part of our sensor systems.

5.5. *Adafruit BNO055 Absolute Orientation Sensor*

The BNO055 IMU provides absolute orientation, angular velocity, acceleration, magnetic field strength, and temperature data. This sensor was chosen because of its high refresh rate, low noise data, and the well documented libraries and packages.

5.6. *Stereolabs ZED 2i Camera*

The primary draw of the ZED camera, shown in Figure 9, is its low cost and high depth-sensing range. Compared to other RGBD solutions, the ZED camera offers much higher depth cloud resolution through software processing of the stereo images for validation purposes. The Stereolabs development team has provided a rich SDK with ROS integration included, speeding up deployment cycles by reducing hardware and embedded development time.

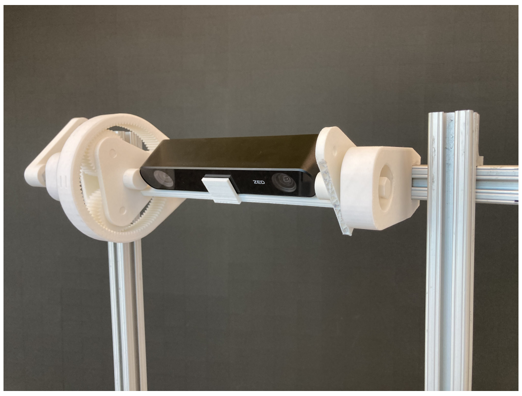


Figure 9: Mounted ZED camera

5.7. *Ublox C099-F9P GNSS*

The Ublox C099-F9P offers less than 1.5 meters of horizontal accuracy. Our decision on the GNSS/GPS system was a crucial design choice as we required high accuracy yet demanded an affordable approach. The Ublox C099-F9P offers exceptional horizontal accuracy with even greater accuracy achievable through RTK. Additionally, the Ublox C099-F9P achieved our performance requirement at a reasonable price point.

5.8. *ODrive S1 Motor Controllers*

The ODrive S1 motor controller offers a much more streamlined user experience than the Odrive previously used by the team. It offers tight integration with velocity commands, having built-in PID position and velocity control. In addition, a wide variety of customization and diagnostic options provide a significant quality-of-life boost while interfacing with hardware. ODrive S1s are also able to add velocity and current limits via software, creating another level of safety for the vehicle. Currently, the maximum speed is set to 5 miles per hour.

5.9. *Neo Brushless Motors*

The Neo motors were chosen to drive due to the availability of the motors within the team. Its low current consumption, empirically measured at less than 2A per motor under load, will not put unnecessary strain on the battery system. Combined with the gearbox, the Neo motors provide enough torque to accelerate the robot to 5 mph within 1 second. With additional power, the motor enables the robot to climb slopes up to 20 degree slope while maintaining 5 mph speed.

5.10. *Phidgets Optical Rotary Encoder ISC3004*

The Phidgets Encoder is mounted on the gear box. With the calculated gear ratio, and the 360 CPR, 80 kHz data, we are able to interpolate the position and velocity of the robot. These encoders also add physical support to the wheel axles themselves.

5.11. *STM32 Nucleo-F446ZE*

The STM32 Nucleo-F446ZE (Nucleo) is the main microcontroller of the robot. The Nucleo interfaces the hardware such as the encoders, E-stop, LED lights, and Odrive motor controller with ROS2 through the use of microROS agents. Although microROS provided frameworks for interfacing with ROS2, the team had to develop libraries for controlling the encoders, E-stop, LED lights and ODrive motor controllers from scratch. The Nucleo board offers significant upgrades from the previously used Arduino as it has much more GPIO functionalities, higher clock speeds, and microROS support.

5.12. *Power Distribution System*

The power distribution system consists of dual nominal 13 V 50 A-hr $LiFePO_4$ batteries connected in series and step down converters. The two batteries are connected to the main power rail that maintains a 26 V difference; this then feeds into a 12 V high-power buck converter that is connected to a secondary power rail. The 26 V power rail directly supplies power to two Odrive S1 motor controllers while the 12 V power rail supplies power to peripheral electronics such as the LiDAR, the Nvidia Jetson, and the remote E-stop. A buck converter converts the 12 V output of the remote E-stop to 3.3 V so it can be safely fed into the Nucleo GPIO pin. The Nucleo is powered by the laptop and the ZED 2i camera is powered by the Nvidia Jetson. Historically, the team used twelve 4700 µF capacitors connected in parallel to ensure power integrity between power and ground, however this is no longer necessary due to the increased main power rail voltage from 12 V to 36 V. Empirical power consumption of the robot is around 30 W at idle and 180 W when operating under load. From aforementioned observations, it is derived that the robot will have a battery life of around 20 hours. The batteries will recharge to full within 2 hours.

5.13. *Safety Devices and Integration*

Being able to operate our robot safely is a key part of the design. The main power from the batteries is enabled to the robot by flipping a circuit breaker mounted on the outside of the bot, easily seen and accessible by anyone. A 100 amp fuse protects the 26V power rail and downstream devices from catastrophic errors while a fuse corresponding to the current consumption of each peripheral device protects the 12V power rail. The ODrive S1s also have a 5 mph limit and 30A current draw limit per motor set as part of its configuration. Upon exceeding this threshold, the ODrive will immediately halt the motor in violation. To ensure that no safety issues arise during a run, a physical E-Stop and a remote E-Stop is attached to the Nucleo through hardware interrupt. When the E-stop is enabled, the robot will come to an immediate stop via a 0 m/s command that overrides all other velocity commands. Pressing the remote and physical E-stop buttons again will allow the robot to resume where it left off. The remote E-stop has a measured operational range of 150 feet.

6. SOFTWARE STRATEGY AND MAPPING TECHNIQUES

The robot's software is powered by the Robot Operating System 2 (ROS2) running on a base Ubuntu 22.04 installation. In line with our modular design philosophy, ROS2 was selected as the robot's operating system due to its extensive modularity, community support, and power features. ROS2 is a distributed networking and communications library that allows multiple devices to work together. A ROS2 computation graph is divided into discrete nodes that can publish and subscribe messages. Nodes communicate with each other over TCP, allowing them to connect to nodes on other computers through our Ethernet switch. This system facilitates the communication between different processes and enables the team to work on independent tasks; each software subteam can develop nodes entirely separately from the others.

With our new approach to machine learning for drivable area detection, further distributed architecture was necessary to facilitate efficient evaluation on each frame and eliminate latency between components. The machine learning algorithms are being run on the Jetson AGX Orin and the raw outputs are passed over ethernet to the laptop through a Mosquitto broker messaging protocol. Post-processing is done to perform a perspective transform that positions the robot in a world frame taken from its aerial view. This ensures compatibility with occupancy grids from computer vision and sensors given to navigation.

The robot's goal is to navigate through a series of waypoints while avoiding obstacles identified with data from the onboard sensors. Figure 10 shows the connections between the sensors and navigation subteams. The sensors team creates an occupancy grid using a variety of different tools, which is then passed to the navigation subteam to path plan to the next GNSS waypoint. This process is explained in more detail below.

6.1. *Obstacle Detection and Avoidance*

We use the Velodyne VLP-16 for identifying obstacles above the ground level. It has a 100-meter range, and 360° field of view, which is perfect for detecting cones and other roadblocks. We use the ZED camera for detecting ground level obstacles such as lanes and potholes. With the raw camera feed as the input, we run our two YOLO v8 models: segmentation for drivable area detection between lanes and around cones, and object detection for pothole identification. After performing a perspective transform based on camera parameters to render an aerial view of the occupancy grid for the lanes and potholes, we pass that information to our path planning algorithm. Simultaneously, data from the LiDAR is fed into SLAM Toolbox, which generates an occupancy grid of the obstacles and localizes the robot in space. Finally, both of these generated occupancy grids are sent to the navigation stack, which reconciles the two sources and uses a global planner and local planner that work together to create a path to the next waypoint while making sure not to move too close to any obstacles detected. Our custom A* and D*Lite nodes are explained in more detail in the Software Strategy and Path Planning section but allow us to quickly change our pathing to avoid obstacles that we may discover or encounter while moving.

6.2. *Map Generation*

We utilize a sophisticated pose-graph Simultaneous Localization and Mapping (SLAM) solution called SLAM Toolbox. SLAM Toolbox offers a robust and highly configurable solution that permits us high confidence in the quality of generated maps, especially in noisy environments.

SLAM Toolbox integrates with ROS and provides an occupancy grid containing the obstacles identified in the point cloud data from the LiDAR. The generated occupancy grid is global and uses the odometry data from our Extended Kalman Filter, which is described in the next section, to dynamically update based on new scans while maintaining localization and previous state, meaning it remembers previous obstacles it can no longer see and is a complete map of the world visible to the LiDAR. At the same time, we utilize data from encoders and the IMU in the Extended Kalman Filter, as well as LiDAR point cloud matching to estimate the location of the robot in the map.

6.3. *Software Strategy and Path Planning*

Sensor fusion between the IMU and wheel encoders is accomplished through an Extended Kalman Filter with careful weighting given to each sensor to ensure accuracy without drift. The GNSS was chosen to be left out of odometric sensor fusion due to its non-continuous nature, which testing revealed significantly reduced the accuracy of pose estimates. This data is then published as odometry messages and used by SLAM to aid in mapping and the generation of the obstacle costmap. In addition to the costmap generated by SLAM, our robot utilizes computer vision data to create a separate map containing lane line and pothole information. The separation of the computer vision data and overall sensor data highlights a major design choice in our robot's mapping. After the generation of these two maps, one from computer vision and one from SLAM, our robot merges the two maps, utilizing the combined map for

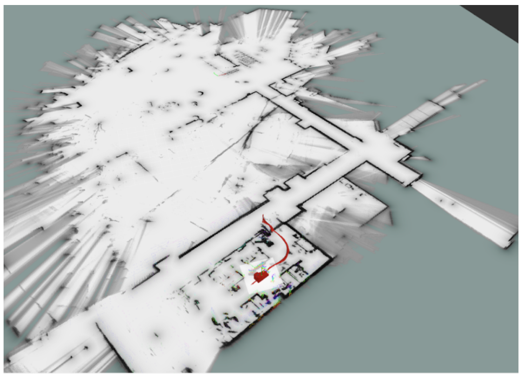


Figure 10: Picture of the Map

path planning. The final costmap from lane/pothole detection and SLAM is continuously provided to our global planner, which uses one of our two custom pathing algorithms nodes. Our first node is a custom A* node implementation, using the euclidean distance as a heuristic function (while not as accurate as other common heuristics such as manhattan distance, can be calculated in constant time rather than time linear with respect to grid size). Additionally, we have implemented a custom D* Lite node (Section 3.4).

We also created a custom behavior tree to limit recalculating a path to only when there are new obstacles that would directly interfere with the current path of the robot. We created custom packages that uses our custom A* and D* Lite code to replace the base global planner provided in nav2. These packages include custom tuning that can be launched interchangeably at robot initialization.

6.4. *Goal Selection and Path Generation*

The GNSS waypoints, provided by IGVC, are transformed from the Geographic Coordinate System (Latitude, and Longitude), to our robot's world frame. Additionally, we transform real time GNSS positioning data, from a u-blox C099-F9P GNSS, into our robot's world frame to calculate current proximity to the current goal. During navigation to a given goal, the GNSS node provides custom waypoints to our search algorithm. In the event that the way points are outside of the global map's bounds, the GNSS node is able to create a temporary local target at the intersection between the global map's boundary and the current goal. After the waypoints have been assigned and a global path has been generated, we utilize the Dynamic Window Approach (DWA) local planner to generate command velocities.

7. CYBER SECURITY ANALYSIS USING RMF

7.1. *NIST Framework*

The NIST framework contains the following steps. Prepare: identify key risk management roles and organization-wide risks. Categorize: determine the adverse impact when the system is compromised. Select: choose and document the control that is necessary to protect the system from the risks. Implement: materialize the selected control. Assess: determine if the controls are correctly implemented, and measure effectiveness against system requirements. Authorize: Provide accountability by requiring a senior official to sign off on the control method and implementation used. Monitor: continuously monitor and assess the effectiveness of the control to maintain situational awareness about the security system.

For the first step in the NIST Risk Management Framework (RMF) we established which team members will be responsible for cybersecurity. The responsibility of security fell to our leadership members, since they had a better understanding of the high level concepts we are using. Our next step was to categorize which systems may be vulnerable to attacks. Our velocity commands were our most important data to protect, since an attacker controlling the robot could be physically dangerous. One example of a NIST SP 800-53 control that we implemented is AC-6(10) (prohibit non-privileged users from executing privileged functions). Our usage of AC-6(10) is explained in more detail in the High Impact Threats Analysis section, and our implementation is as well. The passcode mechanism was assessed throughout our development cycle, as we would have to use it every time we wanted to work on the Nucleo board. We have authorized the use of this system, and will be maintaining it as we deploy it during the competition.

7.2. *High Impact Threats Analysis*

From the embedded systems standpoint, a high impact threat can happen when an adversary gains physical access to the Nucleo board, either through using the USB connector or an JTAG connection, which would allow them to gain access to the code on the Nucleo board and upload unauthorized code. Upon the insertion of malicious code, the adversary will be able to inject commands to the Odrive S1, potentially disabling E-stop and tele-op functions. One control method that can effectively prevent the described issue is through the use of memory protection and passcodes. To prevent the adversary from uploading malicious code, a passcode program can be implemented on non-erasable flash memory. The passcode program will check some predefined registers for passcodes in the user-programmable memory to see if it matches with the pre programmed passcode. This means that the adversary will need access to the passcode, the location of the register for the passcode, and physical access to the Nucleo board in order to upload malicious code. In addition, STM32 chips offer Readout Protection (RDP) level 1 protection which disables access read, erase, and program to flash memory. Combining passcode and RDP level 1 protection, the adversary will not be able to read out the program on the Nucelo board or upload malicious code without knowing the passcode. This method can be tested by attempting to upload code to the Nucleo board.

8. ANALYSIS OF COMPLETE VEHICLE

8.1. *Software Vehicle Failure Modes*

If the vehicle becomes stuck or is unable to find any possible paths to the goal, the vehicle will enter a recovery behavior state. The vehicle will start by slowly rotating in place to re-localize itself on the created map. Once the vehicle has remapped the surrounding area, and finds a path to the provided goal, it will resume normal navigation behaviors. In the extreme case that the robot is fully stuck in place, it will increase the power provided to the motors to forcibly remove itself from an obstacle.

In case of SLAM scan matching algorithm failure, the newest odometry information is used to estimate the current pose of the robot. SLAM nodes are updated using forward projection according to the optimal solution for the pose graph.

8.2. *Hardware Vehicle Failure Points*

Mechanically, the robot could potentially fail from the velcros and the plastic frames getting loose. Furthermore, the bolt attachments could potentially get loose. Electrically, the robot could violate user-specified thresholds (speed, current, etc.), tripping errors on the ODrive S1s which could shut off the motors. Power supply to peripheral electronics could be cut off due to a blown fuse. A LED indicator will be lit on the power rail if a fuse is blown.

8.2.1. *Failure Prevention Strategy*

The general troubleshooting process for hardware failures is as follows. First, check if the emergency stop has been accidentally triggered. Second, ensure that the status light on the Odrive S1 is flashing green, indicating normal operation. Third, verify the LED indicates the fuse's normal operation on the secondary power rail. Then, check that connector cables are securely attached. For software issues, verify that software nodes are running and messages are being transmitted. Run ROS troubleshooting like roswtf, rqtgraph, and viewframes to verify that the node and message graphs are properly set up.

8.2.2. *Mechanical*

To prevent the aforementioned mechanical failure points, the robot has been designed with an aluminum frame, deferring most of the potential stress on the velcros to the subsystems themselves. In case this is not sufficient, the team will keep a surplus of extra velcros to replace any loose velcro connections. The bolt attachments have been designed so that there are no shear forces acting on the bolt during robot operation, mitigating this failure point.

8.2.3. *Electrical*

To prevent the aforementioned electrical failure point, the Odrive S1 parameters are tuned such that the robot will not violate the current limit under normal operation. Additionally, the Odrive S1s are connected to the laptop which monitors and clears the Odrive errors when it arises.

8.2.4. *Software*

On the software side, there are many safeguards put in place to prevent unwanted behavior of the vehicle. First, the robot will not map and move to locations that are in completely unknown space, outside the range of the global costmap. This prevents the robot from moving outside of the emergency stop range during testing. Additionally, real time SLAM and path planning allows for dynamic obstacle avoidance, so the vehicle should avoid any spontaneously appearing objects on its path.

The sensors team utilizes sensor fusion concepts to minimize the effect of a sensor failure. The data from the IMU and the encoders are combined to form odometry data. When either sensor fails during an operation, the robot position can still be estimated using the other sensor, though with less precision. We have also added redundancy and modularity in our sensor systems. There is a secondary IMU in the Zed camera that we use as a backup source for orientation and acceleration data, and encoder readings can be provided by the Odrive motor controller as well as our own wheel encoders. Thus, in a case of critical encoder failure, it is possible to read data directly from the Odrive by simply subscribing to another topic in ROS.

The computer vision team utilizes buffers in their machine learning models to ensure good output even when there is a loss of data. Our software analyzes all output from the model and if an unpredicted behavior (no output, flipped data across lanes, sharp losses in driveable area) is detected, we revert back to our model's previous reading. If our model continues to output poorly we will output a full frame of driveable area which allows the robot to continue to move to a new location and provide the model with a new view. We also manually override the bottom rows of pixels to guarantee the driveable area is always connected to the bottom of the frame which guarantees mARVin always has space to move forward.

8.3. *Testing*

8.3.1. *Navigation*

To test the navigation systems, we took advantage of buildings on campus. We successfully planned paths through hallways and large rooms containing many obstacles such as tables, chairs, and pedestrians. Waypoints were added during testing by directly adding a 2D navigation goal through RViz on the onboard laptop. To test GNSS functionality, we collected multiple rosbags of GNSS data while moving through Ann Arbor. This GNSS data was then tested with the GNSS node code separately.

8.3.2. *Drivable Area Detection*

A course environment was laid out in a parking lot with white tape for lane lines, white circles for potholes, and cones for obstacles. The primary model training and testing lanes were solid, but validation was performed with dashed line, shown in Figure 11. We used teleoperation on the robot to navigate through the course and force edge cases to determine model performance in different scenarios. Physical testing was performed running our two models in the NVIDIA Deepstream SDK: object detection for potholes and segmentation for space between lanes and around cones. We were looking for two primary visual metrics: accurate designation of the drivable area and potential frames where we classify no areas for the robot to go. Given the necessity of extensive

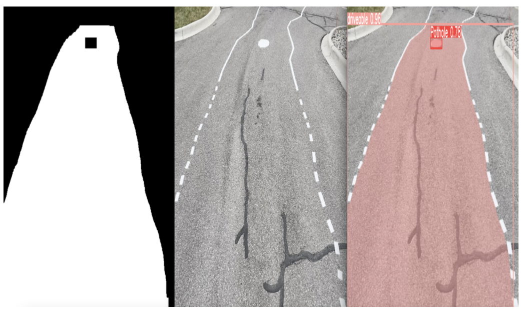


Figure 11: Model Testing with Different Lines

data for training and validation of our machine learning models, testing was also performed on videos and frames from our parking lot course simulation and old competition videos. We verified that confidence scores were within a reasonable range (70-100%) and that model performance was consistent from frame to frame especially in the lower third of the frame where our output matters most to the robot's next moves.

8.3.3. *Sensors*

Testing the sensor system involves testing individual sensors separately and integration testing with a combination of different sensors. To test the encoders, we have pushed the robot on the ground, ensuring no wheel slippage. We then verified the various distances pushed with the number of rotations recorded by the encoder multiplied by the corresponding gear ratio and wheel radius. To test the IMU sensor, we were able to visualize the data collected by the data in RViz and ROS. We have observed fairly accurate orientation measurements but a significant drift in the acceleration data. To combat inaccurate readings, we wrote a custom python calibration script that offsets the acceleration readings to achieve better results, and we use data from multiple sensors rather than just our IMU to build our published odometry. To test the GNSS sensor, we walked different movement configurations in the parking lot, then plotted the collected coordinates. Figure 12 shows the received coordinates for walking in a straight line along the parking lane.

The sensor integration testing started with simulation. We implemented a version of Extended Kalman Filter (EKF) for robot state estimation, and compared the estimated odometry data with the exact odometry vectors that's provided in simulation. We also deployed all the sensors to the robot and published the collected data to SLAM Toolbox. We tuned the configuration file with an iterative approach and were able to make the map building efficient and accurate.

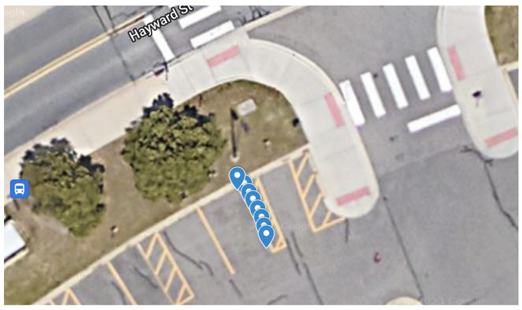


Figure 12: GNSS coordinates from walking in a straight line

8.4. *Vehicle Safety Design Concepts*

In addition to E-stops and software limits, both failure prevention strategies and testing implement several safety design concepts. For the software and mechanical prevention strategies, a soft bumper, made using a pool noodle, was included in the robot to minimize any damage from potential collisions. For the electrical prevention strategy, the robot is halted to prevent any undefined behavior. For testing, one person is assigned to solely operate the remote E-stop, ensuring that someone can immediately stop the robot when necessary.

8.5. *Simulations in Virtual Environment*

The robot and its sensors are simulated in Gazebo, and the various types of data visualized in RViz. These software were chosen because of their well documented integration with ROS. We have built the robot simulation model from scratch, with simulated sensors such as the IMU, depth camera, and LiDAR. The Navigation stack and SLAM Toolbox subscribe to the sensor data, generating the map of the planned path in the process. The environment map, licensed from UToronto, was based on the Auto-Nav course illustrated in the competition rules. We found an issue with lighting, but swiftly resolved it by editing the lighting of the world.

8.6. *Theoretical Concepts in Simulations*

We were able to simulate and debug our SLAM and autonomous navigation algorithms inside simulation. To start with, we modeled the robot and its sensors using the URDF format, and sensor plugins from Gazebo and Toyota. We wrote custom launch files to spawn and tele-op the robot into the competition world. We were able to set up SLAM Toolbox inside the simulation, subscribing to the LiDAR and IMU nodes. We tuned the Python configuration files to improve the map generation speed and quality.

One important theoretical concept that we implemented in simulation was the ground filter. By default, the lasers from the LiDAR will reach the ground and make that into an obstacle. The ground filter will delete points below a desired height to achieve clean maps for navigation.

9. PERFORMANCE TESTING TO DATE

9.1. *Component Testing*

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

9.2. *Integration Testing*

As a team divided into multiple technical groups, integration testing stands as an important step in our development process. The Navigation, Sensors, and Computer Vision team must all merge their outputs to create a navigable cost map. This integration was a complex challenge tackled by our team, as the output from both Sensors and Computer Vision existed in differing frames and resolutions. We began our integration testing early by developing ROS 2 nodes to produce “fake” output. The goal of these nodes was to allow us to test the integration system while the SLAM map and CV lane detection map were still in development. This helped us catch many bugs in our integration system, most notably we discovered a transformation issue with our merging process. As a direct result of these tests we altered our integration system to utilize tf2’s frame publishing for better transformations between the SLAM map and CV map. As we moved these systems from development to deployment, we transferred to testing iteration in simulation. The goal of these tests were to discover issues with path planning on merged data and problems with the output of both the Sensors and CV teams’ respective obstacle detection. This testing took place inside our IGVC simulation world modeling lane lines, cones, and a ramp. This testing directly exposed an issue causing the CV algorithm to mark occupancy grid cells existing on the boundary between known and unknown as obstacles.

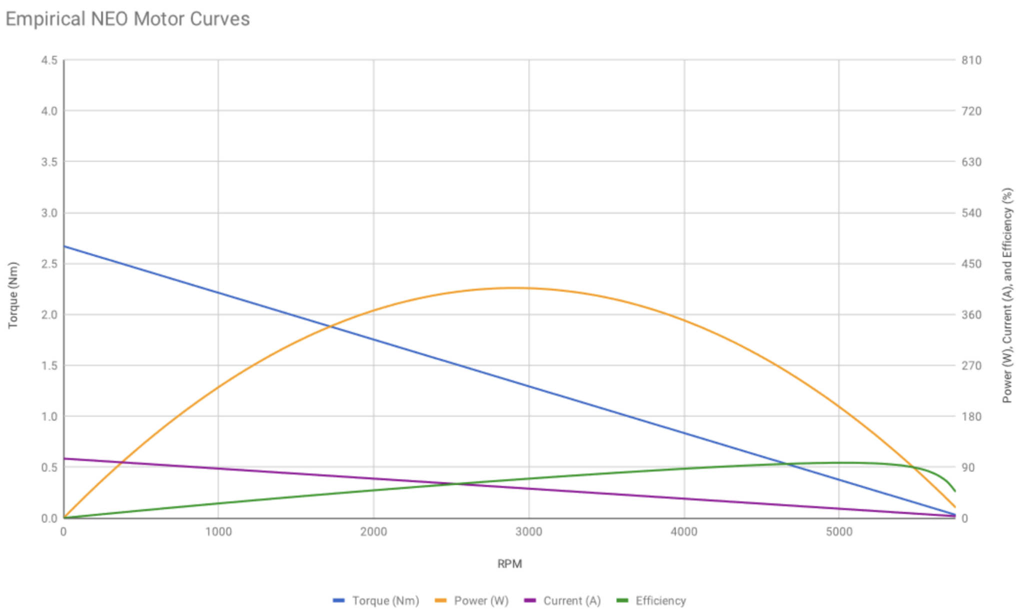
For navigation and control integration testing, the team started with ensuring the vehicle can be teleoperated. The PS4 controller was used to provide velocity commands to the Odrive motor controller. We then verified the accuracy of the velocity by measuring the wheel’s RPM against the desired velocity. Testing was deemed “good enough” when the error between the command velocity and the wheel velocity was less than 5%. At the same time, the PID of the motor ensures the system is critically damped and will reach the desired speed as fast as possible with minimal overshooting. Additionally, onboard sensors were calibrated and the Extended Kalman Filter was tuned to ensure accurate localization (without point cloud matching) with an error of less than 5%.

9.3. *Initial Performance Assessments*

Metric	**Test Result**
Max Speed	3.4 m/s
Acceleration	0.4 m/s^2
Ramp Climbing	20 degrees
Laptop Battery Life	1 hour
Robot Battery Life	120 hours standby
	20 hours running w/ motors

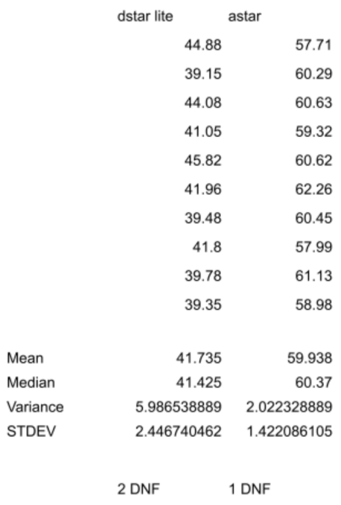
APPENDIX

A. TORQUE AND EFFICIENCY CURVES FOR NEO BRUSHLESS MOTORS

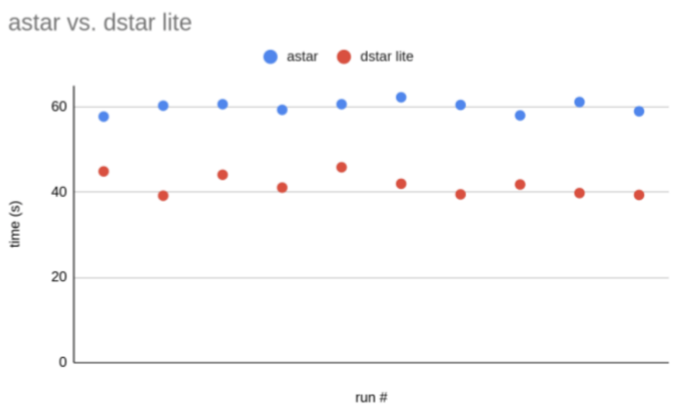


B. DSTAR LITE VS. ASTAR PLANNING SPEED COMPARISON

In order to compare the two algorithms we ran experiments in Gazebo using our simulated IGVC world. Specifically, we ran our test using our mARVin V2 robot model on the south side of the track. We ran the experiment ten times per planner to account for errors coming from any background processes running at time of simulation. All runs were given the same initial position and goal. Both the D* Lite and A* planners were also given the same global planning and robot speed/acceleration parameters. We found on average that D* Lite preformed 30.3697% faster than A*. However D* Lite had a standard deviation of around 2.44 compared to A* with a standard deviation of 1.44. We believe this is due to slight variation in sensor data, which in D* Lite causes variability in the number of nodes needed to be looked at. Overall, we found that D* Lite was less consistent in the time needed to run the model, but was generally faster than A*.



	dstar lite	astar
	44.88	57.71
	39.15	60.29
	44.08	60.63
	41.05	59.32
	45.82	60.62
	41.96	62.26
	39.48	60.45
	41.8	57.99
	39.78	61.13
	39.35	58.98
Mean	41.735	59.938
Median	41.425	60.37
Variance	5.986538889	2.022328889
STDEV	2.446740462	1.422086105
	2 DNF	1 DNF



C. BILL OF MATERIALS

Item	**Price ($)**
ODrive S1 Motor Controllers * 2	360
Neo Brushless Motors * 2	112
Phidgets Optical Rotary Encoder ISC3004	100
STM32 Nucleo-F446ZE	21
Various Buck converters	80
Wireless Receivers	17
Fuses and Power Rails	70
Physical Emergency Stop	10
LED Light	10
BNO05 IMU	30
Velodyne VLP-16 LiDAR	4000
ZED 2i Stereo Camera	500
Razer Blade 15	1600
Jetson AGX Orin	2500
Battery * 2	300
Wheel * 2	100
Gears	68
Fasteners	80
Plastics	150
Aluminum Sheet Stock	240
Aluminum Bar Stock	350
	Total: $10698