IGVC 2026 Design Report

Lawrence Technological University

Team: rACTor

Robot: rACTor

Team Captains	Bru Lamarca	blamarcag@ltu.edu
	Deepak Goud Nalla	dnalla@ltu.edu
Members	Arun Tejaswi	narunteja@ltu.edu
	Jacob Hallett	jhallett@ltu.edu
	Jurius Azar	jazar@ltu.edu
	Ilyarzhan Assimov	iassimov@ltu.edu
	Siva Ganesh Varma Paidi	spaidi@ltu.edu
	Shaik Mohinpasha	mshaik@ltu.edu
	Manjunadh Reddy	mkommired@ltu.edu
	Srii Akhillessh	agaddam@ltu.edu
Faculty Advisors	Chian-Jin “CJ” Chung	cchung@ltu.edu
	Vijay John	vjohn@ltu.edu
CS Robotics Lab manager	Devson Butani	dbutani@ltu.edu

1. Team presentation, workflow, and responsibilities

1.1. Team presentation

Figure 1: Team **rACTor**[1], from left to right: Vijay John, Srii Akhillessh, Manjunadh Reddy, Siva Ganesh Varma Paidi, Shaik Mohinpasha, Devson Butani, Jacob Hallet, Jurius Azar, Illyarzhan Assimov, Aarun Tejaswi, Chian-Jin Chung, and Bru Lamarca.

1.2. Team workflow

The team consists of ten students, graduates, and those still pursuing their degree. They had their first meeting on February 5, 2026, and have had recurring meetings every Thursday since then. In those meetings, they discussed task updates, new options for upgrading the rACTor, and general topics. Each member has their own tasks, assigned based on skills such as experience and character, as well as a general understanding of the robot's design, workflow, programming, and hardware. The team's goal was to significantly improve the robot, which had been reused from the previous year.

To foster a cooperative environment, we implemented a communication system using simple tools like Discord, where members can notify others of their availability to facilitate meeting times for development, testing, or debugging. These meetings are designed to be held by two to four members, agilitating the workflow and preventing conflicts in git and in testing.

Our team received a robot from the former year's team, the winners of the IGVC competition, so the stakes were very high. The process during these months has been the following:

---

1 Pronounced /riˈæktər/; an acronym for **r**e- (again) **A**utonomous **C**amp,us **T**ransp**or**t.

Detection of issues → Assignment of task to a member or group of members → Debugging to understand the problem → Programming to solve the issue → Testing the solution → Applying it

1.3. Team responsibilities

Name	Degree program	Professional Responsibilities	Hours worked
Bru Lamarca	B.S Mathematical Sciences	Team Captain	150
Deepak Goud Nalla	M.S. Data Science	Team Captain & Mechanical/Systems Architecture Design	80
Arun Tejaswi	PhD Computer Science	Autonomous Navigation (Nav2) & Version Control Management	80
Jacob Hallett	M.S. Computer Science	Computer Vision, Nav2 Integration, Braking Control Systems, & Spatial Lane Mapping	120
Jurius Azar	Dual enrollment	Embedded Systems & Low-Level Firmware Engineering	80
Siva Ganesh Varma Paidi	M.S. Computer Science	Sensor Integration & Precision Time Synchronization	90
Shaik Mohinpasha	M.S, Computer Science	Sensor Hardware Validation & Multi-Sensor Temporal Alignment	80
Manjunadh Reddy	M.S. Computer Science	Multi-Sensor Calibration (Camera/LiDAR) & Computer Vision Pipelines	80
Srii Akhillessh	M.S. Computer Science	Perception Engineering & Vision-Based Lane Detection	90
Ilyar Assimov	M.S. Computer Science	Autonomous Navigation Stack (Nav2) & System Design	80

Figure 2: Team responsibilities

2. Vehicle Architecture

rACTor is a purpose-built autonomous mobile platform co-sponsored by Dataspeed and individual contributors. The vehicle integrates a 3D LiDAR unit, an RTK-capable GNSS receiver, a vision sensor, several inertial measurement units, and a fully redesigned omnidirectional drive system. All sensor and actuator modules communicate through a Raspberry Pi 5, which links to a GPU-equipped laptop via a wired Ethernet connection on the robot's internal network. An onboard portable power supply delivers energy to the entire system. Over successive competition cycles, the team has substantially overhauled the hardware inherited from earlier ACTor generations — most notably in LiDAR, positioning, the Breaking System, and compute infrastructure. Figure 2 presents a simplified view of the organization of the major software and hardware components.

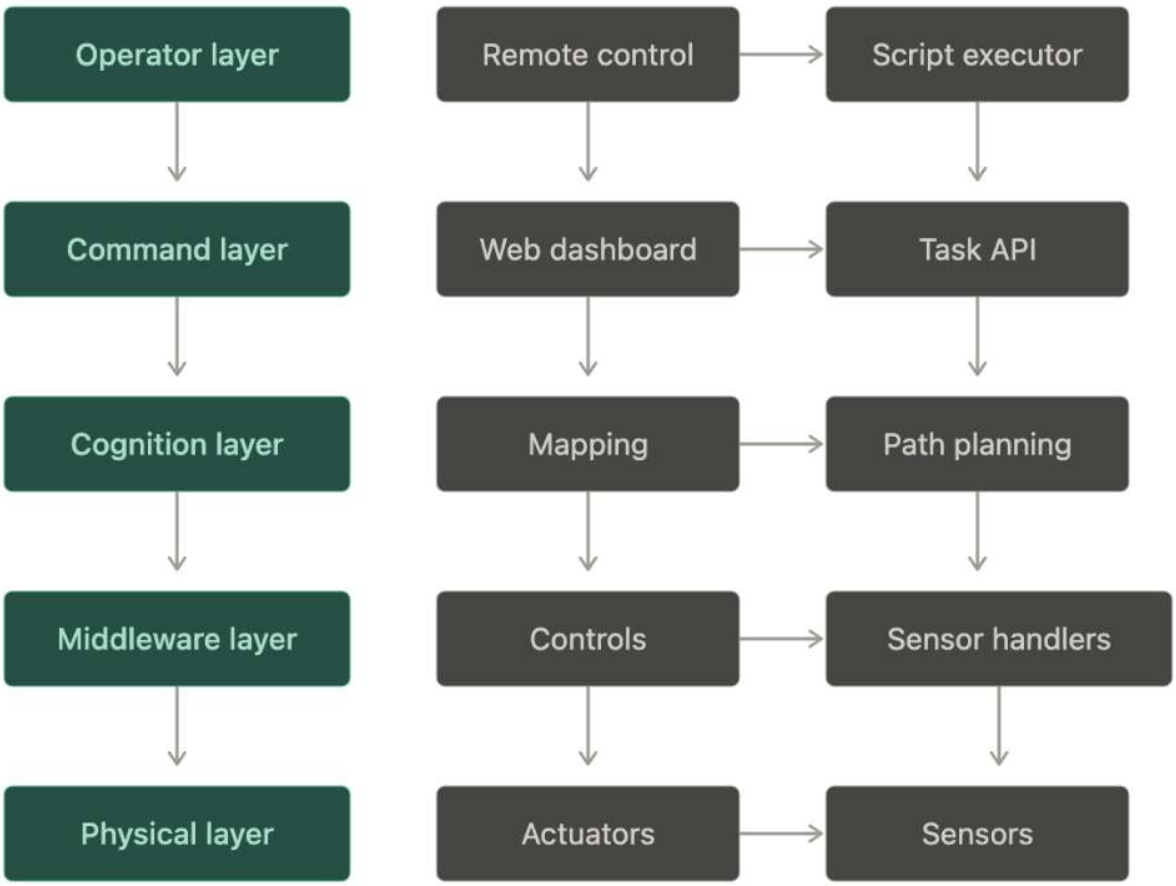

Figure 3: High-level Component Layers

2.1. Safety as a Core Principle

Reliable operation is the foundational priority behind every design decision on rACTor. The system continuously monitors the health of each architectural tier and responds with appropriate

countermeasures when anomalies arise. A dedicated emergency mechanism — the Big Red Button (BRB) — together with a wireless controller, enables operators to invoke an immediate hardware-level halt. Internally, rACTor's software watchdog continuously inspects subsystem states across all tiers and can autonomously initiate a stop if it detects a malfunctioning sensor, a non-responsive actuator, a software exception, or a communication breakdown.

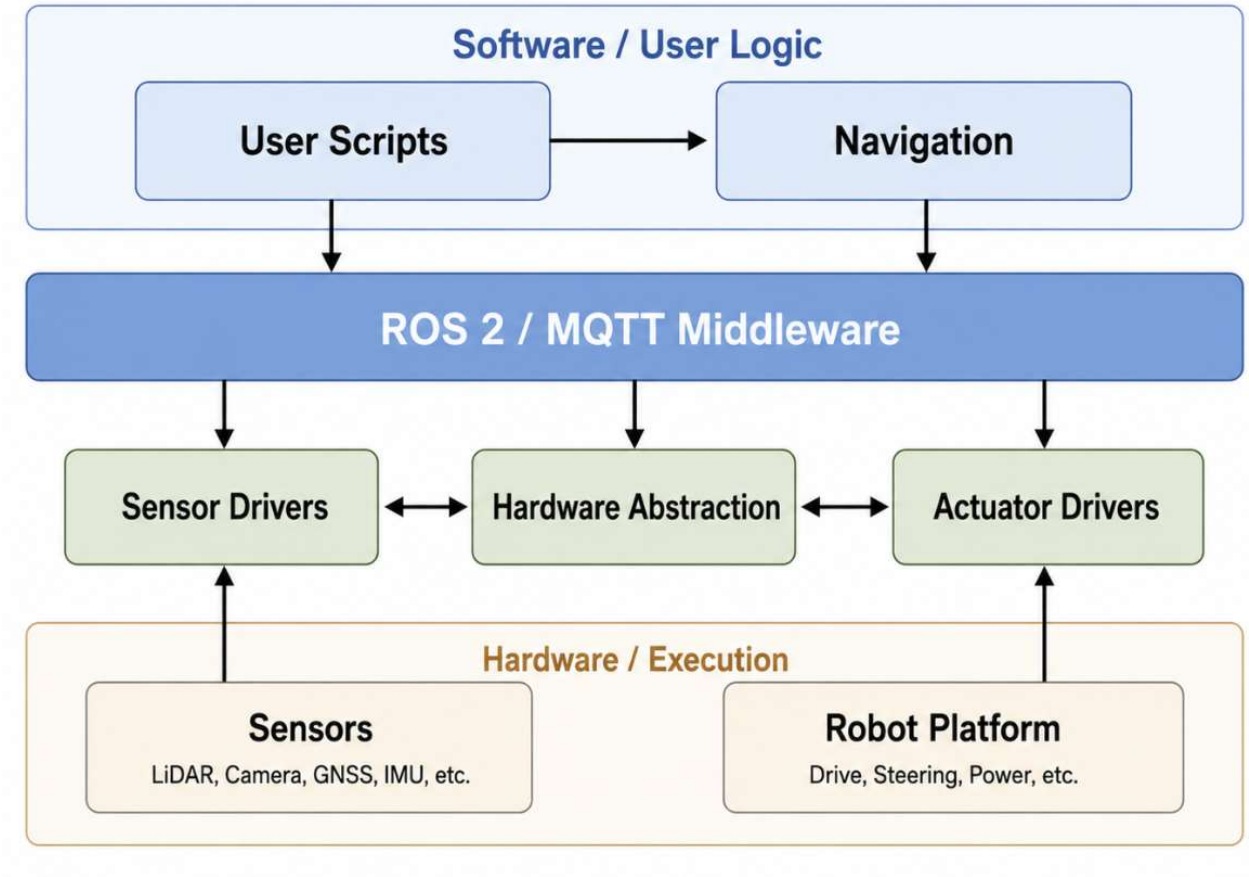

Figure 4: Operating Architecture

2.2. Software Stack and Navigation Framework

This competition cycle introduced substantial software refinements. rACTor runs on a ROS2 Jazzy laptop with Ubuntu 24.04 and leverages ROS2 Navigation to provide a modular, extensible foundation adaptable to diverse mission profiles. ROS2 Navigation provides built-in support for sensor fusion workflows, including Simultaneous Localization and Mapping (SLAM) and dynamic path computation through complex terrain. The omnidirectional chassis supports two distinct drive strategies — "Double-Ackermann" and "Fixed Heading" — selectable based on the task at hand. Mission-specific behaviors are encoded in Python scripts that serve as high-level task executors, each targeting an individual IGVC challenge objective. A browser-based control panel on the robot's local network provides real-time access to sensor telemetry, script management, and actuator controls.

3. Improvements since last year's robot and other ACTors

This year's development focused on three areas: upgrading the sensor suite for more reliable perception, replacing the odometry and navigation pipeline with Ackermann-aware components, and adding real-time sensor fusion and collision safety layers. Together, these changes transform the rACTor navigation stack from a generic grid-based planner into a kinematically-aware system that respects the robot's double-Ackermann steering constraints.

3.1. Improvements since last year

- LiDAR Upgrade from Ouster OS1 to Velodyne 3D LiDAR. The Ouster OS1 used in the 2025 competition has been replaced with a Velodyne 3D LiDAR unit [TODO: reason for swap]. The new sensor publishes full-surround PointCloud2 data on the /velodyne_points topic and integrates directly with the KISS-ICP odometry pipeline and the pointcloud_to_laserscan converter used by SLAM Toolbox.

- IMU Upgrade from Yahboom CMP10A to Bosch LSM6DSOX. The previous Yahboom CMP10A IMU has been replaced with a Bosch LSM6DSOX 6-axis inertial measurement unit. The LSM6DSOX provides gyroscope and accelerometer data filtered through the imu_filter_madgwick node before entering the EKF pipeline. Unlike magnetometer-based orientation sensors, the LSM6DSOX avoids the 90–180° heading drift caused by magnetic interference in indoor environments, providing stable angular velocity measurements for yaw rate estimation.

- LiDAR Odometry with KISS-ICP. The previous LiDAR Inertial Odometry (LIO) implementation has been replaced with KISS-ICP (Keep It Small and Simple Iterative Closest Point), a scan-to-map LiDAR odometry engine that produces pose estimates at approximately 10 Hz. KISS-ICP computes the robot's displacement by registering each incoming point cloud against a local map built from recent scans, publishing the result as the odom→base_footprint TF transform. The team maintains a custom fork with indoor-specific tuning: max_range is limited to 10 m, and voxel_size is set to 0.10 m to ensure stable pose estimation in confined hallway environments where distant points introduce noise.

- Sensor Fusion via Extended Kalman Filter (EKF). The robot_localization package now fuses the KISS-ICP pose (x, y, yaw) and the LSM6DSOX angular velocity into a unified /odometry/local estimate at 15 Hz. This two-source fusion smooths odometry noise from both sensors and fills temporal gaps when one source temporarily degrades. IMU absolute orientation is explicitly turned off in the EKF configuration to prevent indoor magnetic interference from corrupting the fused estimate—only the gyroscope's angular velocity is used, providing yaw rate smoothing without magnetometer dependency.

- Ackermann-Aware Global Planner (SmacPlannerHybrid). The grid-based global planner has been replaced with Nav2's SmacPlannerHybrid, which performs Hybrid A* search in (x, y, θ) configuration space using Dubins curves. This planner respects the robot's 0.75 m minimum turning radius—derived from the double-Ackermann steering geometry (0.711 m wheelbase, 45° maximum steering angle)—and generates kinematically feasible curved paths. The previous grid-based approach produced paths

with sharp right-angle turns that the Ackermann steering could not execute, causing the controller to oscillate as it attempted to follow infeasible trajectories.

- Regulated Pure Pursuit Controller (RPP). The local controller has been replaced with Nav2's Regulated Pure Pursuit Controller, designed specifically for non-holonomic robots that cannot rotate in place. The controller is tuned for a 0.8 m/s indoor cruise velocity with a 1.2 m lookahead distance, providing smooth path tracking. Ackermann-specific constraints are enforced: use_rotate_to_heading is disabled (the robot cannot spin in place), allow_reversing is disabled (forward-only motion), and the minimum approach velocity is set to 0.3 m/s to ensure the robot maintains enough speed to overcome indoor floor friction near the goal.

- Velocity Smoother and Collision Monitor. A two-stage velocity post-processing pipeline has been added between the controller output and the motor control interface. The velocity smoother enforces acceleration limits (0.5 m/s^2 acceleration, −2.5 m/s^2 deceleration) to prevent mechanical stress on the drivetrain. Downstream, the collision monitor reads the raw 3D point cloud from the Velodyne LiDAR and applies a footprint-based model with a 1.2-second time-to-collision threshold, providing a real-time safety layer that operates independently of the costmap and can halt the robot even if the costmap fails to detect a close obstacle.

- Spatio-Temporal Voxel Layer (STVL) in Local Costmap. The local costmap now uses a Spatio-Temporal Voxel Layer to represent obstacles as time-decaying 3D voxels with 0.1 m resolution and a 1.0-second decay window. Unlike the previous static obstacle layer, STVL clears obstacles from the costmap within one second of the sensor, no longer observing them. This temporal awareness enables the robot to navigate dynamic environments—such as hallways where people walk past—without treating stale sensor readings as permanent barriers that block path planning.

3.2. Improvements regarding other ACTors

Figure 5 summarizes the key subsystem changes between the 2025 and 2026 competition entries.

Area	2025	2026
LiDAR	Ouster OS1	Velodyne 3D
IMU	Yahboom CMP10A	Bosch LSM6DSOX
LiDAR Odometry	LIO	KISS-ICP (indoor-tuned fork)
Sensor Fusion	Raw odometry only	EKF (robot_localization, 15

		Hz)
Global Planner	Grid-based (NavFn A*)	SmacPlannerHybrid (Hybrid A*, Dubins)
Local Controller	Basic FollowPath	RegulatedPurePursuitController
Velocity Processing	Direct cmd_vel to motors	Velocity Smoother + Collision Monitor
Costmap Obstacles	Static obstacle layer	STVL (time-decaying 3D voxels)
Object Detection	YOLOv12	[TODO]
Steering	Double Ackermann + Fixed Heading + Rotation mode	Unchanged

Figure 5: Key Subsystem Comparison, 2025 vs. 2026

End-to-end autonomous navigation now works reliably: SLAM Toolbox builds a live map, SmacPlannerHybrid plans Ackermann-feasible paths, and the RPP controller tracks those paths with clean stops at each goal. The velocity smoother and collision monitor provide layered safety between the planner and the physical drivetrain. Remaining challenges include the inability to rotate to a specific goal heading via Nav2 (the double-Ackermann steering can physically rotate in place, but Nav2 does not trigger the control tower's drive-mode switch—currently worked around by accepting any final heading), and residual path oscillation in tight corridors that, while reduced by the switch to curved Hybrid A* paths, has not been fully eliminated.

4. rACTor Design And Hardware Description

The mechanical design of the rACTor is guided by a hardware philosophy focused on rapid prototyping, symmetry, and simplicity. Our primary motivation for implementing four-wheel steering (4WS) was to explore a new approach for the team and to unify the steering control strategies required for both AutoNav and Self-Drive courses. While Self-Drive typically relies on Ackermann steering, this configuration would make navigating around barrels on the AutoNav course difficult. Alternatively, a differential-drive system would rely too heavily on wheel slip during lane following in Self-Drive. By adopting 4WS, we achieve a balanced solution: it maintains the benefits of Ackermann steering while Double-Ackermann (steering on both front and rear axles) delivers full maneuverability for complex navigation tasks. Additionally, rACTor features two extra driving modes: Fixed Heading, where all wheels have the same angle φ, which we use in some select moments to ease the task, as well as Rotation mode, which allows the car to turn along its center, removing the need for extra road real estate, allowing sharper turns and ease of molility.

In addition to the unique steering architecture, this year's mechanical development focused on two key subsystems: an active braking system and a modular weather protection enclosure. These additions were designed to improve operational safety and environmental robustness without compromising the system's modularity or serviceability.

Using standardized, readily available commercial off-the-shelf (COTS) components significantly reduced costs and development time. This approach enabled rapid design iterations and early identification of component clearances and structural design. SolidWorks (CAD) enabled the design of a constrained frame geometry and 3D-printed mounting for most components. The design process was kept simple and iterative, starting from the ground up, adding components only for functionality and ensuring they were well constrained.

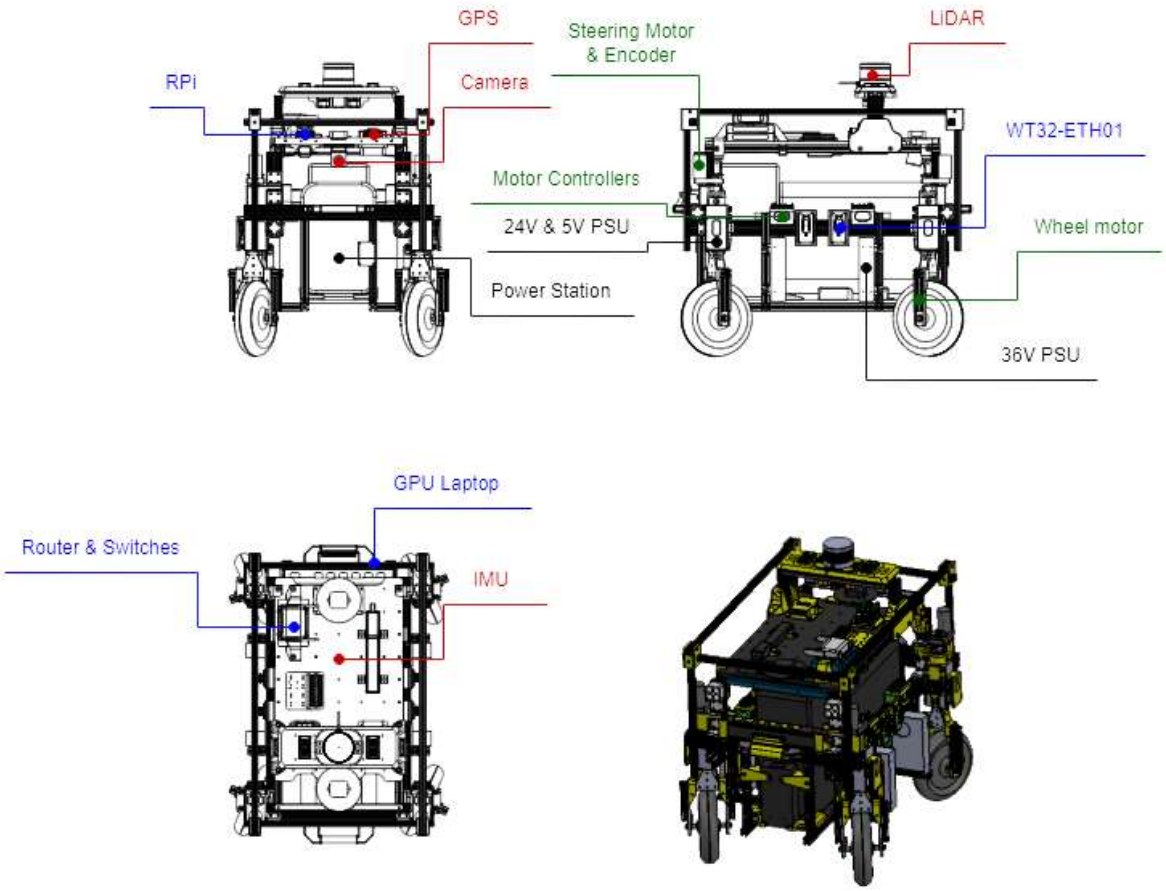

Figure 6: rACTor Mechanical Design and Component Layout. Sensors are the components colored in red, computers and connectivity in blue, controls in green, and power in black.

Building on the existing steering system, power station, and frame architecture, this year's primary mechanical focus was on integrating an electromechanical braking system and a lightweight protective enclosure. These additions introduced new design constraints related to mounting, cable routing, environmental protection, and load distribution, which influenced component placement and

overall frame refinement. The overall structure continues to follow the “exact constraint design” principle, ensuring the robot is neither over-constrained nor under-constrained. The frame design prioritizes structural integrity and symmetry to simplify fabrication, improve serviceability, and maintain balanced weight distribution. Utilizing standardized fasteners (primarily ¼-20 and M6) further supports simple assembly and rapid maintenance timelines.

4.1. Aluminum Extrusion Framework

Cut-to-order aluminum extrusions and repurposed extrusions form the backbone of the robot's structure. Different profile sizes were selected based on their torsional rigidity, load capacity, availability, and ease of assembly. This modular approach allowed for flexible mounting options and simplified the integration of additional components as the design evolved. The T-slotted aluminum extrusion system provided unprecedented tensile strength, vibration resistance, and load-bearing stability while enabling easy maintenance and part replacement.

4.2. Material Combinations

The design incorporates a strategic mix of materials to optimize performance while managing costs and lead times:

- Aluminum and steel for primary load-bearing structures
- Wood for static, non-critical structural elements
- 3D printed PETG for complex geometries and connector parts, leveraging PETG's high tensile strength and resilience

4.3. Drive System

The rACTor utilizes a Segway Ninebot Max G30P Electric Scooter Hub Motor (500W BLDC). This integrated wheel-motor combination delivers powerful, smooth movement at lower speeds while maintaining quiet operation on paved surfaces. The integrated tire design simplifies the overall mechanical assembly while ensuring reliable traction.

4.4. Steering Mechanism

The symmetrical steering assembly features an aluminum square tube with 3D-printed inserts that house steel ball bearings. A smaller square aluminum extrusion acts as the shaft, which slides into the bearings using a 3D-printed collar. This design creates a robust yet smooth steering action, with loads transferred from the shaft to the frame via steel bolts. The square profile prevents unwanted rotation while allowing necessary vertical movement.

4.5. Braking System

To enhance safety and control, the rACTor incorporates a mechanically actuated braking system on the front wheels. The system uses standard bicycle brakes coupled with NORJIN 12V Mini Electric Linear Actuators (1.2-inch stroke) to apply braking force. Each actuator is mounted in alignment with the brake lever mechanism, enabling precise and repeatable braking through linear extension.

This design provides several advantages:

- Linear actuators allow controlled braking force rather than natural deceleration from disengaging the drive system.
- COTS bicycle components simplify replacement and tuning

The braking system is integrated into the frame to preserve symmetry and minimize interference with steering articulation.

4.6. Weather Protection System

A modular weather cover was developed using corrugated plastic panels mounted to the frame via custom 3D-printed brackets. This enclosure protects sensitive electronics and wiring from rain, debris, and environmental exposure while maintaining airflow for thermal management.

Key design considerations include:

- Lightweight construction: Corrugated plastic minimizes added mass
- Ease of removal: Panels are mounted for quick access during maintenance
- Drainage and airflow: Openings and panel orientation prevent water accumulation and overheating

The mounting system uses an aluminum extrusion frame, enabling the enclosure to be easily reconfigured or expanded as needed.

4.7. Suspension

Although the steering column design supports the addition of a suspension system, we opted not to implement one due to the competition's scope and budget constraints. Instead, Noise, Vibration, and Harshness (NVH) management for electronics is achieved through soft mounts for vibration-sensitive components, thread lockers and washers for fasteners to prevent loosening, and strategic component placement to minimize vibration effects. This approach balances performance requirements with design simplicity while maintaining the option to add suspension in future iterations if necessary.

4.8. Weatherproofing

The outdoor-rated hub motor features built-in weatherproofing, while non-waterproof electronics are positioned higher in the frame to avoid water exposure. The frame design incorporates natural drainage paths and protective covers for sensitive components. PETG's excellent chemical resistance and UV light protection capabilities make it ideal for outdoor-exposed 3D-printed components. The external housing geometry is designed to direct water away from sensitive components and connection points. All external sensors feature waterproof housings or are installed with appropriate protective covers to ensure reliable operation in various weather conditions. This approach is further reinforced by the addition of the removable corrugated plastic enclosure, which provides a first layer of defense against environmental exposure while maintaining accessibility and modularity.

The electrical and power design of the rACTor prioritizes reliability, flexibility, and safety. The system architecture features a centralized power source with distributed power management for various subsystems. The electronics design also incorporates COTS components where appropriate, complemented by custom mounting solutions to meet specific performance requirements.

4.9. Power Source

The rACTor's core is an Anker SOLIX F2000 Portable Power Station featuring a 2kWh LiFePO4 (Lithium Iron Phosphate) battery that supports all types of power needs at 2kW max draw; pure sine wave 120VAC via 4×NEMA 5-20 and 1×NEMA TT-30, 12VDC via a 2xSAE J563 auxiliary outlets, and 2xUSB-A and 3xUSB-C PD ports.

This power station was selected for its high energy density, extended cycle life, and enhanced safety characteristics compared to traditional lithium-ion batteries. The LiFePO4 chemistry provides excellent thermal stability, reducing fire risks during operation and charging. All the main electronic components work on AC power, allowing them to be run outside the robot while the power station fast-charges 0 to 80% in just 1.4 hours. Moreover, the ability to directly add 1kW of Solar Panels opens the possibility of installing a charging dock that enables 100% renewable energy-powered deployment. Circuit protection is handled by the power station and by connected surge protectors for additional safety.

4.10. Distribution

The power station is connected to individual components via AC-DC and DC-DC converters, or via direct AC plugs, depending on the component's power requirements. In terms of power, rACTor can be physically separated into two parts: the driver suite (responsible for autonomous driving), which senses and thinks, and the robot base (solely responsible for motion and power storage), which acts. This unique feature allows independent maintenance and development of each part. For example, by simply removing the top driver suite and plugging it directly into a wall outlet, members can work on software development and testing without needing the robot base connected.

4.11. Robot Base

A dedicated 350W BLDC motor controller (commonly found as ebike controllers) drives each hub motor, and a separate 250W brushed DC controller operates the steering motor for each corner. These components are selected due to budget constraints and can be easily upgraded later on. For each corner, both motor controllers are driven by an ESP32-based board (WT32-ETH01) connected over Ethernet. Keeping all the analog signals to very short cable lengths allows individual corners to operate independently. This strategy was chosen to kickstart the firmware development concurrently with frame design and other component selection. Additionally, the linear actuators used for the braking system are powered via the 12VDC rail and controlled through the onboard microcontrollers, allowing integration with both manual ESTOP signals and autonomous control logic.

4.12. Driver Suite

A standalone set of sensors is directly connected to the Raspberry Pi 5 and a laptop with a discrete GPU via USB and an Ethernet switch. The following sensors are installed:

- Ouster OS1 LiDAR
    - Provides full-surround 3D point cloud data with high resolution and range.
- RouteCAM_CU22_IP67 IP Camera
    - A PoE camera sensor that captures high-resolution images with low latency.
- Yahboom CMP10A Inertial Measurement Unit (IMU)
    - Tracks robot orientation and motion using gyroscopes (yaw), accelerometers (roll and pitch), and a magnetometer.
- ArduSimple simpleRTK2B GNSS
    - 2x Ublox ZED-F9P multiband receiver supplies a GPS fix with RTK accuracy.

4.13. ESTOP

A wireless estop using an RC controller and receiver (Turnigy i6S TX and iA6B RX) is connected in series with mechanical estop buttons, which drive active-on relays that directly connect to the motor controller brake line. Breaking the loop anywhere applies instant braking force via the BLDC motor controllers and the actuator-driven mechanical braking system, providing additional stopping capability.

5. rACTor software

5.1. Camera and LiDAR Fusion

The perception architecture of Team rACTor was designed to combine semantic scene understanding with precise spatial awareness for autonomous navigation within IGVC environments. Cameras provided high-resolution visual information for lane detection and object classification, while LiDAR delivered accurate depth estimation and 3D geometric measurements. Individually, each sensor had limitations: cameras struggled with depth estimation and lighting variations, while LiDAR lacked semantic understanding of detected obstacles. By combining both sensing modalities, rACTor achieved more reliable obstacle detection, localization, and tracking during autonomous navigation and obstacle-avoidance tasks.

The complete perception stack was implemented using ROS2 Jazzy middleware, enabling modular communication between camera, LiDAR, inference, fusion, and planning nodes through synchronized ROS topics and TF transforms.

5.2. Camera Perception Pipeline

5.2.1. Camera Integration

rACTor used the RouteCAM IP camera, integrated with ROS2 Jazzy via the /image_raw topic. The camera continuously streamed high-resolution RGB frames to the onboard NVIDIA GPU for real-time inference. Image preprocessing included resizing, normalization, RGB conversion, tensor preparation, and batch optimization to maintain stable inference latency under varying environmental conditions.

The camera feed was visualized and debugged through RViz2 and OpenCV visualization tools during both simulation and field testing.

5.2.2. Object Detection and Model Development

The vision pipeline used YOLO-based object detection models trained using the Hugging Face ecosystem along with custom annotated datasets collected from simulation environments and outdoor IGVC testing sessions.

Training datasets included:

- Traffic cones
- Potholes
- Mannequins
- Pedestrians
- Lane boundaries
- Static environmental obstacles

Transfer learning techniques were applied using pretrained YOLO backbones to accelerate convergence and improve generalization performance under varying lighting conditions, shadows, grass textures, and partial occlusions.

The trained models were exported for real-time ROS2 deployment and optimized for GPU inference acceleration.

Reference architecture (not directly used as-is): https://huggingface.co/yaraa11/road-lane-semantic-segmentation-unet-resnet50

This model was used only as a **reference baseline for lane segmentation design ideas**, while the final deployed system used **custom-trained models on independently collected datasets**, later combined into a unified perception pipeline.

Confidence threshold filtering and non-maximum suppression (NMS) were applied to reduce duplicate detections and minimize false positives during runtime inference.

5.2.3. Lane Detection and Tracking

Lane detection within rACTor combined OpenCV-based image processing with neural-network-assisted feature extraction. The pipeline utilized:

- Canny edge extraction
- Perspective transformation
- Region-of-interest masking
- Morphological filtering
- Polynomial lane fitting

These operations enabled robust real-time lane boundary estimation. The lane tracking subsystem continuously updated steering guidance commands and generated navigable path estimates for autonomous motion control during Self-Drive and AutoNav operations.

Temporal smoothing filters were additionally applied to reduce lane jitter caused by lighting fluctuations and terrain irregularities.

5.2.4. Camera Hypothesis Generation

Following inference, the perception node generated 2D object hypotheses consisting of:

- Bounding box coordinates
- Semantic class labels
- Detection confidence scores
- Timestamp metadata

These detections were published through the /camera_hypotheses topic and consumed by downstream fusion nodes. The camera subsystem provided the semantic understanding required to distinguish between cones, mannequins, pedestrians, potholes, and unknown obstacles during autonomous navigation.

5.3. LiDAR Perception Pipeline

5.3.1. LiDAR Integration and Point Cloud Processing

rACTor utilized the Velodyne VLP-16 LiDAR sensor publishing PointCloud2 messages through the /velodyne_points topic. Incoming point clouds were processed using the Point Cloud Library (PCL) within ROS2.

Preprocessing operations included:

- Voxel grid downsampling
- Region-of-interest cropping
- Pass-through filtering
- Statistical outlier removal

These filtering stages significantly reduced computational overhead while preserving the obstacle geometry necessary for reliable real-time operation.

5.3.2. Ground Removal and Clustering

Ground segmentation was performed using RANSAC-based plane estimation to remove planar terrain surfaces from the environment. After ground removal, Euclidean clustering and DBSCAN-based clustering grouped neighboring points into obstacle clusters representing physical objects around the robot.

Each cluster corresponded to obstacles such as:

- Cones
- Pedestrians
- Barriers
- Environmental debris
- Unknown obstacles

Cluster filtering thresholds were tuned experimentally during field testing to improve robustness against sparse point cloud noise.

5.3.3. Distance Estimation and 3D Bounding Boxes

Obstacle localization was performed using centroid estimation and 3D bounding box generation from clustered LiDAR points. Each cluster produced:

- Position estimates
- Object dimensions
- Orientation estimates
- Distance measurements

Principal Component Analysis (PCA) was used to estimate the dominant orientation axes for obstacle alignment and path-planning integration.

The resulting 3D object hypotheses were published through the /lidar_objects_3d topic for sensor fusion processing.

5.3.4. Camera–LiDAR Calibration

Intrinsic camera calibration was performed using checkerboard-based calibration procedures to estimate focal lengths, distortion coefficients, and optical parameters. Extrinsic calibration aligned the LiDAR coordinate frame with the camera frame using rotation and translation matrices generated through ROS2 TF transforms.

This calibration process enabled projecting LiDAR points into image coordinates using homogeneous transformation matrices and camera projection equations, thereby enabling spatial association between 2D image detections and 3D LiDAR geometry.

5.4.

5.4.1. Camera–LiDAR Fusion Pipeline

The fusion node synchronized /camera_hypotheses and /lidar_objects_3d using ROS2 timestamp-based message synchronization. LiDAR points were projected into the image plane using intrinsic and extrinsic calibration matrices.

Object association was performed using:

- Bounding box overlap
- Euclidean proximity checks
- Intersection-over-Union (IoU) matching
- Confidence-based filtering

Matched detections generated fused 3D obstacle representations containing:

- Semantic labels
- Confidence scores
- Distance estimates
- Spatial orientation
- Object dimensions

The fusion pipeline reduced false positives commonly observed in standalone camera systems while simultaneously improving semantic interpretation unavailable in standalone LiDAR processing.

Fused objects were visualized in RViz2 for debugging and validation during both simulation and outdoor field testing.

5.4.2. Results and Benefits for rACTor

The final multimodal perception architecture enabled rACTor to detect, classify, localize, and track multiple obstacles simultaneously in real time. Compared to standalone perception approaches, the fused system improved obstacle reliability, depth estimation accuracy, and navigation stability during autonomous operation.

The system successfully identified:

- Cones
- Potholes
- Pedestrians
- Mannequins
- Lane boundaries
- Static obstacles

during simulation and outdoor testing scenarios.

The fusion architecture improved collision avoidance reliability while maintaining stable autonomous navigation performance under varying environmental conditions, including uneven terrain, shadows, partial occlusions, and changing illumination.

Despite these improvements, several limitations remained:

- Reduced camera performance under extreme lighting conditions
- Sparse LiDAR returns at long distances
- Increased computational load during dense obstacle scenarios
- Calibration sensitivity between the camera and LiDAR frames

Future improvements include temporal multi-object tracking, expanding sensor redundancy, and deploying transformer-based perception models for enhanced scene understanding.

5.5. Results, Fusion Benefits, and Nav2 Integration

The final multimodal perception architecture enabled Team rACTor to detect, classify, localize, and track multiple obstacles simultaneously in real time. Compared to standalone perception approaches, the fused system improved obstacle reliability, depth estimation accuracy, and navigation stability during autonomous operation. The fusion pipeline reduced false positives while improving semantic interpretation and spatial consistency across dynamic IGVC environments.

During the early development stages of rACTor, lane detection was implemented using traditional OpenCV-based HSV masking. Separate HSV masks were developed for white lane markings and yellow lane boundaries commonly observed within IGVC environments. The white HSV mask extracted high-intensity lane regions under standard illumination conditions, while the yellow HSV mask improved lane segmentation performance in areas affected by shadows, faded paint, grass reflections, and uneven terrain coloration. These segmented outputs were combined with Canny edge extraction, region-of-interest masking, and contour filtering to generate preliminary lane boundary estimates for autonomous navigation.

As the perception architecture evolved, the HSV-based lane extraction pipeline was integrated with neural-network-assisted perception and later combined with LiDAR fusion for improved environmental understanding. This transition significantly improved robustness under varying outdoor lighting conditions, partial occlusions, and dynamically changing terrain surfaces encountered during field testing. Temporal smoothing filters and polynomial lane-fitting algorithms

were also applied to minimize lane jitter and improve trajectory stability during Self-Drive and AutoNav operations.

The fused perception output was directly integrated with the Nav2 autonomous navigation stack within rACTor. Camera–LiDAR fusion provided accurate obstacle positions, free-space estimation, lane boundary awareness, and semantic obstacle classification to the Nav2 global planner and local planner modules. This enabled rACTor to continuously generate safer navigation trajectories while dynamically adapting to environmental changes during autonomous operation.

The Nav2 global planner utilized fused environmental representations to compute collision-free traversal paths throughout the operating environment. Simultaneously, the Nav2 local planner continuously updated short-range trajectory decisions using real-time fused perception outputs. Since the fusion pipeline produced semantically classified 3D obstacle representations rather than isolated raw sensor measurements, rACTor achieved improved obstacle localization accuracy, smoother path generation, and more stable trajectory tracking.

The fusion system further enhanced Nav2's local and global costmap generation by directly inserting semantically classified 3D objects into the planning framework. Cones, mannequins, pedestrians, potholes, lane boundaries, and static environmental barriers were represented with improved positional accuracy, spatial orientation, and confidence estimation. This allowed rACTor to more effectively distinguish between traversable free space and hazardous regions during obstacle avoidance and autonomous path replanning.

The synchronized perception, fusion, and navigation architecture significantly improved decision-making performance within rACTor. The robot could dynamically replan traversal paths around unexpected obstacles while maintaining stable autonomous motion control under uneven terrain, shadows, partial occlusions, changing illumination conditions, and dense obstacle scenarios encountered during simulation and outdoor field testing.

6. Cybersecurity measures

Cybersecurity is treated as an essential part of rACTor's safety architecture because the vehicle depends on networked sensors, distributed compute nodes, ROS2 communication, MQTT-based wheel control, and remote development tools. The cybersecurity goal is to protect the confidentiality of configuration and mission data, preserve the integrity of control commands and sensor messages, and maintain the availability of safety-critical functions such as E-Stop, braking, and manual override.

The team follows a defense-in-depth approach inspired by the NIST Risk Management Framework. Security is applied at multiple layers, including the robot's local network, ROS2 middleware, MQTT wheel-control path, developer access, GitHub-based source control, operating-system configuration, physical access, and fail-safe behavior during communication loss or abnormal operation.

6.1 Secure Communication

rACTor uses ROS2 Jazzy as the primary middleware for communication between perception, localization, navigation, and control nodes. ROS2's DDS-based communication model supports authentication, encryption, and access-control policies, which can be enabled to protect inter-

process communication between nodes. These protections reduce the risk of unauthorized message injection, topic spoofing, or eavesdropping on sensor and control traffic.

For the low-level drivetrain, rACTor uses MQTT-based communication between the ROS2 control layer and the WT32-ETH01 microcontroller-based wheel controllers. Since these messages influence steering, throttle, braking, and wheel behavior, the MQTT broker is restricted to the robot's internal network. Only known robot devices are allowed to communicate with the broker, and the system is configured to reject unknown or unnecessary network connections.

6.2 Access Control and Authentication

Access to rACTor's onboard Raspberry Pi, GPU laptop, and supporting development devices is limited to authorized team members. Remote access is controlled through SSH key-based authentication whenever possible, with password login disabled or restricted for critical systems. Administrator privileges are limited to team members responsible for software integration, firmware deployment, and system maintenance.

Role-based access control is applied informally through repository ownership, GitHub permissions, and controlled deployment procedures. Code changes are reviewed before being merged into shared repositories, and critical configuration files are modified only by members responsible for the corresponding subsystem. This reduces the risk of accidental changes to navigation, braking, E-Stop, or firmware behavior.

6.3 Network Protection

The robot operates on a local internal network that connects the Raspberry Pi 5, GPU laptop, Ethernet switch, camera, and microcontroller interfaces. Critical control pathways are kept on wired Ethernet wherever possible to reduce exposure to wireless interference, packet loss, or intentional jamming. Wireless access is used only for operator interaction, testing, and monitoring when required.

Firewall rules are used to limit inbound and outbound traffic to only the ports required for ROS2 communication, MQTT messaging, SSH access, camera streaming, and the browser-based control interface. Unused services and unnecessary open ports are disabled. The robot network is isolated from public or untrusted networks during competition operation, reducing the attack surface and limiting access to internal systems.

6.4 System Hardening

The onboard compute devices run supported operating systems and software stacks, including Ubuntu 24.04, ROS2 Jazzy, and Raspberry Pi OS where applicable. System packages are updated

during development and testing to reduce exposure to known vulnerabilities. Unused services are disabled, default credentials are changed, and only required software packages are installed on competition systems.

Configuration files for navigation, perception, launch files, and hardware interfaces are stored in version-controlled repositories. This allows the team to track changes, revert faulty updates, and restore known-working configurations quickly. System images and backups are maintained for critical devices such as the Raspberry Pi, enabling recovery if a device becomes corrupted or misconfigured.

6.5 Secure Software Development and Code Integrity

All software development is managed through the LTU-Actor GitHub organization. The codebase is organized into modular repositories for navigation, perception, camera publishing, wheel-control bridging, firmware, launch files, and development setup. This separation improves maintainability and reduces the risk that a change in one subsystem unintentionally affects another subsystem.

Before deployment, code is tested in simulation and on the robot where appropriate. Git history provides traceability for changes made to robot behavior. Repository access control, code review, and version control protect against unauthorized or accidental modification of critical software. Development tools and install scripts also help ensure that new developer machines are configured consistently, reducing environment-related failures.

6.6 Data Protection and Integrity

rACTor generates and uses several types of data, including sensor streams, maps, calibration files, logs, navigation parameters, object-detection models, and mission scripts. Sensitive configuration files and backups are stored only on trusted team devices or controlled repositories. Important files are backed up so that they can be restored after device failure or accidental deletion.

Data integrity is protected through version control, file checks, and controlled deployment procedures. Calibration files, launch files, and navigation parameters are not modified during competition unless the change is intentional and tested. Logs are preserved after testing sessions to support debugging, safety review, and performance analysis.

6.7 Secure Remote Access

Remote access is limited to development, testing, and debugging situations. When remote access is required, team members connect only through approved interfaces such as SSH or the robot's local browser-based control panel. Direct exposure of the robot's internal services to the public internet is prohibited.

Remote sessions are monitored during testing so that unexpected behavior can be stopped immediately. If abnormal communication, unstable control behavior, or unauthorized access is suspected, the robot is placed into a safe state and the affected service or network connection is disabled before testing continues.

6.8 Monitoring and Anomaly Detection

rACTor continuously monitors the health of safety-critical and mission-critical subsystems. The watchdog logic checks for communication loss, failed sensor streams, process failures, and abnormal subsystem behavior. If a required process fails or communication with a critical component is lost, the robot is designed to enter a safe state rather than continue autonomous operation.

System logs, ROS2 topics, MQTT messages, and launch output are reviewed during debugging and field testing. Monitoring supports early detection of problems such as missing camera feeds, failed sensor launches, microcontroller communication faults, or navigation stack instability. This approach supports both cybersecurity and operational reliability because many cyber-physical faults first appear as abnormal timing, missing messages, or unexpected actuator behavior.

6.9 Firmware and Update Security

The wheel-control firmware running on the WT32-ETH01 microcontrollers is treated as safety-critical because it affects throttle, steering, braking, and wheel feedback. Firmware updates are performed only by authorized team members using known source code from the team repository. Before deploying firmware changes to the robot, the team verifies that wheel direction, steering angle, Hall-effect feedback, throttle behavior, and braking behavior remain correct.

Rollback procedures are maintained so that the team can return to a known-working firmware version if an update causes unexpected behavior. This prevents unstable firmware from remaining on the robot during competition testing or final operation.

6.10 Physical Security

Physical security is part of cybersecurity because direct access to the robot can allow tampering with USB ports, Ethernet cables, power connections, sensors, microcontrollers, or the E-Stop circuit. Access to the robot is limited to team members and authorized personnel during development and competition. Unused ports are disconnected, disabled, or monitored when not needed.

The Big Red Button, wireless E-Stop, and braking system remain independent safety layers. If software control becomes unstable or communication is lost, the E-Stop system provides an immediate hardware-level method to halt motion. Critical cables and connectors are checked before testing to reduce the chance of accidental disconnection or malicious tampering.

6.11 Cybersecurity Threats and Mitigations

Attack / Failure Scenario | Risk Level | Potential Impact | Mitigation | Safe Response

Local network breach | Medium | Unauthorized access to robot services, ROS2 topics, or web control panel | WPA2-protected local network, firewall rules, MAC/IP restrictions, limited open ports | Disable network access, stop robot, remove unauthorized device

Unauthorized SSH access | Medium | File tampering, process disruption, configuration changes | SSH key authentication, restricted users, disabled default credentials | Terminate session, rotate credentials, restore from backup

ROS2 message injection | Medium | False sensor data or unsafe control messages | DDS security where enabled, isolated network, restricted topic access | Watchdog detects abnormal behavior and triggers safe stop

MQTT command spoofing | Medium | Invalid wheel, steering, or braking commands | Internal broker only, allowed-device restrictions, network isolation | Stop robot and restart verified control stack

Source code tampering | Low to Medium | Unexpected robot behavior after deployment | GitHub version control, code review, protected repositories, backups | Revert to known-good commit and redeploy

Wireless jamming | Low to Medium | Loss of remote access or wireless E-Stop communication | Wired links for critical control, hardware E-Stop, local manual override | E-Stop fails safe and robot halts

Sensor data loss | Medium | Poor perception, localization, or obstacle avoidance | Watchdog checks sensor health and launch status | Stop autonomous motion until sensor returns

Firmware corruption | Medium | Incorrect wheel control or actuator behavior | Controlled firmware updates, repository-based source, rollback procedure | Reflash known-good firmware and retest

Physical tampering | Medium | Cable disconnection, USB misuse, sensor misalignment | Restricted access, cable checks, protected electronics, tamper awareness | Inspect system and validate safety before operation

6.12 Summary

The cybersecurity design of rACTor focuses on practical protection for an autonomous competition robot. The system combines secure communication, access control, network isolation, software version control, firmware management, monitoring, backups, and physical protection. Most importantly, cybersecurity is connected directly to safety: if communication, control, sensing, or software behavior becomes unreliable, rACTor is designed to stop rather than continue operating in an unsafe state.

7. Complete vehicle analysis and comments

7.1. Construction and Integration

Initial testing confirms that the frame design is both robust and sturdy, capable of withstanding rough driving conditions. However, these tests also highlighted the need for a small suspension system when operating off-road or on poorly maintained pavement, as well as more powerful motors to increase the wheels' torque, allowing the car to drive at a steeper angle. The symmetrical chassis design proved advantageous, reducing the number of replacement parts required and simplifying both manufacturing and maintenance. Constructing the robot as two main modules-the motion base and the driver suite-enabled parallel integration timelines and minimized delays due to component sourcing. This approach allowed the team members to work on different parts of the robot simultaneously with no hassle.

7.2. Robot Safety and Reliability

To ensure adequate safety for team members and other pedestrians, rACTor has implemented several safety features. Whenever the robot is active, a safety light flashes, indicating that it is on and possibly mobile. rACTor has a set firmware speed limit of 5 MPH, ensuring that the robot will not cause significant damage in the event of a collision. Steering limits prevent the robot from pulling out cables or wires by ensuring it turns only within its given range of motion. Finally, the robot utilizes a physical E-Stop line that prevents the robot from moving as soon as the E-Stop button is pressed, the wireless E-Stop is activated, or a connection to critical devices is lost. All of these features are meant to keep

the people and the robot safe from harm. Additionally, the camera correctly detects when it is about to hit a pedestrian and stops accordingly, using our fine-tuned YOLO model.

Even though rACTor hardware is built to be robust, we have encountered various recurring failures, as listed in the table below.

Failure Points	Resolution
The camera is not found	Unplug and plug the Ethernet cable back in.
The controller drivers are installed but unused	Flash the Raspberry Pi and update the r/w permissions.
Sensors are not sending a signal	Check the integrity of the file responsible for their launches, update it, and, if needed, copy it from previous backups

Figure 7: Hardware Failures and their Respective Resolutions

7.3. Robot Control

As detailed in Section 4, rACTor has two primary control modes, Double-Ackermann and Fixed Heading. These modes are available and used in both autonomous and teleoperation modes. Nearly all of our autonomous strategies use Double-Ackermann, as Ackermann is a very common control scheme and is natively supported by Nav2. Fixed Heading mode is used in select circumstances where it would provide a significant advantage, such as in parallel parking. Rotation model is also used in punctual instances.

7.4. Software Development and Version Control

All software development done for rACTor is publicly available on our GitHub organization. We use the proven principles of version control in Git/GitHub to maintain software and enable proper collaborative programming.

We also make disk backups of important devices, such as our Raspberry Pi, so that they can be restored in the event of a major failure. Additionally, we create install scripts that set up developer devices for rACTor, enabling new devices and members to be easily onboarded.

7.5. Simulation Environment and Testing

The simulation environment used for robot development is Gazebo, an open-source simulator that enables developers to create and test robots in virtual environments. Used alongside ROS2 Jazzy, Gazebo provides a realistic simulation of a robot's behavior and simulated sensor data. These simulations allow testing software such as Nav2 and SLAM with realistic data across various environments.

Gazebo can use various sensors via a URDF file. The URDF file defines the robot's physical model, including sensor placement. This makes the data received more closely resemble that of a real robot,

enabling more effective simulation and algorithm fine-tuning. The primary goal of testing was to ensure that Nav2 could navigate complex environments and to optimize sensor placement on the actual robot.

8. Performance assessments

Using our course at LTU, we can test functions required for the competition ahead of time. The tables below detail the status of each test.

Self-Drive Tasks

Qualification Tests
Complete- Q.1: E-Stop Manual
Complete- Q.2: E-Stop Wireless
Complete- Q.3: Lane Keeping (Go Straight)
Complete- Q.4: Left Turn
Complete- Q.2: Q.5: Right Turn

Machine Vision Tests
Complete- FI.1: White Line Detection
Complete- FI.2: Static Pedestrian Detection (Vision)
Complete- FI.3: Tire Detection

Traffic Sign Tests
Incomplete- FII.1: Stop Sign Detection
Intersection Tests
Incomplete- FIII.1: Lane Keeping
Incomplete- FIII.2: Left Turn
Incomplete- FIII.3: Right Turn

Parking Tests
Complete- FIV.1: Parking. Pull Out
Complete- FIV.2: Parking. Pull In
Complete- FIV.3: Parking. Parallel

VRU Tests
Incomplete- FV.1: Unobstructed STATIC pedestrian detection
Incomplete- FV.2: Obstructed DYNAMIC pedestrian detection
Incomplete- FV.3: STATIC pedestrian detection. Lane changing
Incomplete- FV.4: Obstacle detection. Lane change

Curve Road Tests
Incomplete- FVI.1: Lane Keeping
Incomplete- FVI.2: Lane Changing

Other Tests
Testing- FVII.1: Pothole Detection
Incomplete- FVII.2: Merging

Main Course
Incomplete- Simple Main

AutoNav Tasks

Qualification Tests
Complete- Robot Dimensions
Complete- E-Stop Manual
Complete- E-Stop Wireless

Course Tests
Testing- Autonomous Control
Complete- Ramps/Grades
Incomplete- Obstacle Routing

AutoNav Tasks

Testing- Safety Light
Complete- Speed Control
Testing- Lane Following
Incomplete- Obstacle Avoidance
Incomplete- Waypoint Navigation
Complete- GPS Localization