UNIVERSITY OF ILLINOIS AT CHICAGO

MECHANICAL AND INDUSTRIAL ENGINEERING

ME 392 - UNDERGRADUATE RESEARCH PROJECT

Autonomous Ackermann Car with Nav2 Stack

Date: 12/10/2025

Submitted by:

Mir Yamin

UIN: 670529549

Abstract

This project focuses on building the hardware foundation of an autonomous Ackermann steering car. The work includes designing and assembling the chassis, developing an appropriate power system, and integrating sensors and key components such as LiDAR, camera, and NVIDIA Jetson Orin Nano. The objective is to complete a fully functional autonomous system from scratch that will support autonomous tasks. This first phase lays the groundwork for developing a machine learning model for lane detection in the next semester.

Methods

1. Design and Development of Parts

The mechanical design of the vehicle was developed by studying the structure and functionality of real passenger vehicles, with particular attention given to steering geometry, drivetrain layout, and axle configuration. Based on this analysis, an Ackermann steering architecture with front steering and rear-wheel drive was selected.

Initial concepts were modeled in SolidWorks, and multiple iterations were evaluated before finalizing the design. Key components designed include the front steering assembly, rear axle assembly, battery holder, Jetson mounting plate, and servo mount. These components were fabricated using a Bambu Lab A1 Mini 3D printer with PETG carbon-fiber filament to achieve a balance between stiffness and weight.

An early design iteration employed a fully 3D-printed differential mechanism; however, durability issues were observed during testing. As a result, the drivetrain design was revised to incorporate a metal differential for improved reliability. Within approximately one month, a fully assembled physical vehicle was completed and prepared for sensor and power-system integration.

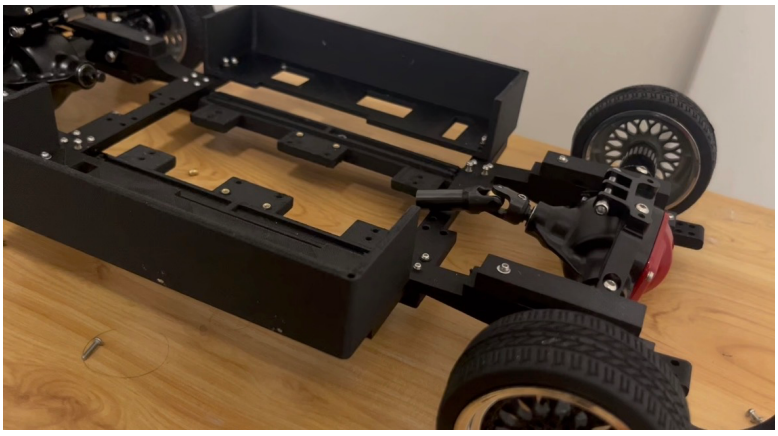

Figure 1: Structure of car

Vehicle Dimensions and Physical Specifications

The autonomous vehicle is a compact Ackermann-steered platform designed for indoor navigation and corridor-scale environments. The overall vehicle dimensions and kinematic parameters are defined consistently across the mechanical design, URDF model, control system, and odometry estimation.

The chassis is modeled as a rigid rectangular body with approximate external dimensions of 0.29 m in length, 0.195 m in width, and 0.09 m in height, matching the fabricated platform. The wheelbase, defined as the distance between the front and rear axles, is 0.29 m, while the track width, measured between the centers of the left and right wheels, is 0.195 m.

Each wheel has a radius of 0.033 m, corresponding to the physical tire dimensions used on the vehicle. These parameters are used consistently in the URDF model, motor-based odometry calculations, and Ackermann steering geometry to ensure accurate kinematic behavior and state estimation.

Sensors are mounted within the vehicle envelope to maintain a compact footprint. The LiDAR is mounted approximately 0.085 m above the base_link frame, centered laterally on the chassis, while the RGB-D camera is mounted forward of the chassis centerline with an optical frame aligned to ROS conventions. The compact size of the platform enables maneuverability in narrow indoor spaces while still supporting realistic Ackermann steering behavior.

A summary of key physical dimensions is provided below:

Parameter	Value
Overall length	0.29 m
Overall width	0.195 m
Overall height	0.09 m
Wheelbase	0.29 m
Track width	0.195 m
Wheel radius	0.033 m
LiDAR mounting height	0.085 m

Table 1: Physical Parameters

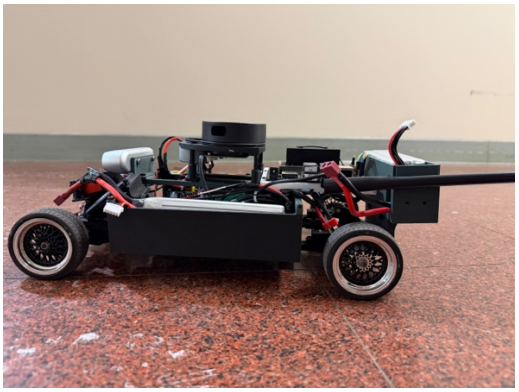

Figure 2: Assembled Car

2. Sensor and Power Integration

Car Movement is provided by a 2200 KV sensored brushless DC motor controlled through a VESC motor controller. The VESC is interfaced with the NVIDIA Jetson Orin Nano via UART communication, enabling both command transmission and telemetry feedback. The vehicle power system uses two separate 4S 4400 mAh lithium-polymer batteries: one dedicated to powering the Jetson and onboard electronics, and the other dedicated to the motor and drivetrain. To protect the Jetson from overcurrent conditions, a 5 A inline fuse is installed on the computing power line.

For perception, the vehicle integrates two primary sensors: a 2D LiDAR with a maximum range of 12 m and an Intel RealSense D435i RGB-D camera. Steering is actuated using a high-torque servo motor rated at 25 kg-cm. The servo is driven through a PCA9685 multi-channel servo controller, which communicates with the Jetson via the I2C protocol. Both the LiDAR and the camera are connected to the Jetson through USB interfaces.

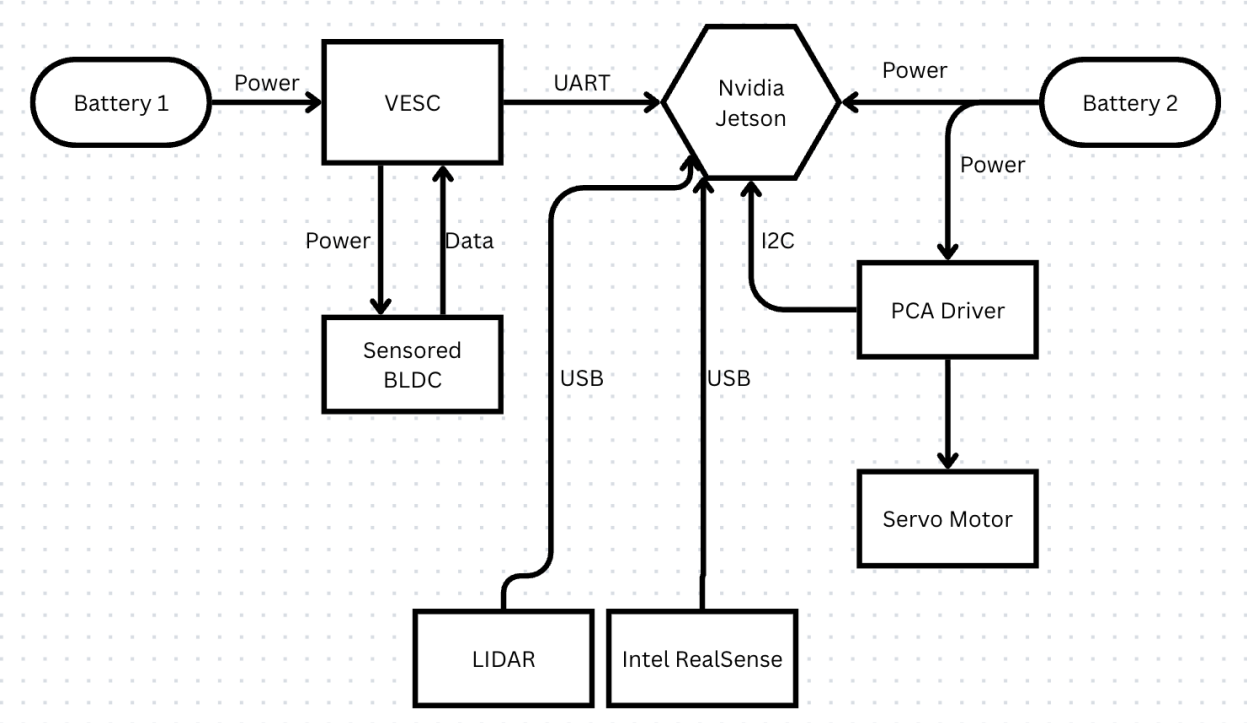

Figure 3: System Flowchart

3. Control System

To integrate all the sensors, ROS 2 Humble was used. Several nodes were created. The VESC node streams motor data using an imported Python library for VESC. The VESC can stream:

1. ERPM
2. Duty Cycle
3. Current
4. Voltage
5. Amp Hours Drawn
6. Fault Code

The autonomous Ackermann car utilizes a modular ROS 2-based software architecture designed to support both real-world hardware operation and future simulation-based testing. The software stack integrates low-level motor and steering control, state estimation through odometry, robot description and visualization, and a web-based teleoperation interface. The system is structured to be compatible with higher-level autonomous frameworks such as SLAM and the Nav2 navigation stack.

The overall design philosophy emphasizes:

1. Clear separation between hardware control and high-level command generation
2. Consistent coordinate frame definitions across URDF, odometry, and navigation
3. Safety and robustness through command timeouts and avoidance of hardware resource conflicts

Robot Description and Kinematic Modeling

A complete kinematic and geometric description of the vehicle is defined using a Xacro-based URDF model (`robot_core.xacro`). This file serves as the authoritative source for all physical dimensions, coordinate frames, joint definitions, and inertial properties used throughout the system.

The robot reference frame is defined at `base_link`, with a fixed `base_footprint` frame used for planar navigation. The chassis is modeled as a rigid rectangular body with dimensions matching the fabricated platform. Mass and inertia for the chassis, wheels, steering knuckles, and sensor mounts are defined using reusable inertial macros to ensure physically reasonable values for visualization and simulation.

An Ackermann steering configuration is implemented using:

- A fixed rear axle with continuous wheel joints for rear wheel rotation
- A fixed front axle supporting two revolute steering joints
- Continuous wheel joints downstream of the steering joints

Steering limits are explicitly defined in the URDF to reflect mechanical constraints. The wheelbase, track width, and wheel radius used in the URDF are aligned with values used in control and odometry nodes to ensure consistency across the system.

Sensor mounting is explicitly represented in the URDF. Camera links are rigidly attached to the chassis with a corresponding optical frame to follow ROS camera conventions. A LiDAR frame is mounted above the chassis centerline to match the physical sensor placement. These frame definitions enable correct transformation of sensor data into the robot reference frame for mapping and navigation.

Simulation-specific Gazebo plugins for Ackermann steering, joint state publishing, and simulated sensors are included but commented out. This preserves simulation capability while ensuring the URDF remains appropriate for real-robot deployment.

Visualization and Frame Broadcasting

The robot model and coordinate frames are brought up using a dedicated ROS 2 launch file (`display.launch.py`). This launch configuration executes the robot state publisher, which dynamically generates the URDF from the Xacro file at launch time and publishes the complete TF tree based on incoming joint state data.

A static transform between `base_link` and the LiDAR frame is published to define the rigid mounting of the sensor. This transform is required for correct projection of LaserScan data into the robot frame and is essential for SLAM and navigation pipelines.

An optional joint state publisher can be enabled via a launch argument to allow RViz visualization of wheel rotation and steering angles in the absence of live hardware feedback. This feature is particularly useful during debugging and early integration testing.

RViz2 is launched with system time enabled, reflecting the real-robot configuration where sensor timestamps are derived from the onboard computer rather than a simulated clock.

Velocity Command to Hardware Control Bridge

Low-level actuation is handled by a dedicated ROS 2 node that converts velocity commands into physical motor and steering commands. This node subscribes to `geometry_msgs/Twist` messages on `/cmd_vel`, which are typically produced by Nav2 or teleoperation interfaces.

Linear velocity commands are converted into motor duty cycle commands for a VESC-controlled brushless DC motor. A calibrated scaling factor, offset, and saturation limits are applied to ensure smooth motion and to protect hardware from excessive current draw. Braking behavior is implemented using controlled current braking when the commanded velocity falls below a defined threshold.

Angular velocity commands are converted into steering angles using Ackermann geometry. The steering angle is computed based on the vehicle wheelbase and clamped to a maximum allowable angle consistent with the mechanical steering limits. A dead zone is applied near the center position to prevent small numerical fluctuations from causing unintended steering oscillations.

Physical steering is executed through a high-torque servo motor driven by a PCA9685 controller. The steering angle computed in software is mapped to servo angles using experimentally measured limits and center positions. Servo commands are refreshed periodically to prevent drift.

To ensure safety, a heartbeat mechanism continuously re-sends the most recent motor and steering commands while valid velocity commands are present. If command messages stop arriving for longer than a specified timeout, the system automatically stops the motor and re-centers the steering.

The node also polls telemetry data from the VESC, including RPM and tachometer counts, and publishes this information for downstream odometry estimation.

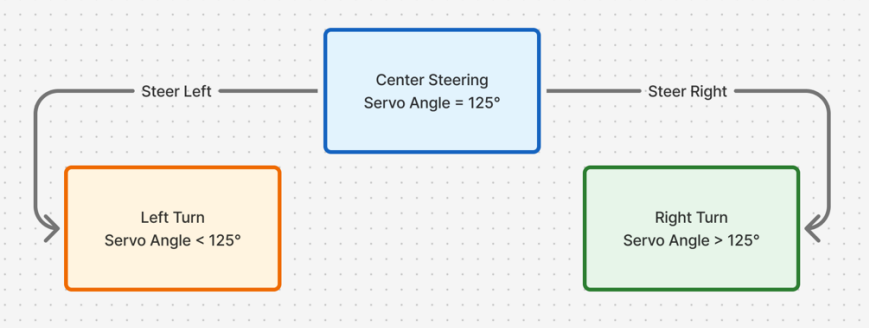

Figure 4: Steering Angles

Motor-Based Odometry and State Estimation

Vehicle odometry is computed using a dedicated motor odometry node that estimates planar position and orientation based on wheel encoder data and steering feedback. This node subscribes to tachometer counts from the VESC and steering angle information from the control layer.

Distance traveled is computed using a directly measured calibration factor relating encoder counts to linear displacement. This empirical calibration improves accuracy compared to relying solely on nominal gear ratios or motor specifications.

The vehicle pose is updated using Ackermann kinematic equations. For straight-line motion, position updates are computed directly along the current heading. For turning motion, the change in orientation is computed from the steering angle and wheelbase, and position updates are calculated using the average heading over the integration interval. The orientation angle is normalized to prevent numerical drift.

Odometry messages are published on the `odom` topic, providing both pose and twist estimates. Velocities are computed from the same distance and heading increments used for pose updates to maintain consistency between pose and velocity estimates.

The node also broadcasts the transform from `odom` to `base_link`, forming the foundational link in the TF tree required by SLAM and navigation frameworks. Additionally, joint state messages

are published to allow RViz visualization of wheel rotation and steering articulation using real encoder and steering data.

Web-Based Teleoperation and Monitoring Interface

A lightweight web-based control interface is implemented using a Flask application that integrates directly with ROS 2. This interface provides a browser-accessible control panel for commanding motion, steering, and emergency stops, and for monitoring basic system state.

Within the Flask application, a ROS 2 node runs in a background thread to allow continuous communication with the ROS graph while servicing HTTP requests. Movement commands issued through the web interface are translated into `cmd_vel` messages rather than direct motor commands, ensuring compatibility with SLAM and Nav2 and avoiding conflicts with other control nodes.

Physical steering commands are sent directly to the servo controller to allow immediate manual steering, while an equivalent steering angle is always published to ROS for odometry and visualization. This dual approach ensures that state estimation remains consistent regardless of whether steering commands originate from autonomous planners or manual inputs.

Direct serial access to the VESC is intentionally disabled within the web interface. This design choice prevents multiple processes from attempting to control the same hardware interface and centralizes all motor control and safety logic within the dedicated motor control node.

An emergency stop endpoint immediately publishes zero velocity commands and re-centers the steering, providing a simple but effective safety mechanism during testing.

Integration with SLAM and Navigation

Mapping of the indoor environment was performed using **SLAM Toolbox**, which generates a two-dimensional occupancy grid map using LiDAR data and motor-based odometry. During the mapping phase, SLAM Toolbox continuously estimates the vehicle pose while building the global map. Once mapping was complete, the generated map was saved and used for subsequent autonomous navigation.

For autonomous operation, SLAM Toolbox was disabled and localization was performed using **Adaptive Monte Carlo Localization (AMCL)**. AMCL estimates the vehicle pose within the pre-built map using LiDAR scan matching and a particle filter, providing a stable and computationally efficient localization solution for real-world deployment.

The Nav2 navigation stack consumes localization data from AMCL, odometry data from the motor-based odometry node, and obstacle information from the LiDAR to perform path planning and control. Velocity commands generated by Nav2 are published on the `/cmd_vel` topic and translated into physical motor and steering commands by the low-level control bridge.

Consistent coordinate frame definitions across the URDF, odometry node, and static transforms ensure proper alignment between the `map`, `odom`, and `base_link` frames. This consistency is essential for reliable mapping, localization, and navigation.

4. Navigation and Path Planning Using Nav2

Autonomous navigation for the Ackermann vehicle is implemented using the ROS 2 Navigation Stack (Nav2). Nav2 provides a modular framework for global path planning, local trajectory tracking, obstacle avoidance, and recovery behaviors. The navigation system is integrated with the existing SLAM, odometry, and control architecture through standardized ROS interfaces, allowing seamless operation on the real vehicle.

Global Planning with SMAC Hybrid Planner

Global path planning is performed using the SMAC Hybrid Planner, which is specifically designed for nonholonomic vehicles such as Ackermann-steered cars. Unlike grid-based planners that assume holonomic motion, the hybrid planner accounts for the vehicle's minimum turning radius and steering constraints.

The SMAC Hybrid Planner generates kinematically feasible paths by searching in a combined state space of position and orientation. This ensures that planned trajectories respect the vehicle's steering limits and do not require infeasible in-place rotations. The planner produces smooth paths that can be directly followed by the local controller without excessive correction.

To further enhance navigation capability, the planner is configured to use Reeds-Shepp motion primitives, enabling the vehicle to plan paths that include reverse motion. This is particularly important in constrained environments such as narrow corridors or dead ends, where forward-only motion may not allow recovery or reorientation. With Reeds-Shepp enabled, the vehicle can autonomously reverse and re-plan, significantly improving robustness and practical usability.

Local Control with Regulated Pure Pursuit

Local trajectory tracking is handled using the Regulated Pure Pursuit Controller, which is well-suited for Ackermann steering vehicles. This controller computes steering and velocity commands based on a look-ahead point along the global path and adjusts its behavior dynamically based on curvature, obstacle proximity, and velocity constraints.

The regulated version of Pure Pursuit introduces additional safeguards compared to the classical algorithm. Velocity is reduced automatically in regions of high curvature or near obstacles, improving stability and reducing the risk of overshoot. Steering commands are smoothed to prevent aggressive oscillations, which is critical for maintaining reliable control on a real vehicle with mechanical steering limitations.

The controller outputs velocity commands in the form of linear and angular velocities on `/cmd_vel`. These commands are subsequently processed by the low-level motor control node, which converts them into motor duty cycles and steering angles using Ackermann geometry.

Reversing Behavior and Nonholonomic Constraints

The inclusion of Reeds-Shepp motion models allows the navigation system to explicitly reason about both forward and reverse motion. This capability is essential for realistic autonomous driving scenarios where turning radius constraints prevent forward-only maneuvers.

When reversing is required, the global planner produces paths that include backward segments. These segments are followed by the Regulated Pure Pursuit controller, which adjusts velocity sign and steering direction accordingly. The low-level motor control bridge supports both forward and reverse motion through bidirectional motor commands, ensuring compatibility with the planner output.

This configuration enables behaviors such as:

- Backing out of dead ends
- Repositioning after localization drift
- Navigating tight indoor environments

Integration with Odometry and TF

Accurate navigation requires consistent frame alignment across the system. The Nav2 stack relies on the `map`, `odom`, and `base_link` frames to compute localization and control commands. These frames are maintained through a combination of SLAM-generated localization, motor-based odometry, and TF broadcasting.

The odometry node publishes the `odom → base_link` transform using encoder and steering data, while the SLAM system provides the `map → odom` transform. The URDF and static transforms ensure that sensor frames such as the LiDAR are correctly positioned relative to the robot body. This unified TF tree allows Nav2 to correctly associate sensor observations, robot pose, and planned paths.

Summary of Navigation Configuration

The navigation subsystem is configured with the following key components:

- **Global Planner:** SMAC Hybrid Planner
- **Motion Model:** Reeds-Shepp (forward and reverse motion enabled)
- **Local Controller:** Regulated Pure Pursuit
- **Control Output:** `/cmd_vel` velocity commands
- **Vehicle Type:** Nonholonomic Ackermann steering

This configuration provides a robust and kinematically consistent navigation solution suitable for real-world indoor environments.

Local Costmap Configuration

The local costmap represents nearby obstacles and is used by the controller for short-horizon collision avoidance. A rolling window is used to reduce computational cost while maintaining responsiveness to dynamic obstacles.

The vehicle footprint is explicitly defined to reflect the physical dimensions of the car.

Parameter	Value	Description
global_frame	odom	Local reference frame
rolling_window	true	Costmap moves with robot
width × height	3 × 3 m	Local planning area
resolution	0.05 m	Grid resolution
footprint	0.32 × 0.20 m	Approximated vehicle footprint
plugins	voxel_layer, inflation_layer	Obstacle processing
inflation_radius	0.1 m	Conservative local inflation

Table 2: Configuration

Global Costmap Configuration

The global costmap represents the environment map used for long-range planning. It integrates the static SLAM map, live obstacle data, and inflation to ensure safe path generation.

Parameter	Value	Description
global_frame	map	Global planning frame
track_unknown_space	true	Allows exploration
resolution	0.05 m	Map resolution
plugins	static, obstacle, inflation	Costmap layers
inflation_radius	0.2 m	Safer global paths
observation_source	/scan	LiDAR-based obstacles

Table 3: Configuration

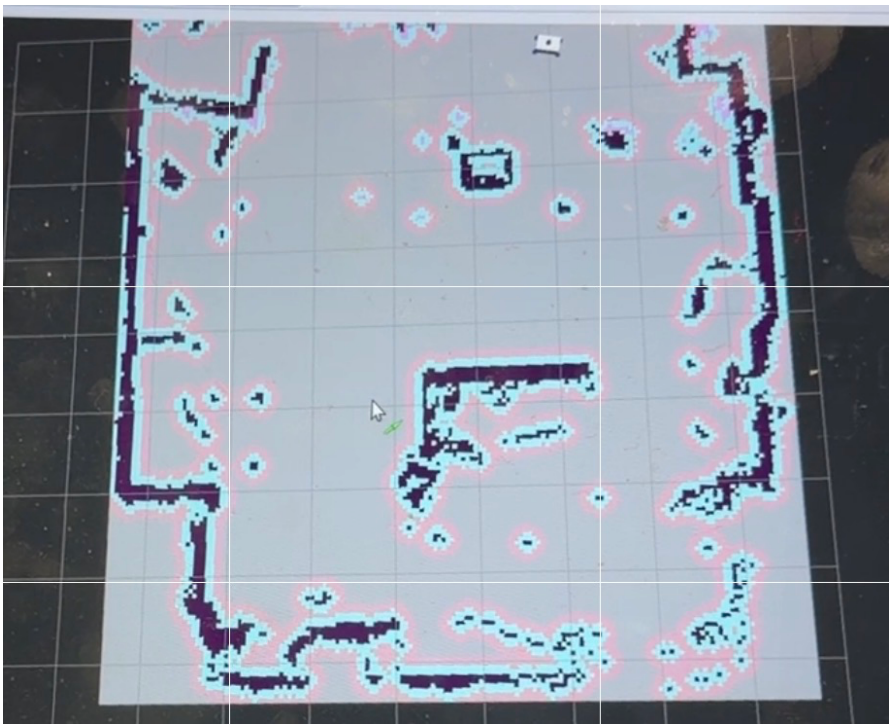

Figure 5: global Cost map

Recovery Behaviors

Recovery behaviors are enabled to improve robustness when navigation fails. These behaviors allow the robot to back up, rotate, or pause when progress is blocked.

Behavior	Purpose
spin	Reorient vehicle
backup	Reverse away from obstacles
drive_on_heading	Move straight for recovery
assisted_teleop	Manual intervention
wait	Pause execution

Table 4: Recovery Behavior

Velocity Smoothing

A velocity smoother is used to limit acceleration, deceleration, and angular rates before commands are sent to hardware. This prevents abrupt changes in motor duty cycle and steering angle.

Parameter	Value	Description
smoothing_frequency	20 Hz	Output rate
max_velocity	±0.3 m/s	Linear limits
max_angular_velocity	±1.0 rad/s	Steering limits
max_accel	2.5 m/s²	Acceleration limit
max_decel	−2.5 m/s²	Braking limit

Table 5: Smoothing details

Summary of Navigation Stack

Component	Selection
Localization	AMCL
Global Planner	SMAC Hybrid Planner
Motion Model	Reeds-Shepp
Local Controller	Regulated Pure Pursuit
Costmaps	Voxel + Inflation
Reverse Motion	Enabled
Vehicle Type	Ackermann steering

Table 6: Summary of Nav2

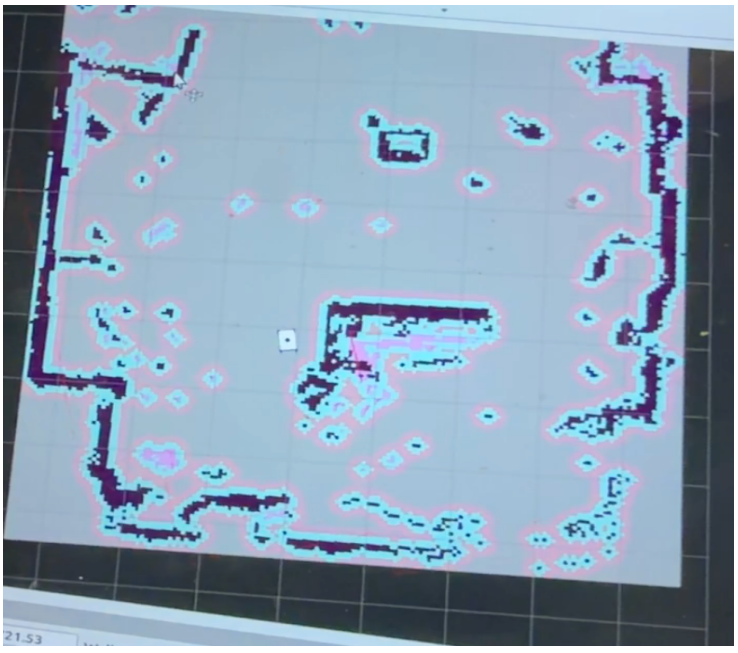

Figure 6: Car localized with AMCL

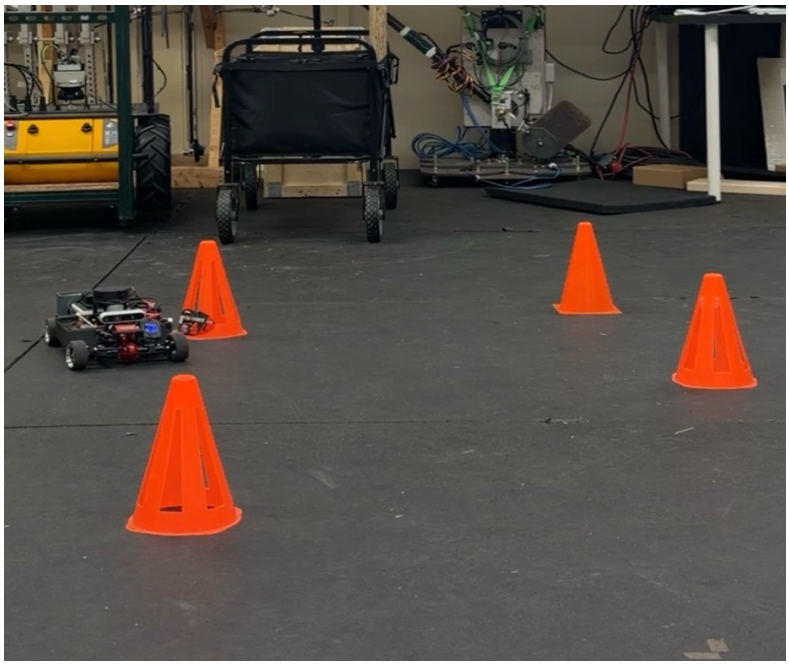

Figure 7: Car navigating autonomously in Lab

Figure 8: Car navigating around a cardboard box

Results

The developed autonomous Ackermann vehicle was evaluated through a series of hardware integration tests, mapping experiments, localization trials, and autonomous navigation runs conducted in an indoor laboratory environment. The results demonstrate that the system met its primary objectives of achieving reliable motion control, mapping, localization, and autonomous navigation using the ROS 2 Nav2 framework.

1. Mechanical and Hardware Performance

The fabricated chassis and steering mechanism operated reliably under repeated testing. The Ackermann steering geometry produced smooth turning behavior without noticeable mechanical binding or wheel slip at low to moderate speeds. The metal differential replacement significantly improved drivetrain durability compared to earlier 3D-printed iterations, eliminating gear wear and torque transmission issues observed in preliminary tests.

The power system successfully isolated computation and actuation loads using dual batteries. No voltage drop–related system resets were observed on the NVIDIA Jetson Orin Nano during extended operation, confirming the effectiveness of the separate power rails and inline fuse protection.

2. Sensor Integration and Data Availability

Both primary perception sensors—the 2D LiDAR and Intel RealSense D435i camera—operated reliably throughout testing. The LiDAR consistently produced clean and dense scan data suitable for SLAM and navigation, while the RGB-D camera provided synchronized color and depth streams for visualization and future perception development.

All sensor data were correctly transformed into the robot reference frame using the URDF-defined frames and static transforms. Visualization in RViz confirmed accurate alignment between the LiDAR scans, robot footprint, and environment geometry.

3. Motor Control and Steering Response

Velocity commands published on the `/cmd_vel` topic were accurately converted into motor duty cycle and steering angle commands by the low-level control node. The servo steering mechanism demonstrated repeatable behavior, with the calibrated center position at approximately 125° and symmetric steering response to left and right commands.

The applied dead zone around the steering center successfully eliminated oscillatory behavior caused by small command fluctuations. Velocity smoothing further reduced abrupt acceleration and deceleration, resulting in stable motion suitable for autonomous navigation.

4. Odometry Accuracy and State Estimation

Motor-based odometry provided continuous pose and velocity estimates during vehicle motion. Straight-line motion tests showed consistent forward displacement without noticeable lateral drift. During turning maneuvers, the Ackermann kinematic model produced smooth orientation updates, with heading changes matching the expected curvature based on steering angle.

The odometry transform (`odom → base_link`) remained stable over time and provided a reliable input for both SLAM Toolbox and Nav2. While minor accumulated drift was observed during long trajectories, this behavior is expected for encoder-only odometry and was corrected during localization using AMCL.

5. Mapping with SLAM Toolbox

The indoor laboratory environment was successfully mapped using SLAM Toolbox operating in synchronous mode. The generated occupancy grid accurately captured walls, corridors, and static obstacles. Loop closures were detected reliably, resulting in consistent map geometry without major distortions.

Once mapping was completed, the final map was saved and reused for autonomous navigation experiments. Visual inspection in RViz confirmed good alignment between the map, LiDAR scans, and robot pose.

6. Localization Using AMCL

During autonomous operation, Adaptive Monte Carlo Localization (AMCL) provided stable pose estimates within the prebuilt map. The particle cloud converged quickly after initialization, and pose estimates remained consistent during navigation.

Localization performance remained robust even during moderate steering maneuvers and velocity changes. Recovery behaviors such as backing up and reorientation were successfully triggered when localization confidence decreased, allowing the vehicle to regain a valid pose estimate.

7. Autonomous Navigation Performance

Autonomous navigation was successfully demonstrated using the Nav2 stack. The SMAC Hybrid Planner generated kinematically feasible global paths that respected the vehicle's minimum turning radius and steering constraints. Paths were smooth and continuous, eliminating the need for in-place rotations.

Reeds–Shepp motion primitives enabled reverse motion when required, particularly in narrow or constrained areas. The vehicle successfully backed out of dead ends and repositioned itself to reach navigation goals.

Local trajectory tracking using the Regulated Pure Pursuit controller resulted in stable path following with minimal oscillation. The controller automatically reduced velocity in high-curvature regions and near obstacles, improving safety and tracking accuracy.

8. Costmap and Obstacle Handling

The local and global costmaps accurately reflected static obstacles and environmental boundaries. Inflation layers provided sufficient clearance around obstacles, preventing collisions during navigation. The rolling local costmap updated responsively as the vehicle moved, enabling short-horizon obstacle avoidance.

Recovery behaviors—including spin, backup, and drive-on-heading—were triggered appropriately when navigation stalled, improving robustness during testing.

Conclusion:

This project successfully demonstrated the complete design, fabrication, and integration of a real-world autonomous Ackermann-steered vehicle using a modern ROS 2–based software stack. Starting from a blank platform, the work encompassed mechanical design, power system development, sensor integration, low-level motor and steering control, state estimation, mapping, localization, and autonomous navigation. The resulting system provides a robust and extensible foundation for further research in autonomous ground vehicles.

Mechanically, a compact yet structurally sound chassis was designed and fabricated using iterative CAD development and additive manufacturing. The final configuration achieved realistic Ackermann steering behavior while maintaining a small footprint suitable for indoor laboratory environments. Key kinematic parameters were defined consistently across the physical platform, URDF model, and control software, enabling reliable motion and state estimation.

From a software perspective, a modular ROS 2 Humble architecture was implemented to ensure clear separation between hardware control, perception, localization, and navigation. Low-level velocity and steering commands were safely translated into physical motor and servo actuation through a dedicated control bridge, while motor-based odometry provided continuous pose estimation. A complete TF tree and robot description enabled accurate visualization and sensor data alignment.

Mapping and localization were achieved using SLAM Toolbox and AMCL, producing a usable two-dimensional occupancy map of the environment and enabling reliable pose estimation during autonomous operation. The Nav2 navigation stack was successfully integrated using a configuration tailored for nonholonomic Ackermann vehicles. The SMAC Hybrid Planner, combined with Reeds–Shepp motion models, allowed the vehicle to generate kinematically feasible paths that included reverse motion when necessary. Local trajectory tracking was handled using the Regulated Pure Pursuit controller, providing smooth and stable motion even in constrained indoor spaces.

Experimental testing demonstrated that the vehicle could autonomously localize itself within a prebuilt map and navigate to user-defined goals while respecting steering constraints and avoiding obstacles. Recovery behaviors and velocity smoothing further enhanced robustness and safety during operation.

Overall, this project establishes a complete and functional autonomous vehicle platform suitable for continued research and development. Future work will build upon this foundation by incorporating machine learning–based lane detection using the onboard RGB-D camera, sensor fusion for improved localization accuracy, and expanded testing in more complex environments. The system architecture developed in this work is intentionally scalable, allowing additional perception, planning, and control modules to be integrated with minimal restructuring.

References

1. ROS 2 Humble Documentation. https://docs.ros.org/en/humble/index.html

2. Nav2 Plugins Documentation. https://docs.nav2.org/plugins/

3. Youyeetoo Robotics Platform. https://www.youyeetoo.com/