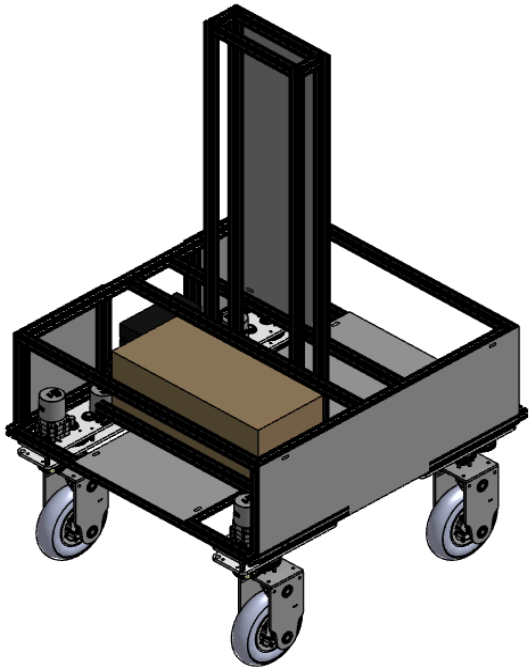
University of Oklahoma

Sooner Competitive Robotics

IGVC 2025 Design Report

Twistopher



I hereby certify that the development of the vehicle, *Twistopher*, as described in this report, is equivalent to the work involved in a senior design course.

Submitted May 15th, 2025

Software	Mechanical	Electrical
Antonio Chappell	Daniel Brown (Captain)	Kyle Engel
Abby Roland	Andrew Lopez Castro	Patrick Gallagher
Arika Khor	Micah Popejoy	Min Kang
Dylan Zemlin		
Lin Tan		
Garrison Hair		

Table of Contents

1. Design Process, Team Identification and Organization

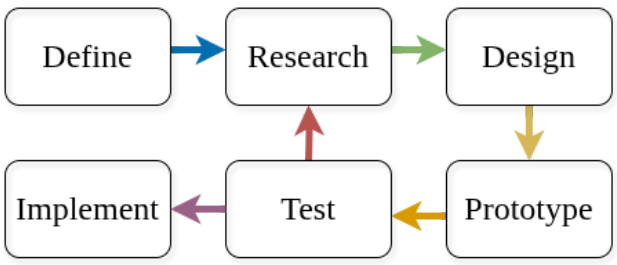
1.1 Introduction

Sooner Competitive Robotics (SCR), representing the University of Oklahoma (OU) and the Gallogly College of Engineering (GCoE), proudly presents *Twistopher*, our entry for the 2025 Intelligent Ground Vehicle Competition (IGVC) in both the AutoNav and Self-Drive categories. SCR is a multidisciplinary student organization consisting of 35 members, with a dedicated AutoNav/Self-Drive subteam of 10 students. The team is advised by Dr. Golnaz Habibi and Noah Zemlin, both faculty members at the University of Oklahoma.

1.2 Team Organization

The AutoNav team is led by a single captain and organized into three specialized subteams: electrical, mechanical, and software. The captain serves as the primary point of contact with SCR leadership, providing regular updates on team progress. Each subteam reports to the captain and collaborates internally to manage tasks based on the robot's design and performance requirements. To promote communication and prevent the formation of silos, subteams are encouraged to hold concurrent meetings and maintain continuous coordination through our team chat.

1.3 Design Process

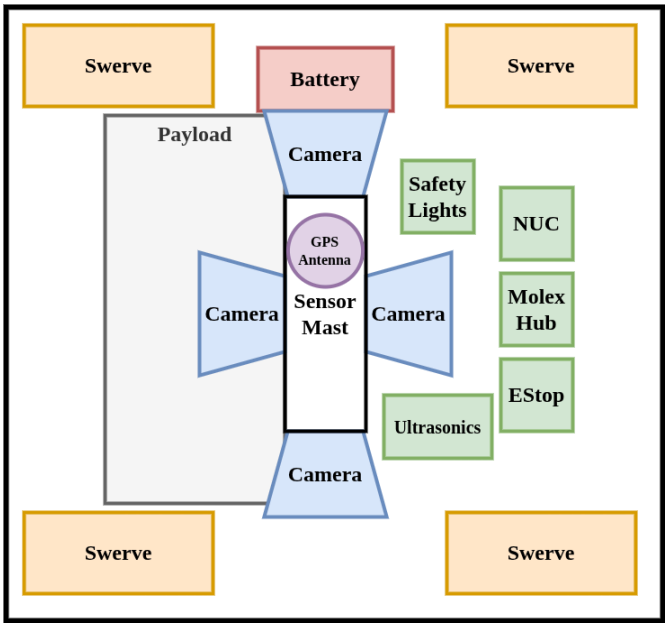


To develop *Twistopher*, the team implemented an iterative design process that focuses on a cycle of researching, designing, prototyping, and testing. The year began with a post-mortem analysis of the previous season's robot, identifying both successful elements and areas for improvement. The team then completed a full re-read of the rules to fully understand the competition and derive a list of rule-based requirements. From these, the team established high-level design goals for the robot. At the highest level was the decision to use swerve drive as the vehicle's drivebase, which primarily drove the rest of the design. With the system-level requirements defined, subteams proceeded to research, design, build, and test the necessary components. As each subsystem was completed, integration testing ensured that all components worked cohesively and that the robot met its overall requirements to ensure success.

2. System Architecture

2.1 Mechanical and Electrical Architecture

At a high level, *Twistopher* consists of several independent subsystems: chassis, swerve modules, vision, sensors, and the remaining electronics.

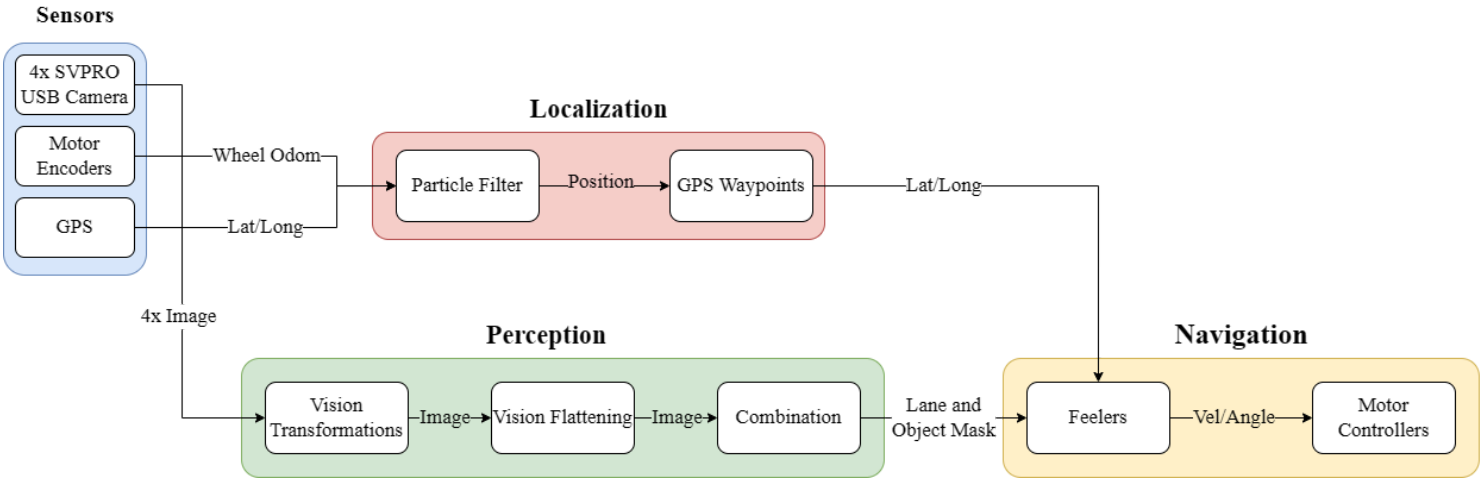


2.2 Robot Safety

Twistopher was designed with safety as a fundamental principle. Power is delivered to the drive motors only when three conditions are simultaneously met: the remote emergency stop, physical emergency stop, and software emergency stop must all be inactive. If any one of these is triggered, the motor relays will not supply power. To prevent tampering, each of the three signals is also monitored in software, where the same logic is redundantly enforced. As a result, the robot will not send any non-zero command to the motors unless all safety interlocks are satisfied.

2.3 Software Architecture

The software is split into four major subsystems: sensors, perception, localization, and navigation. The diagram below shows the communication between the subsystems. Organizing the code in this way allows for easy testing of each subsystem independently as well as making testing different algorithms or parameters quick and easy.



2.4 Robot Communication Bus

To facilitate easy communication and power delivery to all devices on the robot from anywhere, we developed a custom wire harness bus that connects all devices to a common hub. Using a 6-pin Molex connector, each device connects to our custom Molex hub which provides power, data, and an emergency stop signal. Data is communicated between devices using the CAN bus protocol, a common protocol seen ubiquitously throughout the automotive industry.

2.5 Bill of Materials

The total cost to construct *Twistopher* was approximately $5,400. Of this, around $3,500 was the chassis and mechanical hardware, while $1,900 covered electronics and computing components. Each custom swerve module cost approximately $500 in materials and electronics. The table below outlines the key commercial off-the-shelf (COTS) components integrated into the vehicle.

Major COTS Item	**Quantity on Robot**	**Total Market Cost (USD)**	**Total Team Cost (USD)**
Intel NUC Mini-PC	1	$550.00	$550.00
SVPRO AR0234 Global Shutter Cameras	4	$320.00	$320.00
12V 18Ah AGM Lead Acid Batteries	1	$50.00	$50.00
VN-200 Rugged IMU and GPS	1	$3,050.00	$0.00
NEO Motor and SPARKMAX Controller	8	$1,120.00	$1,120.00

3. Effective Innovations

3.1 Swerve Drive

Twistopher is SCR's first robot featuring swerve drive: a holonomic control system in which each wheel is independently driven and 360-degree steerable. Having a holonomic control system allows the robot to translate and rotate with full freedom in each degree. Compared to other holonomic drive systems like mecanum, swerve drive has more traction and is more suited for outdoor environments with uneven terrain or obstacles. It is also less susceptible to wheel slip and error in odometry than mecanum.

3.2 Full 360-degree Vision

SCR has identified that one of the most common ways that teams collide with obstacles at IGVC is from the sides or the back of vehicles due to cameras most frequently only seeing what is in front of the robot. To improve the consistency of *Twistopher*, the team chose to give it one camera on each of its four faces to give complete vision over its surroundings. This also unlocks the full potential of the swerve drive, as it allows for sideways translational movement to avoid obstacles detected by the side cameras.

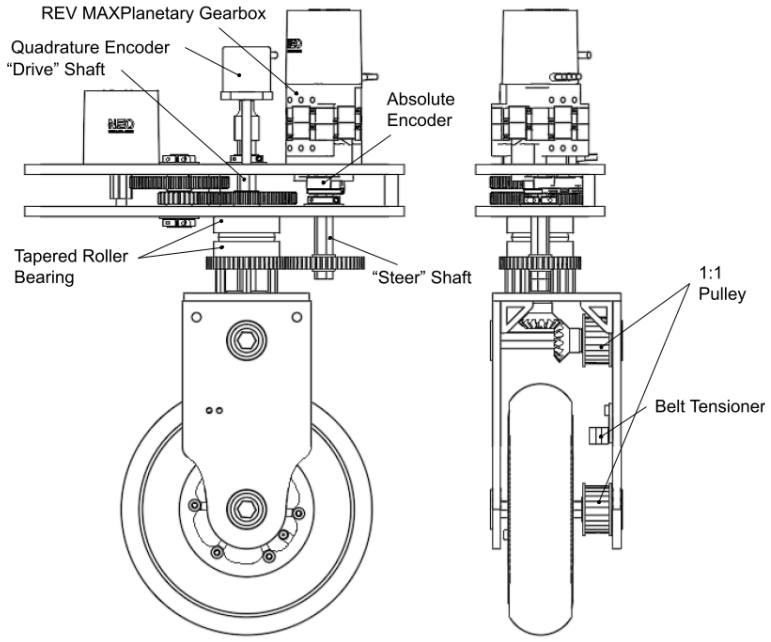
4. Mechanical Design

4.1 Overview

The mechanical design of *Twistopher* is focused on making a robot that allows for complete freedom of movement, easy access to electrical subsystems, a complete view of the course, and weather proofing. *Twistopher* has complete control over its direction and position due to the custom-machined swerve modules, and can strafe in any direction, rotate, or strafe and rotate. The electrical systems are easily accessed and wired together thanks to the large, open internals of the chassis with large doors on the top. The tall sensor mast allows the cameras to have a complete view of the course and obtain a highly accurate position of the robot relative to the lanes and obstacles. The robot protects against the elements using acrylic panelling and weatherstripping.

4.2 Swerve Modules

Twistopher features four custom-designed swerve modules that are placed in a square configuration on the robot to form the swerve drive base. The modules are made out of ¼ in. machined aluminum. 3D printed prototypes were made to test the assembly of the modules, and also for mounting on the robot for software testing while the metal modules were being machined. Each swerve module is capable of turning the wheel 360-degree continuously while providing full driving force.



The modules have a 15:1 drive gear ratio for a designed top speed of ~8 MPH so that the peak of the motor power curve is at typical IGVC speeds (motors perform optimally at around 60% of their maximum speed). An absolute encoder is installed on the steering shaft and is used to read and maintain the angle of the module between power cycles. Each swerve module features an 8-inch pneumatic wheel, which is ideal for all-terrain and all-weather performance. The modules are designed such that the gearbox is attached to the frame of the robot, so that the modules can be easily removed and installed independently of the gearboxes. This allows for easier assembly and transport, and we can have a spare module fully assembled on standby in case any need to be quickly replaced.

4.3 Frame and Sensor Mast

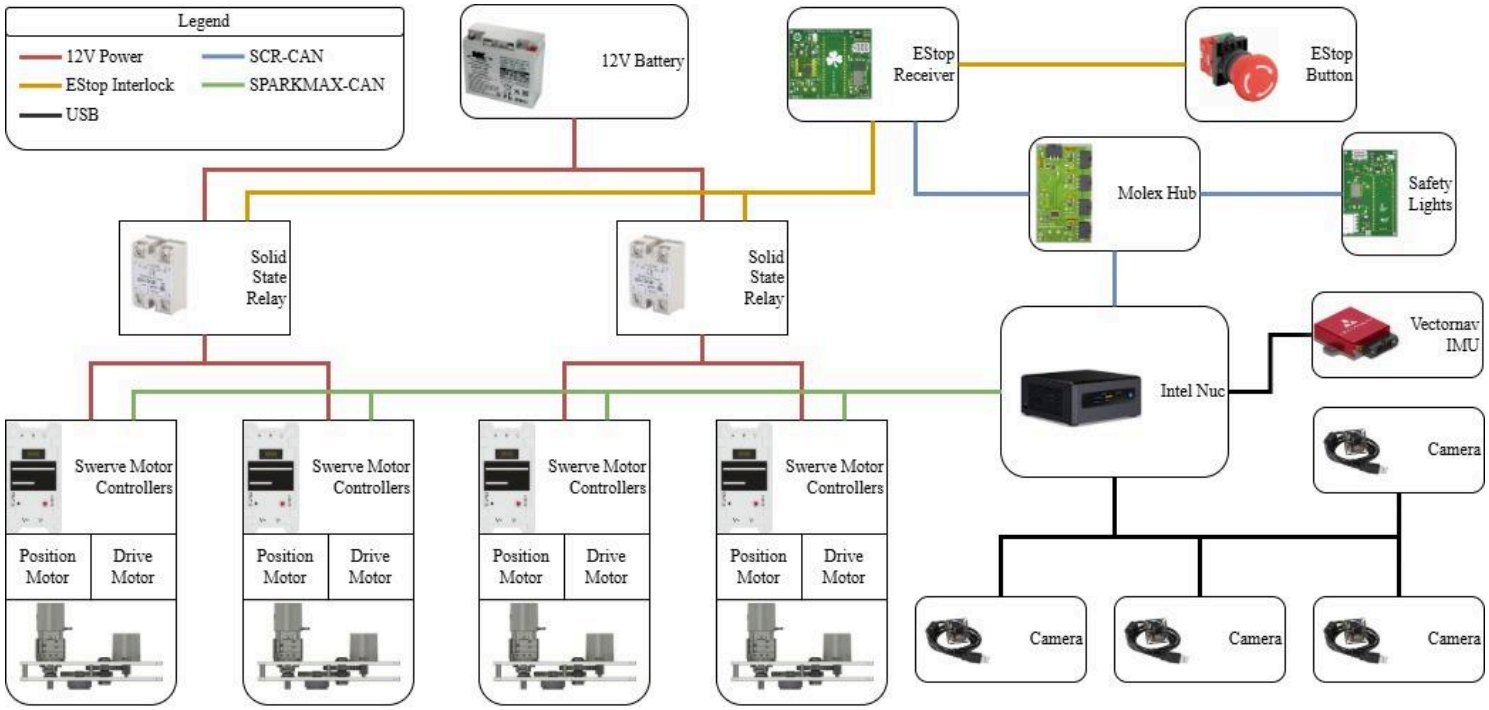
The frame was designed to maximize the performance of the swerve modules. A deliberate effort was made to mount the modules such that the wheels formed a square, as it makes turning in place easier. The frame was also designed for ease-of-access to make tasks like swapping the battery and toggling power easier. Compared to last year's vehicle, *Twistopher* is both wider and taller.

The sensor mast in the middle of the robot was designed to provide a 360-degree view of the course, as the robot can drive in any direction. One of the most important lessons learned from simulating robot parameters is that maximizing the height of the cameras can significantly improve performance. Higher cameras allow for a wider view of the course while also allowing the image to be 2D projected with minimal distortion. Additionally, the GPS antenna, the safety lights, and the emergency stop button are placed on the mast to provide clear visibility and easy access.

4.4 Weatherproofing

Twistopher features large acrylic panels for protection from both rain and sun. Weatherstripping is used to provide a watertight seal around the acrylic. This year, all electronic components are fully enclosed within the acrylic paneling, eliminating the need to route wires through a gasket. The design of the sensor mast, with custom 3D-printed enclosures for each camera, keeps rain off the sensitive electronic components.

5. Electronic and Power Design



5.1 Overview

For 2025, the SCR team continues to use a highly modular approach to electronics to keep repairability and reliability high. *Twistopher*'s electronic suite is diagrammed above and features several custom PCBs designed by the team. Every individual electronic module is connected to our custom hub in a star-configuration. The connections are made using 6-pin Molex cables that carry our 12-volt power, data over CAN bus, and the e-stop signal.

By using almost all custom electronics, we are able to implement a custom protocol designed by the team called CONBus, which allows us to remotely configure any device on the vehicle over any bus. For

Twistopher, CONBus is implemented using a specific range of message IDs as part of our CAN bus specification. Using CONBus, the software can quickly modify firmware parameters on any board, which allows for rapid testing.

5.2 Power Distribution

The vehicle uses a single 12V 18Ah AGM lead-acid battery to power both the electronics and the drivebase. Lead acid was chosen for safety, cost, and availability. The high capacity of 18Ah allows for runtimes that far exceed the 5 minutes of runtime needed for the course. Because only a single battery is used, the team can quickly swap out the battery with a fresh one any time the vehicle is operated to make sure the robot is always operating at its peak.

5.3 Motor Control

Twistopher controls the eight motors that make up the swerve drive by commanding the SPARKMAX motor controllers on a separate CAN bus from the main CAN bus. Using separate buses allows us to maximize the throughput of motor control instructions. To drive the swerve drive, four of the SPARKMAX controllers are placed in “position control” which use a high-rate internal PID control loop to set the steering position of each module based on absolute encoder feedback. The remaining four controllers are placed in “velocity control” which allows us to set the desired drive velocity of each module based on quadrature encoder feedback. The main computer can then set the desired velocity state of the robot by computing the desired velocity state of each module and commanding the motor controllers over CAN. By offloading the PID loop onto the motor controllers, we are able to simplify our electronics and further minimize the latency of the robot.

5.4 Safety Lights

Having safety lights on the vehicle to indicate autonomous state is not only required by the rules, but also a feature that we highly depend on for safety. Our safety lights are controlled by a custom PCB, which is capable of reading both the vehicle-wide CAN bus as well as the e-stop line directly. This allows the PCB to immediately update the state of the light when the robot is emergency stopped, while also giving us software control of the light to enable flashing when the robot is enabled in autonomous control.

We chose an RGB LED strip for the safety lights indicator. This is so we can change the color of the lights to use as a debugging tool for evaluating vehicle state while testing and during runs. The color and animation of this strip can be controlled via CAN bus and is implemented in our software to change whenever the robot changes state. The following table shows the map of light color to their meaning.

Light Color	**Meaning**
White	Standard operation
Yellow	Waypoint following activated
Green	Waypoint reached
Red	Unable to move forward, reversing if possible

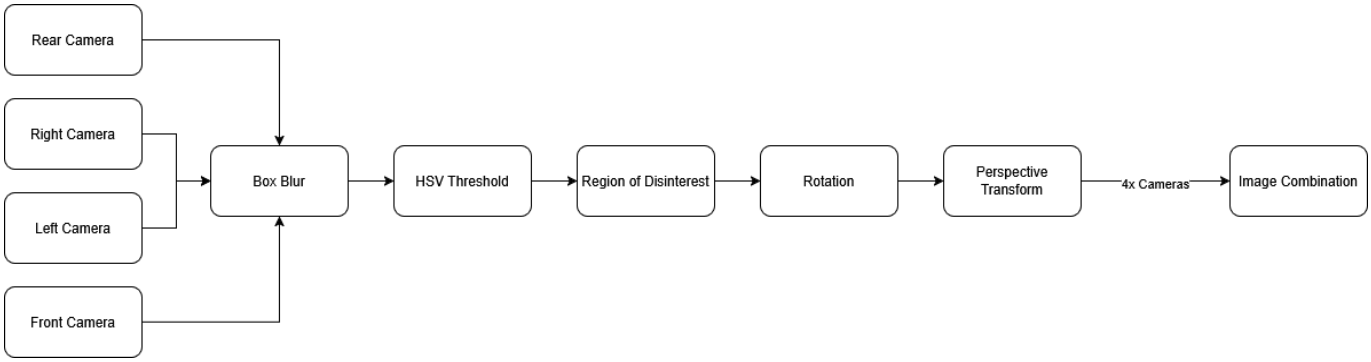
5.5 Emergency Stop and Safety

Safety is a top priority for the vehicle and is at the core of our electrical design. To maximize reliability, our emergency stop circuit is controlled by an e-stop line that is available to every control node on the robot. The circuit is powered by the e-stop relay board which features three interlocks that must be disabled for the motors to receive power: the on-vehicle button, the remote button, and a software-controlled interlock. Motor power is gated by a solid-state relay that is controlled directly by the e-stop line. To ensure that no board can accidentally affect the e-stop line, a buffer is implemented on the Molex hub to make sure that only the e-stop relay board has authority.

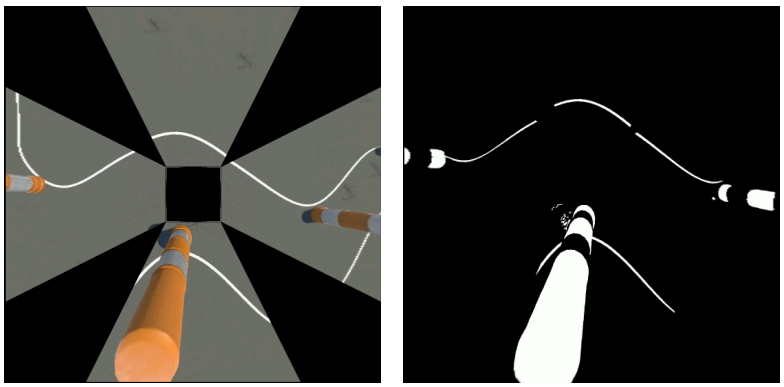
6. Software Systems

6.1 Overview

Our software uses the Robot Operating System (ROS) library, a robotics library that is widely-used in both academia and industry. It is based on a publisher-subscriber model, with individual processes called nodes sending messages to each other. This allows for much of the software work to be developed in parallel in separate packages.



6.2 Image Transformation



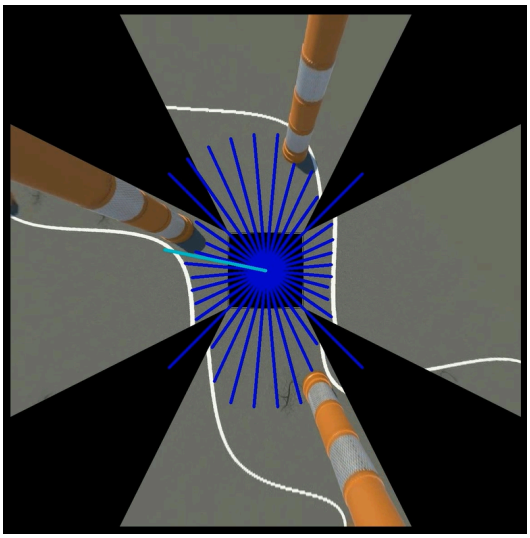
Twistopher uses a classical computer vision approach to transform our four video streams into an obstacle map that the vehicle can use to navigate. First, the images are passed through correction filters to correct for camera warp and to project the image onto a 2D plane. Then, the images pass through a series of filters that turn the images into obstacle maps. Finally, the four separate obstacle maps are overlaid to create a

master obstacle map that fully details all obstacles around the robot. This series of filters all use the OpenCV library and have shown excellent performance with latency lower than human perception.

To account for changing conditions such as cloud cover and time of day, the robot automatically calibrates at the start of each run. Upon entering autonomous mode, before enabling mobility, it captures a portion of the front-facing camera image to generate a tuned HSV threshold for operation. This threshold is then validated through an automated process that validates the percentage of obstacles detected across multiple images, as well as through manual verification by the robot operator, ensuring successful calibration.

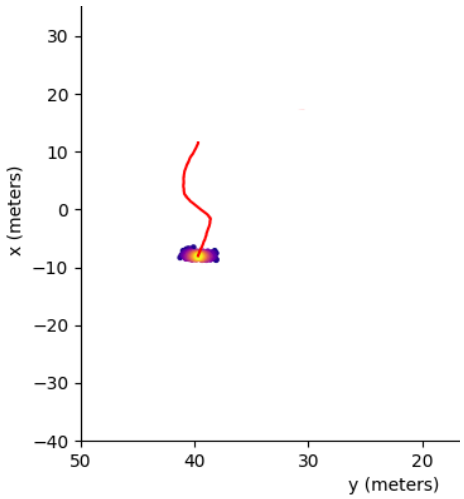
6.3 Camera-based 2D Raycast Heuristic Navigation

In previous years, SCR has used traditional mapping and path-planning approaches for navigation. This worked well, but had a major impact on latency and vehicle responsiveness due to the time and space complexity of creating and using the maps. This year, *Twistopher* features a new approach to performing navigation: camera-based 2D point cloud heuristic navigation. By performing ray-casting on the 2D obstacle image, we can create a 2D point cloud of obstacles. Then, a custom heuristic algorithm is used to decide how the vehicle should navigate based on obstacles around it. The heuristic weights individual raycasts and gives precedence to those moving in the direction of travel, as well as those towards the next GPS waypoint. Combined with the 360-degree view of the field, this allows the vehicle to make highly optimized movements that are natural, continuous, and maximize the distance to obstacles and lanes. This technique was implemented in C++ and uses parallelization techniques to minimize latency. Current tests show it to operate much faster and react to obstacles more quickly than last year's A* Python implementation.



6.4 Particle Filter Localization

Autonomous localization is required for qualifying for and completing the IGVC Autonav challenge. Waypoint navigation is only possible with accurate and precise location data. A particle filter combines precise odometry data from swerve drive motor encoders with high accuracy GPS data. This location data is consumed by the path planner as the robot's current position and used to determine waypoint collision and distance from the start of the course.



6.5 Swerve Drive Control

Swerve drive enables holonomic movement by independently controlling the speed and steering angle of each wheel module. Desired chassis velocities (forward, sideways, and rotational) are converted into individual wheel commands through a swerve transform matrix, as shown in the accompanying diagram. This matrix operation converts from an overall desired robot state to module states, which are then used to compute setpoints for the drive velocity and azimuthal position of each module. To estimate the robot's actual movement, we use feedback from two sources: absolute encoders on the angle motors provide the orientation of each wheel module, while encoders in the drive motors measure the linear displacement of each wheel. These measurements are resolved into x and y velocity components for each module, forming a state vector representing the observed wheel motions. By multiplying this vector with the pseudo-inverse of the module positions matrix, we compute the robot's estimated movement in terms of translation and rotation, providing continuous feedback for odometry and control.

Swerve Transform Matrix

$$\begin{bmatrix} 1 & 0 & -r_{y,FL} \\ 0 & 1 & r_{x,FL} \\ 1 & 0 & -r_{y,FR} \\ 0 & 1 & r_{x,FR} \\ 1 & 0 & -r_{y,BL} \\ 0 & 1 & r_{x,BL} \\ 1 & 0 & -r_{y,BR} \\ 0 & 1 & r_{x,BR} \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \rightarrow \begin{bmatrix} v_{x,FL} \\ v_{y,FL} \\ v_{x,FR} \\ v_{y,FR} \\ v_{x,BL} \\ v_{y,BL} \\ v_{x,BR} \\ v_{y,BR} \end{bmatrix}$$

Desired Robot State → **Desired Module States**

$$\text{Module Drive Speed} = \sqrt{v_{x,N}^2 + v_{y,N}^2}$$

$$\text{Module Azimuthal Position} = atan2(v_{y,N}^2, v_{x,N}^2)$$

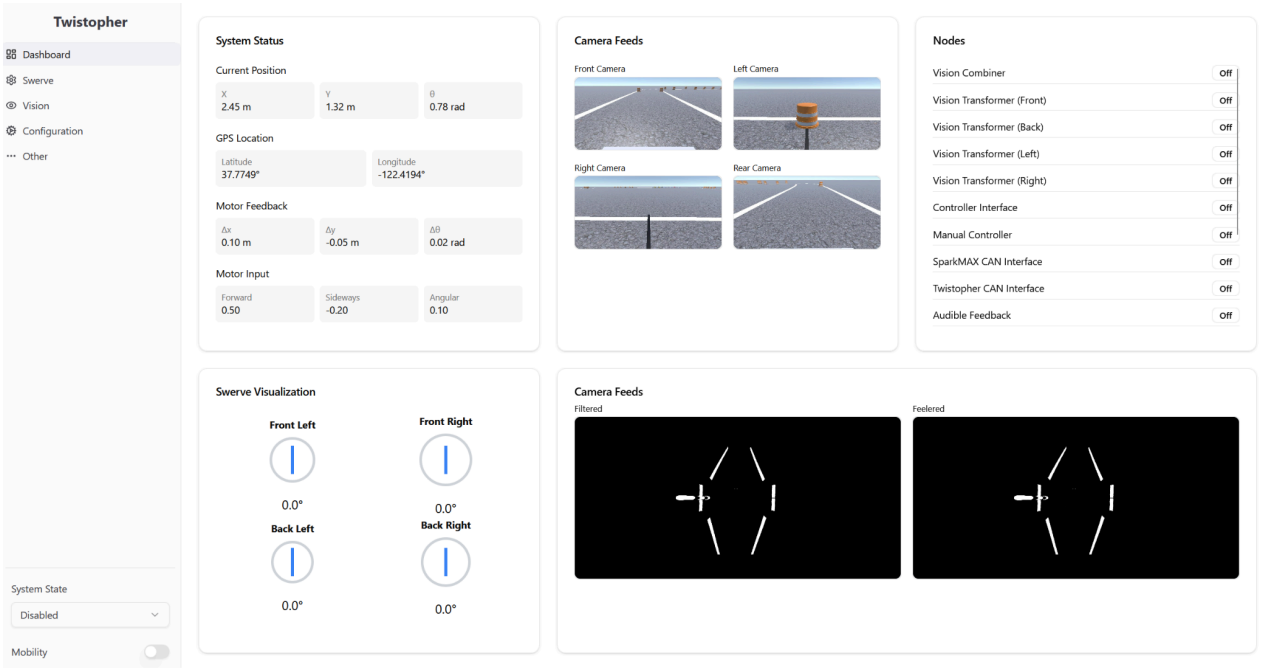
Pseudo-inverse Swerve Transform Matrix

$$\begin{bmatrix} 1 & 0 & -r_{y,FL} \\ 0 & 1 & r_{x,FL} \\ 1 & 0 & -r_{y,FR} \\ 0 & 1 & r_{x,FR} \\ 1 & 0 & -r_{y,BL} \\ 0 & 1 & r_{x,BL} \\ 1 & 0 & -r_{y,BR} \\ 0 & 1 & r_{x,BR} \end{bmatrix}^{-1} \times \begin{bmatrix} \hat{v}_{x,FL} \\ \hat{v}_{y,FL} \\ \hat{v}_{x,FR} \\ \hat{v}_{y,FR} \\ \hat{v}_{x,BL} \\ \hat{v}_{y,BL} \\ \hat{v}_{x,BR} \\ \hat{v}_{y,BR} \end{bmatrix} \rightarrow \begin{bmatrix} \hat{v_x} \\ \hat{v_y} \\ \hat{\omega} \end{bmatrix}$$

Measured Module States → **Measured Robot State**

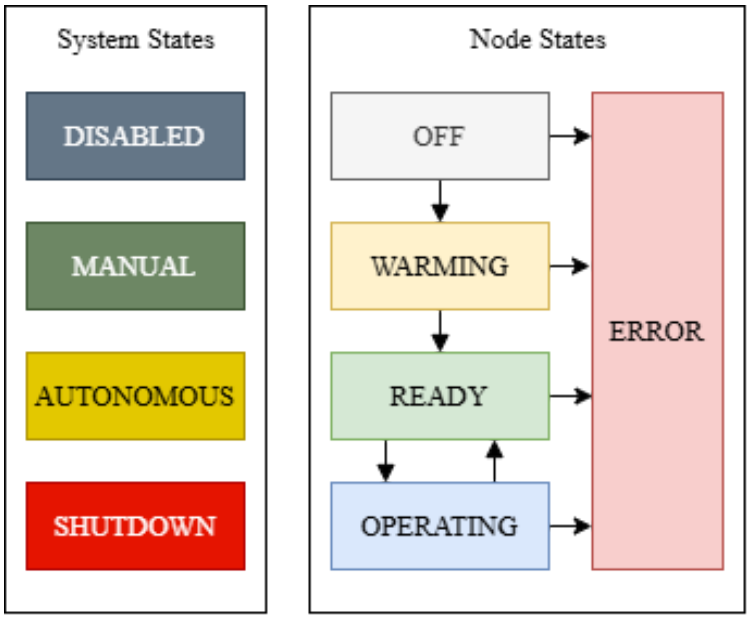
6.6 Configuration and GUI

Configuration of tunable parameters is handled via a custom web-based graphical user interface (GUI) running as a ROS node on the Intel NUC. This makes tuning the robot on-the-fly much faster and easier than before, as it does not require a restart and recompile of the robot code for changes to take effect. The GUI is also very useful for debugging *Twistopher*, and displays useful information such as the robot state, the state of each individual ROS node, and live sensor readings.



6.7 State Machine

All ROS nodes in our system inherit from a custom base class that provides shared functionality, including management of the global system state, mobility control, local device state, and configuration. The global system state defines the robot's overall mode (autonomous, manual, disabled, or restart) and determines how the robot behaves at a high level. Mobility refers to the robot's ability to move autonomously and serves as a safety mechanism, ensuring all subsystems are verified and functioning correctly before any autonomous actions begin. In addition to these shared states, each node maintains its own local state, which can be off, ready, or operating. This local state indicates whether the node is inactive, prepared to operate but awaiting the correct conditions, or actively operating, such as processing sensor input or controlling hardware. This architecture allows for clear monitoring of both global behavior and individual node activity.



6.8 Logging

While real-time data visualization is essential during testing, both in competition and practice, the most valuable information is often generated behind the scenes as the robot operates. After every "run," defined as the period between the start and end of a manual or autonomous system state, a comprehensive log file is created. This log captures nearly all data collected during the run, including sensor readings (such as camera feeds, GPS, IMU, and encoder feedback), diagnostic metrics like distance to waypoints, current navigation targets, swerve drive states, and other debug information. To make this data actionable, we developed a graphical user interface that organizes and visualizes the information into logical categories, such as camera views, swerve drive visualizations, and more. This post-run analysis enables us to investigate issues that are difficult or impossible to catch in real time, such as fine-grained movement precision errors, communication latency, and other subtle system behaviors.

7. Cyber Security Analysis

7.1 NIST RMF Process

Security vulnerabilities in self-driving vehicles pose significant risks, not only to passengers but also to pedestrians, other drivers, and surrounding infrastructure. To effectively mitigate these threats, it is essential to follow a comprehensive, systematic approach to security analysis and protection. One proven methodology is the NIST Risk Management Framework (RMF), which provides a structured process for identifying, assessing, and managing risks throughout a system's life cycle. By leveraging the NIST RMF, organizations can methodically evaluate the threats and vulnerabilities inherent in autonomous vehicle systems, prioritize risks based on potential impact, and implement appropriate safeguards. This ensures that all critical aspects of security are addressed, from hardware and software to network communications

and sensor integrity, ultimately enhancing the safety and resilience of self-driving vehicles in real-world environments.

7.2 Vehicle Threats

Subsystem	Threat Description	Confidentiality	Integrity	Availability
GPS Waypoints	GPS Waypoint tampering could lead the vehicle to drive in unintended ways.	Low	High	High
Remote E-stop	Malicious interference could disable or trigger the e-stop.	Moderate	High	High
Onboard Computer	Malware or unauthorized access could compromise system control.	High	High	High
Odometry	Sensor data tampering can result in incorrect state estimation.	Low	High	Moderate
Git Repo	Unauthorized access could inject malicious code into the vehicle's software.	High	High	Moderate

7.3 Cyber Controls and Implementation

Subsystem	Control	Implementation
GPS Waypoints	SI-4 System Monitoring	Display GPS waypoints to the user and validate using range checks.
Remote E-stop	AC-3 Access Enforcement	Authenticate and encrypt the e-stop radio channel
Onboard Computer	SC-28 Protection of Information at Rest	Use secure boot, enable SSH authentication, and disable passwords.
Odometry	SI-7 Software, Firmware, and Information Integrity	Validate sensor input using range checks and sensor fusion.
Git Repo	CM-5 Access Restrictions for Change	Enforce code access control with role-based permissions.

8. Vehicle Analysis

8.1 Lessons Learned

The custom swerve modules required significant mechanical and machining experience to design and manufacture, and are beyond the scope of anything our team has done in the past. We learned a lot about the engineering process when approaching a new problem, especially one that is complex not only mechanically but also in software and electrically. One of the key ideas we learned was to prototype early and often, as many things that were possible in the mechanical design CAD were hard to assemble once manufactured.

8.2 Failure Points and Resolutions

Category	**Failure Point**	**Cause**	**Resolution**
Mechanical	Nuts become loose and fall off	Vibrations that occur due to driving	Use Loctite and Nyloc nuts.
Electrical	Loss of connection with the wireless e-stop	E-stop remote goes out of range, or an unrecoverable error occurs	The e-stop Receiver does not receive a heartbeat on time and stops the robot.
Electrical	Encoder feedback becomes unstable	Damage to connections or mechanical separation	Particle filter accounts for reasonable losses in encoder feedback
Software	Objects are misidentified	Changing weather conditions can affect HSV threshold	An automatic hsv calibration algorithm runs before the robot is allowed to drive.
Software	Loss of incoming sensor data	Unaccounted errors or cables coming loose	Nodes monitor the states of other nodes and will stop running if conditions are not met.

8.3 Safety, Reliability, and Durability

Twistopher was designed from the ground up to be safe, reliable, and durable to any conditions. Through dozens of test runs in numerous scenarios such as sensor loss, changing weather conditions, failure to maintain connection with the estop remote, etc. we were able to determine that the robot reliably activates all of its safety measures. As described above, this includes stopping operation on sensor loss, adjusting hsv thresholds, or automatically cutting power to the motors to prevent further operation without human intervention.

8.4 Performance Metrics

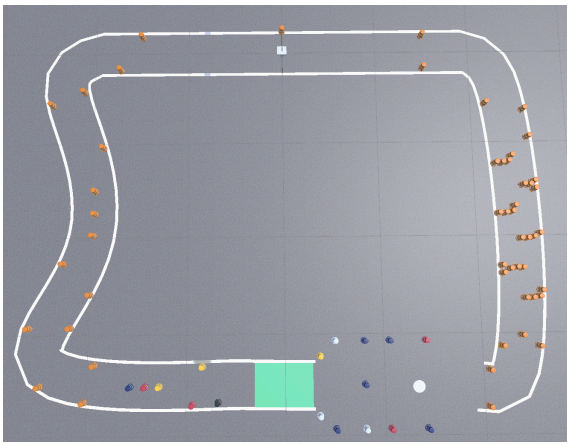
Parameter	Requirements Value	Measured Value
Course Completion Time	<5 minutes	3 minutes, 34 seconds (simulated)
Obstacle Detection Range	>2 meters	2.3 meters
Obstacle Reaction Time	<250 ms	47ms
Remote E-stop Range	>100ft	>500ft
Navigation Accuracy	<1 meters	<1 meters
Battery Life	>5 minutes	30 minutes
Ramp Ascension	Successful	Successful

8.5 Software Testing, Bug Tracking, and Version Control

The software subteam utilized Git and GitHub to store and version control our codebase. By utilizing branches for each new software feature, those features could be isolated and independently tested before being merged into the main codebase. The main codebase as well as each branch is automatically tested weekly for code compilation and dependencies.

8.6 Simulation

The team continues to use SCR's custom Scrabby simulation platform. This year, it has seen improvements in both graphics and performance. We added randomized barrel colors and placements to add variation to the course that ensures our robots can traverse it in any configuration. Similar to previous years, Scrabby has the ability to automatically randomize variables like sensor noise, robot direction, and some barrel placement locations, which allows for confidence in consistent vehicle performance.



9. Unique AutoNav and Self Drive Software

Twistopher will be competing in both AutoNav and Self Drive, and thus has configuration options to run either the Self Drive course or AutoNav course. AutoNav and Self Drive both use many shared features and configurations for tasks like lane keeping and obstacle avoidance. However, in order to perform Self Drive tasks, a state machine was added that allows for individual tasks to be run as well as groups of tasks, such as the full course.

Many of the features from AutoNav such as lane keeping and obstacle detection are applicable to Self Drive and thus are used in the exact same fashion. However, due to more complex functions such as lane

changing, merging, stop sign detection, and parking, *Twistopher* has also seen numerous features added specifically for self drive.

10. Performance Assessment

As of submitting this report, *Twistopher* is meeting all expectations. Through extensive testing, the swerve modules have held up to conditions far worse than will be observed on the competition grounds and respond to control inputs in expected times.