Ben-Gurion University of the Negev

Faculty of Engineering Science

School of Electrical and Computer Engineering

Dept. of Electrical and Computer Engineering

Fourth Year Engineering Project

Final Report

System architecture in formula student car

Project number:	**p-2024-057**
Students:	**Michael Leib - 319095832**
Supervisors:	**Dr. Igal Bilik**
Submitting date:	**18.08.24**

TABLE OF CONTENTS

System Architecture for an Autonomous Racing Car: Design and Implementation for Formula Student Driverless

Student Name: Leib Michael

leibmi@post.bgu.ac.il

Adviser name: Bilik Igal

Formula Student competitions now include autonomous racing, challenging teams to develop high-performance driverless vehicles capable of navigating complex tracks.

This project aims to develop a system architecture for an autonomous Formula Student car capable of competing in dynamic racing events.

The objectives include designing a software architecture that integrates essential components such as perception, localization, path planning, and control, exploring advanced algorithms suitable for autonomous racing, and incorporating existing research to strengthen the proposed architecture.

The innovation in this project lies in the initial proposal of an autonomous system within the university's BGRacing team, marking a first-of-its-kind effort in the group's history.

The system utilizes a ROS2 framework that integrates YOLOv8 for cone detection, LiDAR clustering for environmental mapping, Kalman filtering for localization, Delaunay triangulation for path planning, and Pure Pursuit with PID control for vehicle navigation. It combines data from cameras, LiDAR, and IMU sensors to deliver a comprehensive autonomous driving solution.

The expected outcome is to establish a foundation for future developments while improving vehicle performance, safety, and reliability in autonomous racing.

Keywords: Autonomous racing, Formula Student, ROS2, perception, localization, path planning, control systems.

ארכיטקטורת מערכת לרכב מרוץ אוטונומי: תכנון ויישום עבור פורמולה סטודנטים אוטונומי

שם הסטודנט: לייב מיכאל

leibmi@post.bgu.ac.il

שם המנחה : ד"ר ביליק יגאל

תחרויות פורמולה סטודנטים כוללות כיום גם מרוצים אוטונומיים, המאתגרים קבוצות לפתח רכבים אוטונומיים בעלי ביצועים גבוהים המיועדים לניווט במסלולים מורכבים.

מטרת הפרויקט היא לפתח ארכיטקטורת מערכת לרכב פורמולה סטודנטים אוטונומי, המסוגל להתחרות במקצים הדינמיים של התחרות.

מטרות הפרויקט כוללות פיתוח ארכיטקטורת תוכנה המשלבת רכיבים חיוניים כמו תפיסה, לוקליזציה, תכנון מסלול ובקרה, חקר אלגוריתמים מתקדמים המתאימים למרוצים אוטונומיים, ושילוב מחקר קיים לחיזוק הארכיטקטורה המוצעת.

החידוש בפרויקט זה טמון בהצעה הראשונית למערכת אוטונומית במסגרת קבוצת המרוצים של האוניברסיטה, ומהווה ניסיון ראשון מסוגו בהיסטוריית הקבוצה.

המערכת עושה שימוש במסגרת ROS2 המשלבת YOLOv8 לזיהוי קונוסים, אלגוריתמי למידת מכונה עבור LiDAR למיפוי סביבתי, מסנן קלמן ללוקליזציה, אלגוריתמי תכנון מסלול, ושליטה במערכת ההיגוי באמצעות Pure Pursuit ובקרת PID . המערכת משלבת נתונים ממצלמות LiDAR ,וחיישני IMU כדי לספק פתרון נהיגה אוטונומי מקיף.

התוצאה המצופה היא יצירת תשתית לפיתוחים עתידיים, שיפור ביצועי הרכב, בטיחות ואמינות במרוצים אוטונומיים.

Project Objectives

The primary objective of this project is to propose a practical software architecture for the autonomous system of a racing car. This architecture will serve as the guiding framework for developing the vehicle's autonomous capabilities, enabling it to operate effectively in dynamic and high-speed environments typical of the Formula Student competition.

1. **System Architecture Proposal**: The project aims to develop a software architecture that integrates essential components such as perception, localization, path planning, and control. This architecture will be designed to handle real-time data processing, ensuring that the vehicle can autonomously respond to its environment and make informed decisions.

2. **Exploration of Algorithms**: Additionally, the project involves exploring various algorithms that can be applied to different aspects of the autonomous system, including sensor data processing, localization, path planning, and control. The goal is to evaluate and select algorithms that are suitable for implementation within the proposed architecture, ensuring they meet the performance requirements of autonomous racing.

3. **Integration of Existing Research**: The project also seeks to incorporate existing research and methodologies from other projects to strengthen the proposed architecture. This involves adapting proven techniques and integrating them into the system to meet the specific needs of the racing car's autonomous system.

4. **Foundation for Future Work**: Finally, the project aims to establish a software framework that can be further expanded and refined in future projects. This framework will allow for the integration of new technologies as they become available, providing a solid foundation for ongoing development.

The success of this project will be measured by the effectiveness of the proposed software architecture in supporting the autonomous operation of the vehicle, with a focus on creating a robust and adaptable foundation for future work.

Introduction

The automotive industry is rapidly advancing towards a future where human involvement in driving is minimized, driven by the development of autonomous vehicles. These vehicles promise enhanced safety, reduced road fatalities, and increased accessibility. Achieving SAE Level 4 autonomy requires sophisticated systems capable of operating independently under complex conditions, addressing the critical need to reduce human error - a major cause of traffic accidents.

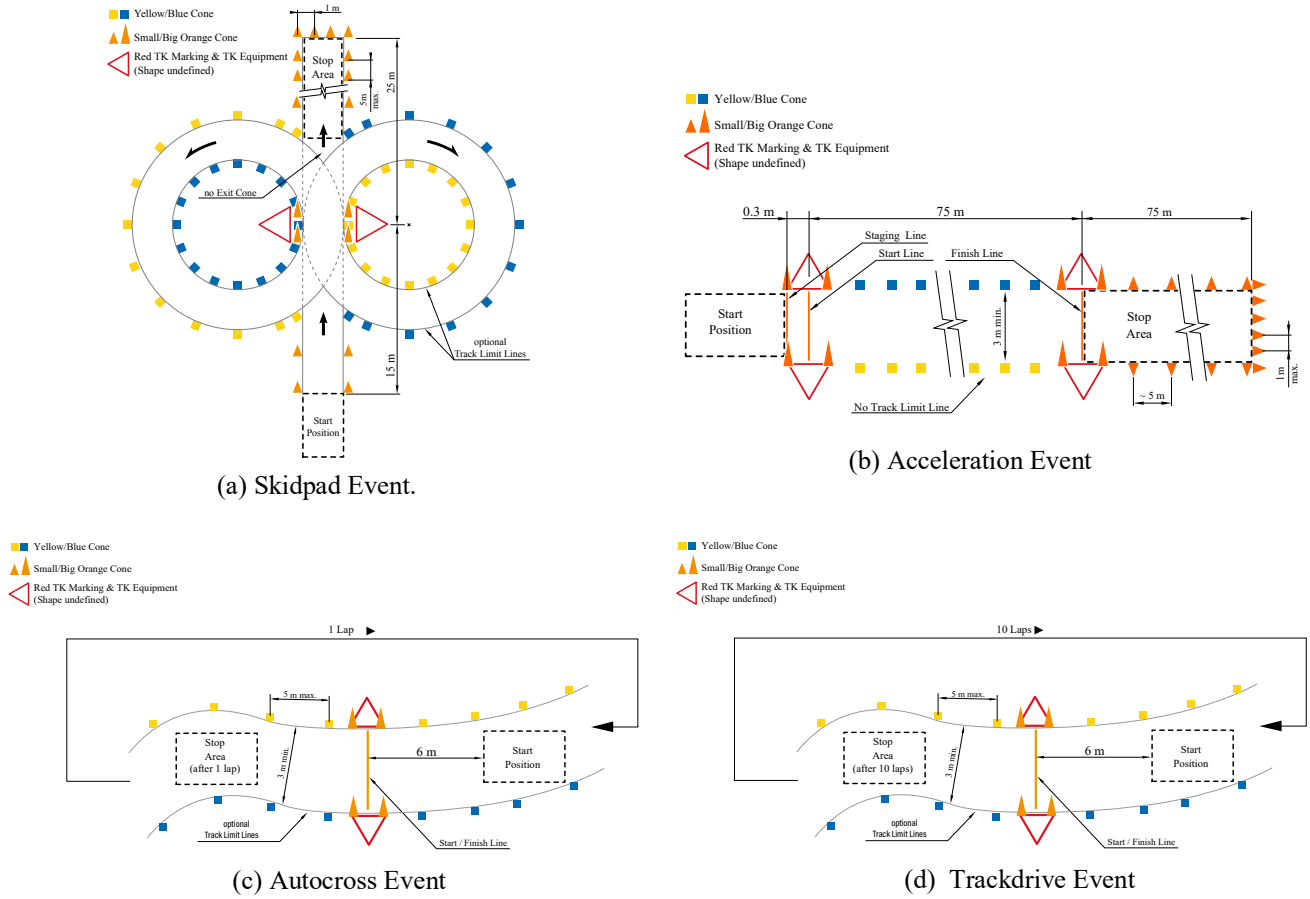
Fig. 1. BGR's electric race car during the Formula Student Germany (FSG) competition.

In line with these trends, the Formula Student competition introduced the Formula Student Driverless (FSD) class in 2017. This competition challenges student teams worldwide to design, build, and race high-performance autonomous cars. These vehicles must navigate various events without any human intervention, relying solely on advanced perception, planning, and control systems. The FSD class serves as a key platform for testing and validating new technologies under demanding conditions, pushing the boundaries of what autonomous systems can achieve.

Autonomous racing poses unique challenges distinct from traditional autonomous vehicle development. The dynamic events in the Formula Student Driverless competition [1], Skid Pad, Acceleration, Autocross, and Trackdrive, require the vehicle to handle high-speed maneuvers, navigate tight corners, and maintain stability under various conditions. These events include:

- **Skid Pad**: The car must complete two full laps as quickly as possible on a track shaped like an eight, which tests the car's lateral acceleration and handling capabilities (see Fig. 2.a).
- **Acceleration**: The car is required to accelerate as fast as possible over a straight 75-meter track, focusing on the vehicle's longitudinal acceleration (see Fig. 2.b).
- **Autocross**: The car must navigate one full lap on an unknown closed-loop track, ranging from 200 to 500 meters in length. This event tests the vehicle's ability to handle various corners and straights, simulating real-world driving challenges (see Fig. 2.c).

- **Trackdrive and Efficiency**: The car must complete ten consecutive laps on the same track layout guideline used in the autocross event. This event tests the consistency and robustness of the vehicle's systems as well evaluates its energy efficiency during the laps (see Fig. 2.d).



(a) Skidpad Event.

(b) Acceleration Event

(c) Autocross Event

(d) Trackdrive Event

Fig. 2. Formula Student Dynamic Events for the Driverless Class. Figures taken from [1].

The tracks used in these dynamic events are marked by cones, with blue cones indicating the left boundary and yellow cones marking the right. Small orange cones are used to define stopping zones, while larger orange cones mark timekeeping zones, as shown in Fig. 3. Accurate perception of these cones and precise navigation around them are crucial for success in these events.

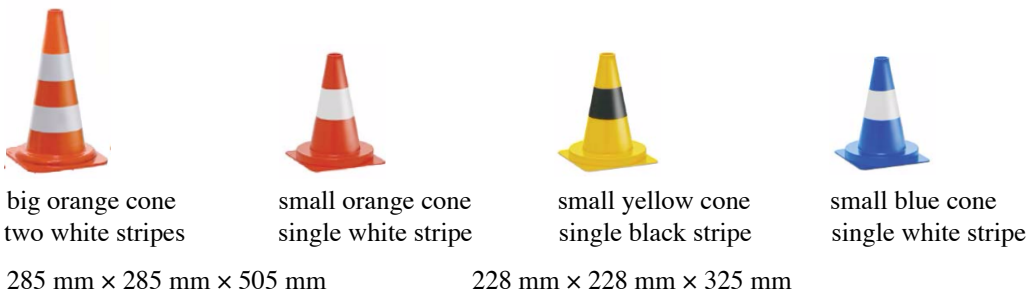


Fig. 3. Specifications of the cones used in Formula Student Driverless competitions.

Different teams have tackled the challenges of autonomous racing with various strategies. KA-RaceIng began with a modular design and adaptive control [2], later enhancing it with robust localization and path planning [3]. Elefant Racing focused on a simple, efficient architecture for key events [4], while AMZ Driverless integrated advanced sensor fusion and decision-making to excel across all events [5]. These insights guided the development of our own system, tailored to the specific challenges of the competition.

Our team, BGRacing, has a history of developing electric race cars for competitions. After successfully completing several competitions (Fig. 1) and overcoming the challenge of transitioning to electric vehicles, we decided to take on the next challenge: developing an autonomous vehicle to compete in the autonomous events. This project is the proposal for the autonomous system that will operate on the next vehicle we are developing.

Our approach combines robust perception and localization systems with advanced path planning and control algorithms, aiming to create a versatile and resilient software architecture. The primary objective of this project is to develop a system that enables our autonomous racing car to compete successfully in the dynamic events of the FSD competition. This work will contribute to the broader field of autonomous vehicle development and lay the groundwork for future participation in these competitions.

System Architecture

The autonomous racing system was developed as an advanced solution to meet the requirements of the Formula Student Driverless competition. The final architecture integrates both robust hardware components and sophisticated software systems, designed to ensure precision, reliability, and performance under the challenging conditions of dynamic racing events.

Hardware System

The hardware system was collaboratively designed by the various divisions within the BGRacing team - Electrical, Mechanical, and Autonomy. Each division brought its expertise to design the specific components of the system. Developed in strict accordance with the competition regulations, this system forms the critical foundation upon which the entire software architecture is built. The system consists of:

1. **Computing Unit**

- **Intel NUC11PHKi7C**: This core processing unit is powered by an Intel Core i7-1165G7, providing the necessary computational power for real-time processing of sensor data and decision-making tasks. It also includes an RTX 2060 GPU with 6GB GDDR6 memory for handling intensive graphical processing and parallel computing tasks.

2. **Sensor Suite**

- **LiDAR:** The Innoviz One LiDAR is essential for environmental perception, this LiDAR offers a 120-degree horizontal field of view with a 250-meter range. It provides high-resolution 3D point clouds, crucial for obstacle detection and mapping.
- **Camera**: The See3CAM_24CUG, a Full HD global shutter camera, equipped with an AR0234CS CMOS sensor, captures 1080p resolution at 60 fps. It is optimized for detecting fast-moving objects with minimal motion blur, a critical feature for high-speed racing environments.
- **IMU (Inertial Measurement Unit):** The Condor Pacific Ltd. IMU ensures vehicle stability and accurate localization by providing detailed measurements of orientation, acceleration, and rotational rates.

(c) Condor Pacific Ltd. IMU

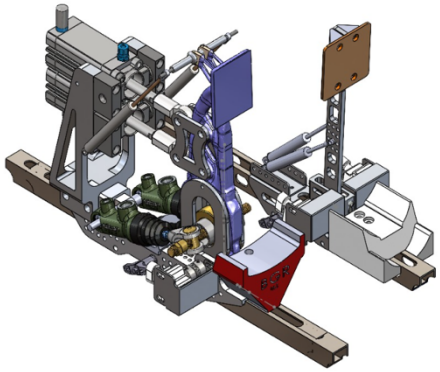
(c) See3CAM_24CU Camera

(c) Innoviz One LiDAR

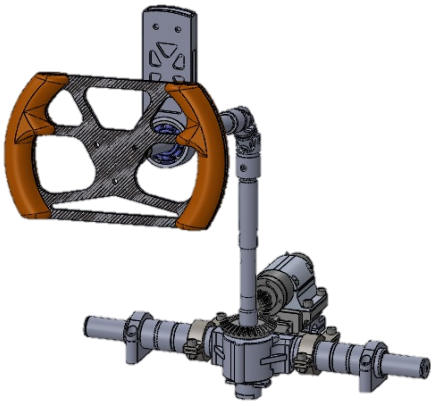
Fig. 4. Sensors suite for the autonomous vehicle system

3. Actuation Systems

- **Steering System**: A custom-engineered steering system designed for precision and responsiveness. It allows for rapid and accurate maneuvering, ensuring the vehicle can effectively navigate tight corners and complex track layouts with minimal delay.
- **Motors**: AMK in-wheel motors are utilized for 4-wheel drive, offering precise torque control and quick response times essential for maintaining optimal vehicle dynamics.
- **Emergency Braking System (EBS)**: A safety-critical feature designed to ensure the vehicle can be safely stopped in emergencies, compliant with the competition's stringent safety requirements.



(b) Emergency Braking System



(b) Steering System

Fig. 5. Self-developed actuation systems to control the vehicle

Software Architecture

The software system designed for the autonomous racing car integrates a range of custom and advanced modules, each configured to handle specific aspects of autonomous driving. The overall architecture, as illustrated in Fig. 6, includes custom drivers for sensors, object detection using YOLOv8, LiDAR detection with clustering, state estimation, SLAM-based mapping, path planning, and vehicle control using Pure Pursuit steering and PID speed control.

1. **Perception Module**

- **Sensor Drivers and Calibration**: Custom drivers were developed for each sensor in the system, including the LiDAR, cameras, and IMU. These drivers include necessary calibrations and adjustments to ensure accurate data collection that is configured to the car's specific characteristics.

- **Object Detection**: YOLOv8 is employed for real-time object detection and tracking. This model processes visual data from the camera, identifying and classifying the cones that define the track boundaries. YOLOv8 was chosen for its balance of speed and accuracy, making it ideal for the dynamic environment of autonomous racing.

- **LiDAR Data Clustering**: A clustering algorithm is applied to the 3D LiDAR data to group detected points into cones, which indicate the track boundaries. Using 3D LiDAR, this process not only identifies the cones but also precisely determines their spatial positions, which is essential for constructing a detailed and accurate map of the track.

2. **Localization and Mapping**

- **State Estimation**: The vehicle’s position and orientation are estimated using data from wheel rotations, steering angle, and the IMU. This fusion of data is processed through a Kalman filter to provide an accurate estimation of the vehicle’s current state, which is crucial for precise localization and control.

- **SLAM (Simultaneous Localization and Mapping)**: After comparing several SLAM algorithms, GraphSLAM was selected to build and update a map of the environment as the vehicle navigates the track. This choice was made for its effectiveness in providing accurate and up-to-date mapping in racing conditions.

3. **Path Planning**

- **Centerline Extraction**: Path planning begins by identifying the safest route on the track using Delaunay triangulation to determine the centerline between the detected boundaries. This method ensures that the vehicle stays well within the track limits while following the safest path.
- **Trajectory Planning**: Once the centerline is identified, the system generates a trajectory that the vehicle will follow, taking into account factors such as speed and turning radius.

4. **Control Systems**

- **Steering Control**: For steering, the system uses Pure Pursuit, a method that calculates the necessary steering angle to follow the planned trajectory. This approach ensures that the vehicle can smoothly navigate the track while maintaining high precision.
- **Speed Control**: PID (Proportional-Integral-Derivative) control is employed to manage the vehicle's speed. This method adjusts the throttle and braking to maintain the desired speed, responding to changes in the vehicle's environment and trajectory.

5. **System Integration**

- **Middleware**: The software modules are integrated using ROS 2 Humble (Robot Operating System) a robust middleware framework that facilitates real-time communication and data exchange between the various components of the system. ROS 2 Humble provides the flexibility and reliability needed for managing the complex interactions between perception, planning, and control systems in an autonomous vehicle.
- **Programming Languages**: The codebase is written in both Python and C++, leveraging the strengths of each language. Python is used for rapid prototyping and high-level logic, while C++ is employed for performance-critical tasks, ensuring that the system operates efficiently and responsively.

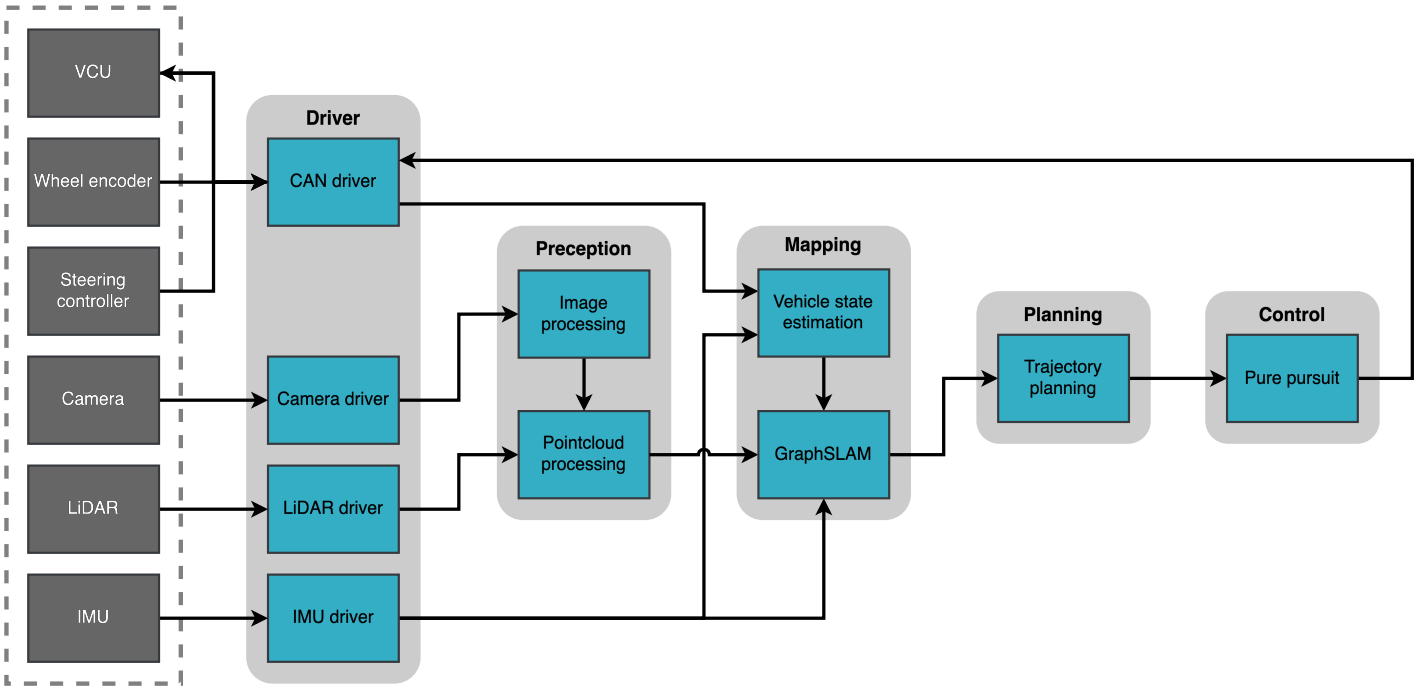


Fig. 6. Software-hardware architecture of the autonomous system

Approach Taken

This section reviews the approaches explored and ultimately selected during the final project, explaining the reasons for these choices and how they were or will be implemented. The engineering design process is detailed for each system component, aimed at achieving the project's objectives.

1. Perception Module

The perception module is the cornerstone of the autonomous vehicle's ability to understand and interact with its environment. It processes data from various sensors, including LiDAR, cameras, and an IMU, to create a comprehensive, real-time map of the surroundings. This information is crucial for the vehicle's decision-making processes, such as path planning and maintaining environmental awareness. The perception module integrates multiple sensor inputs to detect and track objects, estimate their positions, thereby enabling the vehicle to navigate safely and efficiently on the track.

Sensor Calibration

Proper calibration of the sensors is essential for ensuring that the data from each sensor is accurately aligned and can be used effectively by the perception and mapping systems. Given that each sensor is mounted at a different position on the vehicle, they each have their own coordinate system. Aligning these coordinate systems is crucial for creating a unified understanding of the vehicle's surroundings.

Coordinate System Alignment

To align the coordinate systems of the various sensors, the ROS2 tf2 package is used [6]. This tool is essential for managing the transformations between different coordinate frames in a robotic system. Each sensor operates within its own coordinate frame, and tf2 allows the transformation of data from these individual frames into a common reference frame, such as the vehicle's base frame.

As illustrated in Fig. 7, each sensor: LiDAR, camera, and IMU has its own coordinate system relative to the vehicle's base frame. Using tf2, all sensor data is accurately transformed and aligned with the base frame, allowing for the integration of data from different sensors, such as combining the camera's visual data with the 3D point clouds from the LiDAR, facilitating tasks like object detection and mapping.

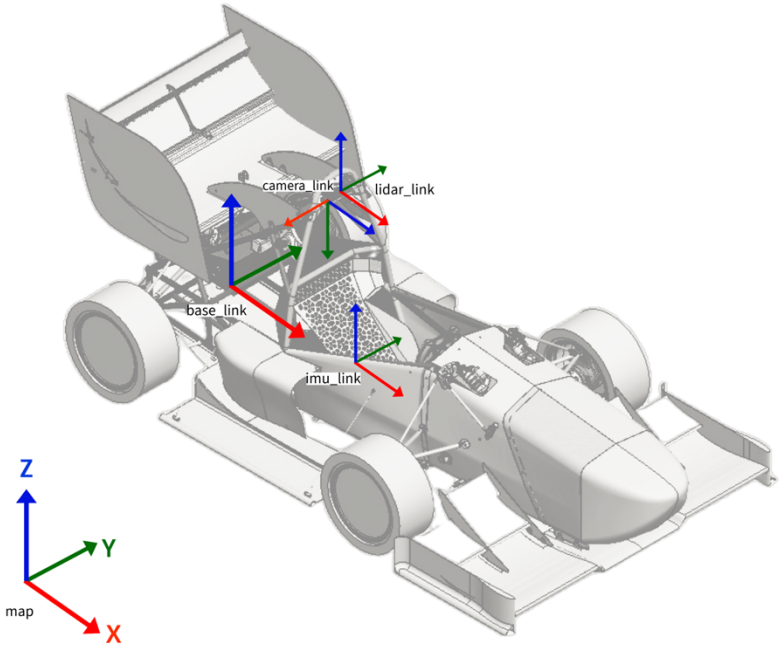


Fig. 7. Coordinate frames for the camera, LiDAR, and IMU in relation to the vehicle's base and map frames.

Camera Calibration

Camera calibration [7] involves determining both intrinsic and extrinsic parameters that define how the camera projects the 3D world onto a 2D image plane.

- **Intrinsic Parameters**: These parameters include the focal length f_x and f_y, the optical center (also known as the principal point) (c_x , c_y), and distortion coefficients that account for lens distortion. The intrinsic matrix K is defined as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The intrinsic matrix K transforms 3D points in the camera's coordinate system into 2D pixel coordinates on the image plane.

- **Extrinsic Parameters**: These define the rotation R and translation t that relate the camera's coordinate system to the world coordinate system (or another sensor's coordinate system). The relationship between a 3D point X in world coordinates and its corresponding 2D point x on the image plane can be described by:

$$x = K\ [R|t]\ X \quad (2)$$

where $[R|t]$ is the extrinsic matrix that transforms world coordinates to the camera's coordinate system.

The camera was calibrated using a checkerboard pattern, as shown in Fig. 8. Multiple images were captured from different angles and distances. Calibration software analyzed these images to determine the camera's intrinsic parameters (focal length, optical center, and distortion coefficients) and extrinsic parameters. The resulting calibration data was saved to a configuration file.

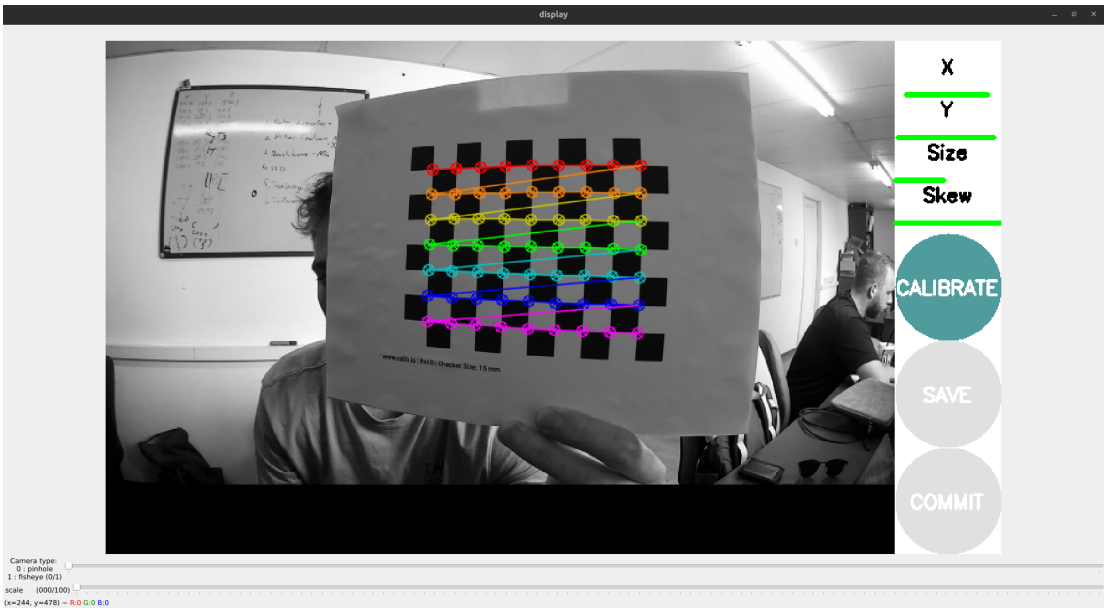


Fig. 8. Camera calibration process using a checkerboard pattern for determining intrinsic parameters such as skew, size, and axis alignment.

Custom Drivers

Several sensors required the development of custom drivers to integrate them into the ROS2 ecosystem:

- **Camera Driver**: The camera data is managed using the usb_cam driver, which reads the camera's calibration file and publishes the calibrated images to ROS. In addition to publishing the images, the usb_cam driver also provides camera information, such as intrinsic parameters, which is critical for integrating the camera data with other sensor data, particularly from the LiDAR.
- **LiDAR Driver**: The Innoviz One LiDAR initially came with a ROS 1 driver, which was not compatible with the ROS2 based system. To address this, a custom ROS2 driver was developed in C++. This driver is responsible for handling the data acquisition from the LiDAR, processing the raw data, and publishing it in the PointCloud2 format that other modules can use.
- **IMU Driver**: Similarly, a custom driver was developed for the IMU. This driver processes the raw inertial data (including acceleration, gyroscopic measurements, and orientation) and publishes it within the ROS2 framework. This data is crucial for tasks such as vehicle state estimation and control.

Cone Detection Using Camera

Cone detection is essential for the autonomous vehicle, as cones define the track boundaries in racing environments. Initially, a YOLOv5 model was used for this task, but the decision was made to transition to YOLOv8 due to its enhanced features, including object tracking capabilities that assist in mapping. The primary goal of detecting cones is to classify them by color and size, enabling the use of this information in path planning algorithms. To achieve accurate and real-time detection and classification of cones, the YOLOv8 model was specifically implemented and trained for this purpose.

YOLOv8 Model Training

- **Bounding Box and Segmentation**: The YOLOv8 model was trained for both object detection using bounding boxes and segmentation. This dual capability allows the model to identify and classify cones and outline their shapes in the images. The segmentation feature enhances the accuracy of cone classification by providing detailed shape information, which is especially useful for distinguishing between different types of cones and ensuring reliable detection even when cones are close together or partially obscured.
- **Custom Dataset**: The model was trained on a custom dataset specifically designed for this application. This dataset includes images of five types of cones commonly used on the track: yellow cones, blue cones, small orange cones, large orange cones, and unknown cones. The dataset was adapted from the FSOCO dataset [8], and full acknowledgment is given to the original creators for their work. The dataset was converted to the appropriate format for the YOLOv8 model using custom code developed for this purpose, but no additional images were added.
- **Training Process**: The training process involved fine-tuning the YOLOv8 model on this dataset using commercial tools, and the model was trained multiple times on Google Colab until stable and satisfactory results were achieved. Although data augmentation was not performed during this training, it remains a potential future enhancement to improve the model's robustness and generalization to real-world scenarios.

Integration with ROS

- **ROS Node for YOLOv8**: The trained YOLOv8 model is deployed within a ROS2 node, which subscribes to the image_raw topic from the camera node. The node runs the YOLOv8 inference in real-time, classifying and segmenting cones in the environment. The ROS2 implementation of YOLOv8 was based on the code from the ultralytics_ros repository [9], which was adapted to meet the specific needs of this project.
- **Cone Classification**: Each detected cone is assigned a class label and a confidence score. A segmentation mask is also generated, providing a pixel-level outline of the cone. The entire process runs at a sufficiently fast speed to maintain a frame rate of 30 fps, leveraging the computer's GPU for efficient real-time performance. The code is implemented in Python, ensuring flexibility and ease of integration within the ROS2 framework.
- **Output to ROS Topics**: The detection and segmentation results are published to ROS2 topics for use by other components in the vehicle's perception and mapping systems. Published Topics: Plotted images (result_image_topic) and Detected objects (result_topic)

Results and Visualization: Detection results, including bounding boxes and segmentation masks, are displayed in RViz, enabling real-time monitoring and providing a clear visual representation of the cones and their classifications. As shown in Fig. 9, the YOLOv8 model successfully detects and classifies cones in the environment, accurately marking their positions and shapes.

Fig. 9. YOLOv8 object detection and segmentation of cones, identifying different cone types with confidence scores.

Cone Detection using LiDAR

LiDAR is a crucial sensor in autonomous systems, offering precise 3D spatial data that directly measures distances, making it ideal for detecting objects like cones on a racetrack. Unlike cameras, LiDAR operates reliably regardless of lighting conditions. After evaluating various detection methods, a clustering-based machine learning approach was chosen due to its effectiveness in grouping LiDAR point cloud data into clusters, allowing for accurate cone detection and localization.

Clustering is a key technique in data analysis that groups objects based on spatial similarities. In this context, it is particularly useful for identifying cones on a racetrack by organizing relevant LiDAR points into distinct clusters. This method allows the perception system to reliably detect and track cones, supporting critical navigation and decision-making tasks in autonomous racing.

Clustering Methods Considered

After evaluating four clustering methods: DBSCAN (Spatial and Grid) and Euclidean Clustering (Spatial and Grid), we selected Euclidean Clustering (Spatial) for our project.

- **DBSCAN (Spatial)**: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clusters points based on density by grouping points within a specified distance of a core point, expanding clusters outward from these core points. It is effective in handling noise and finding clusters of varying shapes, making it suitable for complex environments. However, the performance of DBSCAN is highly sensitive to its parameter settings, such as the distance threshold and the minimum number of points required for clustering. Incorrect parameter choices may lead to missing clusters in sparse areas or merging distinct clusters.
- **DBSCAN (Grid)**: This variant of DBSCAN operates on a grid, dividing space into cells and clustering points based on the density within each cell. The grid-based approach simplifies computation, making it more efficient in structured environments. However, the accuracy of this method depends heavily on grid resolution. A coarse grid can miss finer details, while a fine grid increases computational complexity. The fixed grid structure can also introduce inaccuracies in cluster boundaries.
- **Euclidean Clustering (Spatial)**: Euclidean clustering groups points based on their distance from each other, forming clusters when points fall within a predefined range. This method is

straightforward and works well in environments where objects are distinct and well-separated, such as cones on a racetrack. However, in cluttered or dense environments, it may struggle to distinguish between closely packed or overlapping objects, leading to less accurate clustering.

- **Euclidean Clustering (Grid)**: Similar to spatial Euclidean clustering, this method operates within a grid framework, grouping points based on proximity within grid cells. This grid-based approach enhances processing speed and scalability, especially when handling large datasets in real-time. However, the chosen grid resolution significantly impacts the accuracy of clustering. A coarser grid can oversimplify clusters, while a finer grid requires more computation. The method also faces challenges in separating overlapping clusters due to the rigid grid structure.

Chosen Method: Euclidean Clustering (Spatial)

Euclidean Clustering (Spatial) was ultimately selected for its simplicity and effectiveness in the autonomous racing application. This method accurately groups nearby points, making it particularly suitable for detecting cones on a racetrack. The clustering technique is integrated with the YOLOv8 object detection model to process LiDAR data, resulting in a robust perception system capable of detecting, classifying, and tracking cones with high precision. This ensures the vehicle can create an accurate map of the track effectively and safely.

Algorithm Steps for LiDAR-Based Cone Detection

i. **Ground Filtering using Patchwork++:** The Patchwork++ algorithm [10] removes ground points from the LiDAR data, focusing on objects above the ground, such as cones. By filtering out irrelevant ground points, this step reduces noise and ensures that only relevant data is processed for subsequent detection and clustering tasks.

ii. **Spatial Filtering:** Spatial filtering refines the LiDAR point cloud data by removing points based on their X, Y, and Z coordinates, retaining only those that fall within a predefined 3D region of interest. This step is crucial for focusing on the areas directly relevant to the vehicle's environment, reducing computational load, and enhancing the accuracy of further processing. By limiting the analysis to specific spatial boundaries, this filtering ensures that only pertinent data is considered.

iii. **Euclidean Clustering:** This step groups the filtered LiDAR points into clusters likely representing individual cones. The Euclidean clustering algorithm groups points based on their spatial proximity, with each cluster representing a potential cone. This step is essential for accurately identifying and isolating cones on the racetrack, providing critical positional data for generating an accurate map of the environment.

Alternative Algorithm- Projection and Integration with Yolov8 Detection (Optional): An alternative approach involves integrating LiDAR data with camera data to improve detection accuracy. This step maps 3D LiDAR points onto a 2D plane based on camera images. The algorithm projects the camera's visual data onto the LiDAR data, focusing clustering only on regions where the camera detects cones. This selective clustering reduces false positives and allows for the classification of cone colors, which LiDAR alone cannot achieve.

The block diagrams in Fig. 10 and Fig. 11 illustrate the two algorithm workflows. Fig. 10 shows the process when using only LiDAR data, while Fig. 11 depicts the enhanced algorithm that integrates LiDAR and camera cone detections from YOLOv8 for improved detection and classification.

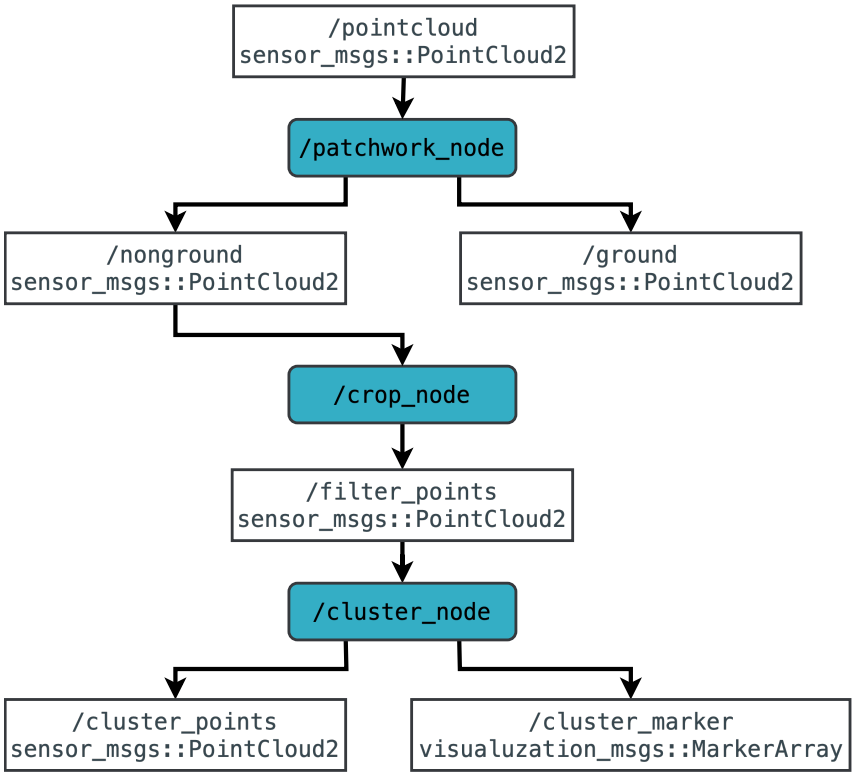


Fig. 10. Point cloud processing pipeline showing nodes for ground segmentation, point filtering, and clustering.

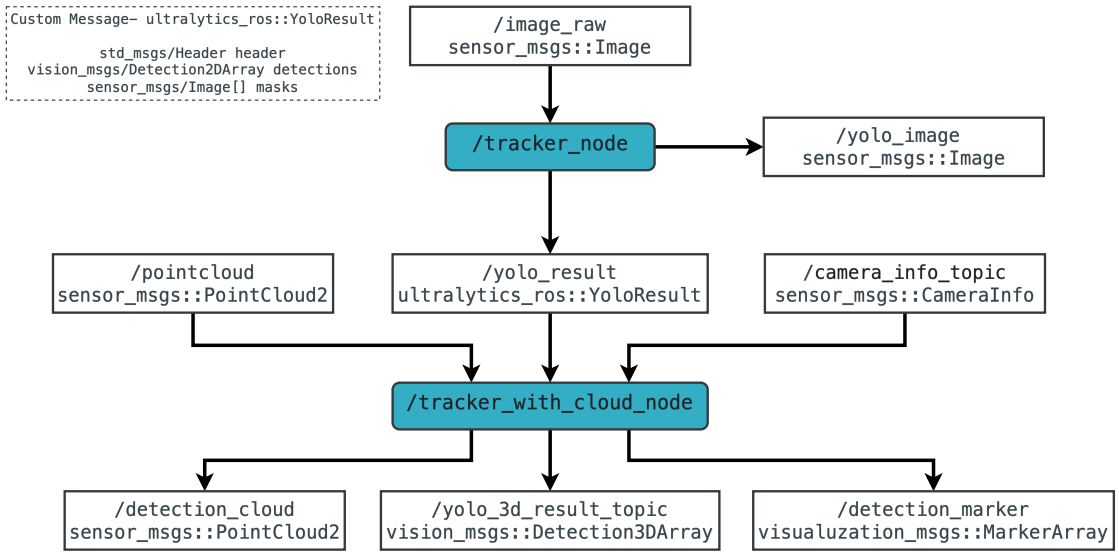


Fig. 11. Data flow diagram showing YOLO detection integrated with point cloud processing through tracking nodes.

2. Localization and Mapping Module

Localization System

Localization in our autonomous racing vehicle involves estimating the vehicle's position, orientation, and velocity in real time. This process is key to precise control and navigation on the racetrack. With guidance from General Motors (GM), we implemented the localization system using a Kalman Filter, which combines predictions from a motion model with real-time sensor data for accurate state estimation.

Introduction to the Kalman Filter

The Kalman Filter estimates the state of a dynamic system by predicting the system's next state using a model and then refining it with sensor measurements. The filter requires:

- **A State Model**: Predicts how the system's state evolves over time.
- **A Measurement Model**: Relates the actual measurements (e.g., sensor data) to the state.

The Kalman Filter's ability to blend model predictions with sensor data provides highly accurate localization, even under challenging conditions like rapid turns and accelerations.

Motion Model

Kinematic Bicycle Model (KBM)

The Kinematic Bicycle Model (KBM) is a simplified representation of vehicle dynamics, commonly used in autonomous driving. It reduces complexity by treating the vehicle as a single point mass with two wheels: a front and a rear. This model assumes no tire slip and focuses on key parameters like steering angle and velocity, which are critical for real-time state estimation. The equations governing the KBM are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\ cos(\theta) \\ v\ sin(\theta) \\ \frac{v\ tan(\delta)}{L} \\ a \end{bmatrix} \tag{3}$$

Where: x, y are the global coordinates, θ is the heading angle, v is the velocity, δ is the steering angle, L is the wheelbase, a is the acceleration.

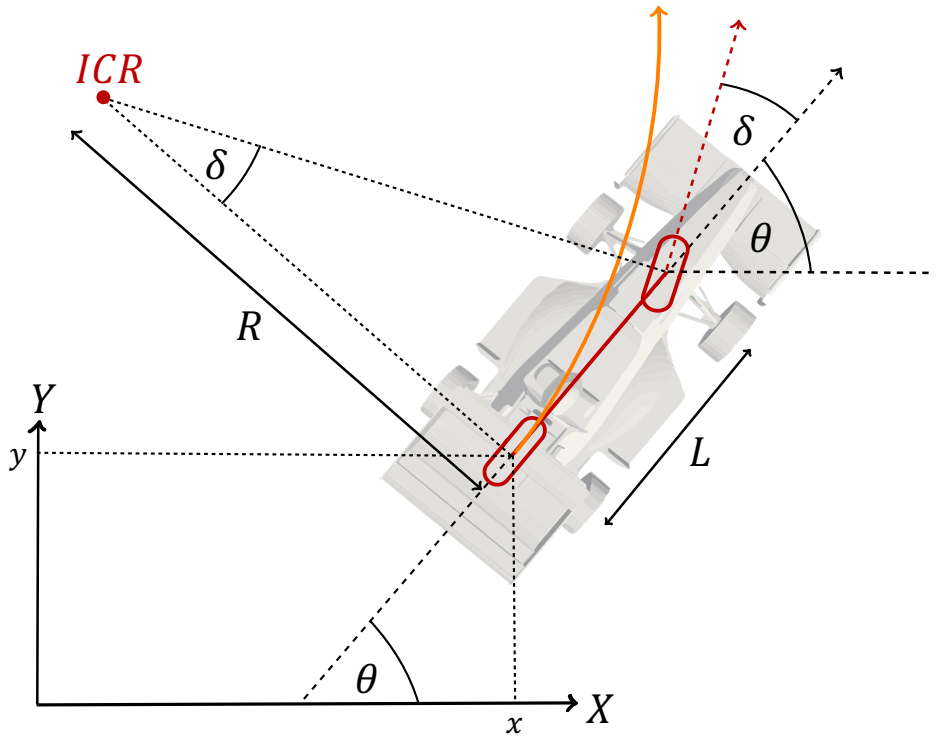


Fig. 12. Vehicle kinematic model showing the Instantaneous Center of Rotation (ICR) and path curvature.

Discrete-Time Implementation

To implement this continuous model in a digital system, the 4th Order Runge-Kutta Method is used to convert the differential equations into a discrete-time model:

$$x_k = f(x_{k-1}, u_k, w_k) \tag{4}$$

Where: x_k is the vehicle state at time step k, u_k represents control inputs such as steering angle (δ) and acceleration (a), w_k introduces process noise into the model, which accounts for uncertainties in the model.

Measurement Models

Motor RPM

The vehicle's speed is derived from motor RPM, accounting for the transmission ratio between the motor and wheels. The relationship is given by:

$$\omega_{motor} = \frac{v_k}{r_{tire} \times gear\_ratio} \tag{5}$$

Where: ω_{motor} is the angular velocity of the motor (RPM), r_{tire} is the tire radius, $gear\_ratio$ is the transmission ratio.

IMU (Inertial Measurement Unit)

The IMU provides measurements of linear acceleration and angular velocity. The acceleration readings are transformed to the vehicle's frame using the rotation matrix R_{VW}, which rotates the vector from the vehicle frame to the world frame. The model is given by:

$$a_{imu} = a_{vehicle} + R_{VW}(\omega_v \times (\omega_v \times r_{imu})) \tag{6}$$

Where: a_{imu} is the acceleration measured by the IMU, $a_{vehicle}$ is the acceleration of the vehicle, R_{VW} is the rotation matrix from the vehicle frame to the world frame, ω_v is the angular velocity

vector of the vehicle, r_{imu} is the position vector of the IMU sensor relative to the vehicle's reference frame.

Kalman Filter Integration

The Kalman Filter is used to fuse the motion and measurement models for state estimation. The filter operates in two main steps:

- **Prediction Step**

  In this step, the state is propagated forward using the system dynamics:

$$X_{k|k-1} = f(X_{k-1}, U_{k-1}) \tag{7}$$

  Where $X_{k|k-1}$ is the predicted state vector at time k given the previous state X_{k-1} and control input U_{k-1}.

- **Update Step**

  The update step refines the state estimate using sensor measurements:

$$K_k = P_{xkzk}\, S_{yk|k-1}^{-1} \tag{8}$$

  Where K_k is the Kalman gain, which optimizes the state estimate based on the measurement uncertainty. P_{xkzk} is the covariance between the state and the measurement, and $S_{yk|k-1}$ is the predicted measurement covariance.

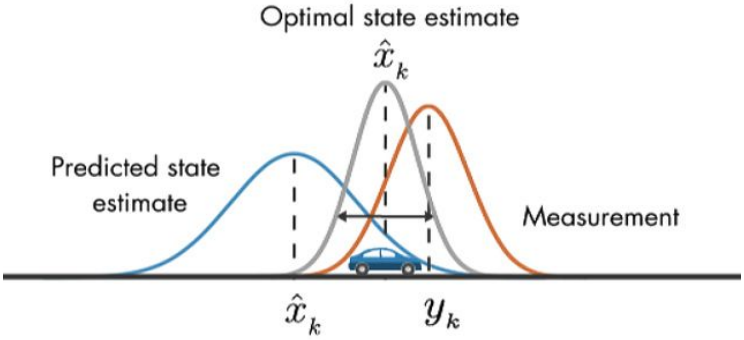


Fig. 13. Illustration of state estimation combining predicted state and measurement to achieve an optimal estimate.

Mapping System

The goal of the mapping system is to use cone detections and vehicle state data to build an accurate map of the racetrack in real time as shown in Fig. 14. This map serves as the basis for optimal path planning. To enhance performance, the system continuously updates and retains the mapped path, allowing it to refine the route for better navigation in later laps.

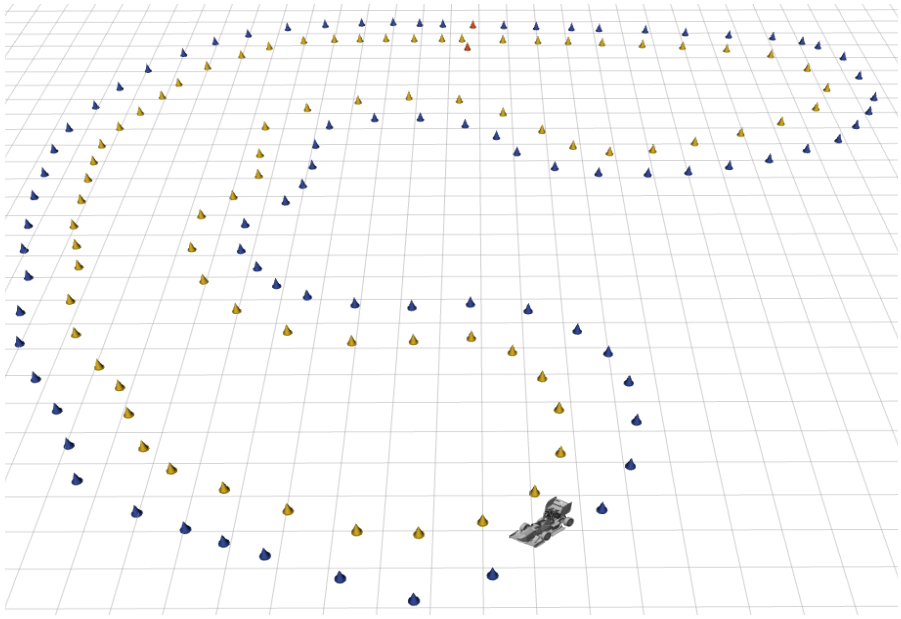


Fig. 14. Simulated racetrack with blue and yellow cones marking the course boundaries for the autonomous vehicle.

Challenges in Map Building with Object Detection and Localization

The vehicle currently relies on cone detection and localization data to build a map. However, this approach has several challenges:

- **Global Reference Integration:** Transforming sensor data from the vehicle's frame to a global frame introduces potential inaccuracies, which are critical to address for precise mapping.
- **Sparse Map Representation:** The map is sparse, focusing solely on cone positions. This design depends heavily on the accuracy of cone detection and localization.
- **Localization Drift:** Relying on IMU and motor RPM data can lead to cumulative errors over time, distorting cone positions on the map.

Enhancing Map Accuracy with GPS

Integrating GPS provides a global reference to correct localization drift and improve map accuracy. However, GPS updates less frequently than IMU data, potentially causing minor discrepancies in high-speed environments. Additionally, maintaining consistent cone positions across laps can be challenging due to GPS accuracy variations.

Introduction to SLAM (Simultaneous Localization and Mapping)

Given the limitations of relying solely on cone detection, localization, and GPS, SLAM offers a robust alternative for accurate map-building and localization. SLAM integrates data from sensors like cameras, LiDAR, IMUs, and GPS to continuously update the map while accurately determining the vehicle's position, allowing for reliable navigation even in dynamic or complex environments.

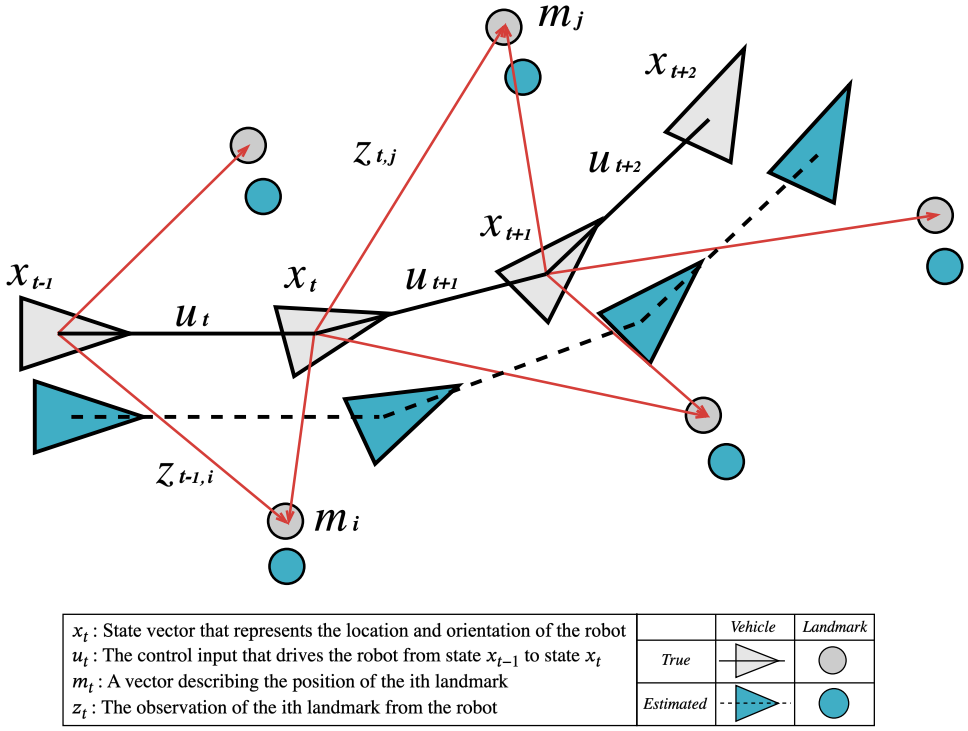


Fig. 15. Schematic diagram of simultaneous localization and mapping (SLAM).

SLAM Benefits in Autonomous Racing

- **Improved Localization Accuracy:** SLAM refines the vehicle's position relative to cones, reducing GPS inaccuracies and sensor drift.

- **Dynamic Map Building:** SLAM updates maps in real-time, adapting to environmental changes and maintaining an accurate track understanding.
- **Handling of Loop Closure:** SLAM corrects accumulated errors when the vehicle revisits previously mapped locations.

SLAM Approaches for Autonomous Racing

Based on a comprehensive review of various SLAM methodologies and their application in autonomous racing, particularly in the Formula Student Driverless studies presented in [11] and [12], the following SLAM approaches have been evaluated and recommended:

- **EKF-SLAM:** Suitable for environments with few landmarks and low-noise measurements. It's simple to implement but becomes computationally expensive with more landmarks and struggles in dynamic environments.
- **FastSLAM:** Uses a particle filter for path estimation and Kalman filters for landmarks, handling many landmarks with low computational cost. However, it may require complex tuning for noisy or dynamic environments.
- **GraphSLAM:** Optimizes a graph of poses and landmarks, making it effective for large-scale environments and accurate mapping, though it requires significant processing power.
- **Visual-SLAM:** Effective in environments with rich visual textures, but it can struggle with lighting changes and repetitive textures.
- **LiDAR-SLAM:** Provides high accuracy in environments with clear structural features and varying lighting conditions but can be complex and costly to integrate.

For our needs, GraphSLAM is the most suitable SLAM approach due to its balance of real-time performance, accuracy, and scalability, making it ideal for dynamic racing scenarios where the environment evolves over time. For more information on GraphSLAM, see [13]. Additionally, we can explore several ready-made SLAM algorithms that are compatible with ROS2, such as Fast_LIMO, GLIM, lidarslam_ros2, LIO_SAM, ORB_SLAM3_ROS2 and RTAB-Map. These algorithms offer flexibility in implementation and provide alternative approaches that could be tested to optimize our mapping and localization system further.

3. Path Planning Module

Path planning is a crucial element in autonomous vehicle navigation, particularly in racing environments where precision and speed are essential. The main challenge is to determine the optimal route for the vehicle, ensuring it stays within track boundaries while maximizing efficiency. In the Formula Student competition, the track is marked by cones of different colors, defining the boundaries. This presents two distinct scenarios for path planning:

- **Colored Map:** The system uses color information from the mapped cones (e.g., yellow and blue) to differentiate left and right boundaries, enabling more precise centerline calculations and improved path planning.
- **Uncolored Map (LiDAR-only):** In cases where only LiDAR data is available and the track boundaries are not distinguished by color, the system must rely solely on spatial data to determine the track boundaries and find the centerline.

In both scenarios, the main goal is to accurately determine the centerline of the track, which represents the safest path the vehicle should follow, keeping it equidistant from the boundaries for optimal speed and safety.

The path planning system operates on a global map generated by the SLAM (Simultaneous Localization and Mapping) system. The input data includes:

- A comprehensive map of cone positions, including their locations and colors (if available).
- The vehicle's global location and orientation.

The system's output is the driving path, specifically the centerline of the track, representing the optimal route for the vehicle to follow.

Path Planning Challenges

The path planning system must address several challenges beyond calculating the centerline:

- **Misidentification of Cones:** The presseption/mapping system might incorrectly identify objects or other cones as part of the track boundaries, leading to potential errors in centerline calculation and unsafe paths.
- **Cone Detection Failures:** The presseption/mapping system might miss detecting some cones due to various factors like poor lighting, sensor limitations, or obstructions. Missing cones can create gaps in track boundary data, complicating the accurate determination of the track.

The path planning system must not only calculate the centerline but also be robust enough to handle scenarios involving misidentified or undetected cones. This requires advanced algorithms capable of filtering out false positives and compensating for missing data to ensure the vehicle remains on the correct path.

Path Planning Methods Researched

Several path planning methods were explored to identify the most effective approach for autonomous racing. Each method was assessed based on its accuracy in determining the track's centerline, its ability to handle dynamic changes in the environment, and its seamless integration with the overall system. The following is an overview of the key methods:

Delaunay Triangulation

Delaunay Triangulation is a geometric method used to create a mesh of non-overlapping triangles from a set of points in a plane. The primary goal of Delaunay Triangulation is to maximize the minimum angle of all the angles of the triangles in the triangulation, avoiding narrow triangles. This method is widely used in various fields, including computer graphics, geographic information systems (GIS), and path planning, due to its ability to create well-structured and efficient meshes.

In autonomous racing, Delaunay Triangulation is used to determine the safest path between track boundaries marked by cones, as described in [14]. By connecting the cones with triangulation, the algorithm calculates the midpoints of the edges of these triangles, which represent the centerline of the track. This centerline is then used as the path for the vehicle to follow.

Steps in Delaunay Triangulation:

i. **Delaunay Triangulation of Cone Positions:**

The process began with the application of Delaunay Triangulation to the positions of the cones marking the track boundaries. This step generated a network of triangles connecting the cones, ensuring comprehensive coverage of the track area.

ii. **Filtering and Identifying Relevant Triangles:**

Triangles where all vertices (cones) shared the same color were identified as outside the track boundaries and were filtered out. The remaining triangles, with vertices of different colors, were considered to lie within the track and were crucial for defining the path.

iii. **Selecting Internal Edges and Calculating Midpoints:**

Next, the internal edges of these relevant triangles (those connecting cones of different colors) were identified. The midpoints of these edges were calculated to form the initial central path. These midpoints ensured that the vehicle would remain equidistant from the track boundaries, providing a balanced and optimal route.

iv. **Smoothing the Path Using Interpolation:**

Finally, interpolation techniques were applied to the sequence of midpoints to smooth the path. Spline interpolation was particularly effective, creating a continuous and optimized trajectory, minimizing abrupt changes in direction, and ensuring stability at high speeds.

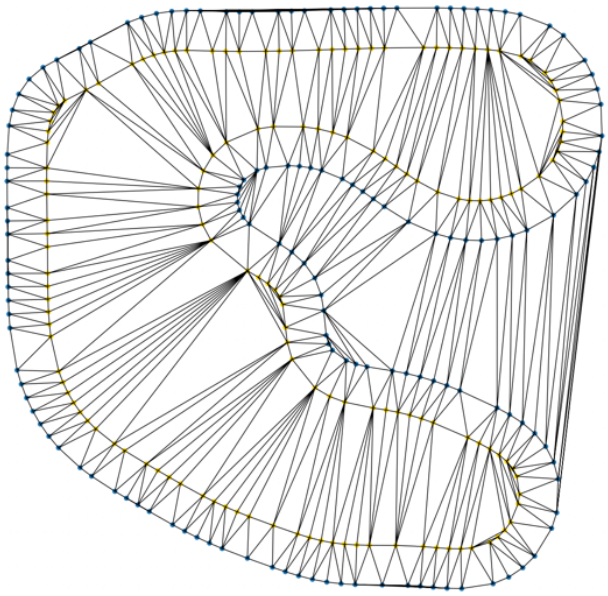


Fig. 16(a). Initial Delaunay Triangulation applied to cone positions, forming triangles that cover the entire track.

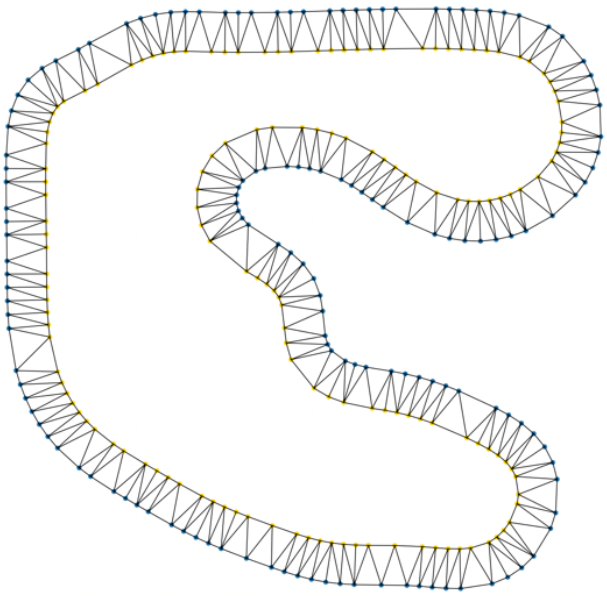


Fig. 16(b). Filtering process that removes irrelevant triangles, isolating the relevant triangles within the track boundaries.

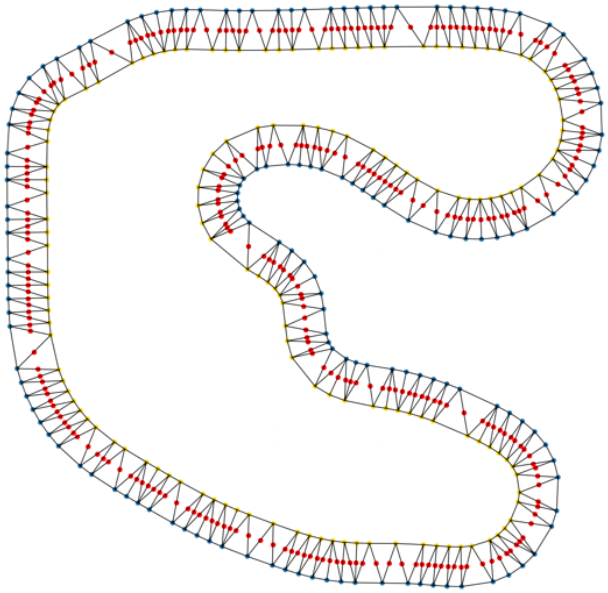


Fig. 16(c). Midpoint calculation of internal edges connecting cones of different colors to define the central path.

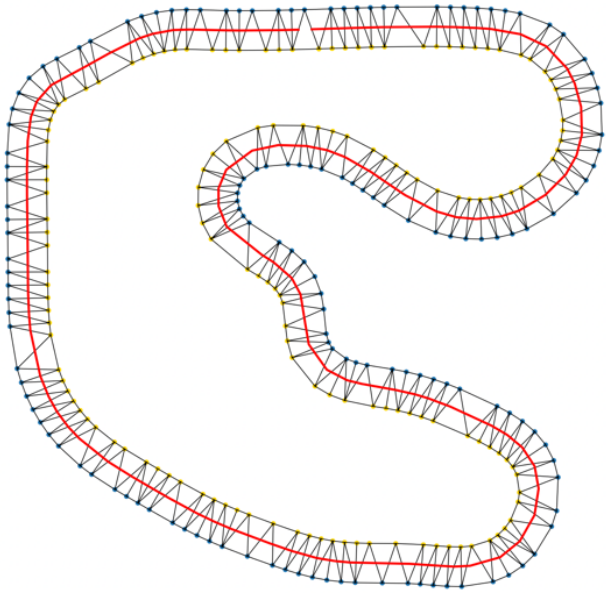


Fig. 16(d). Smoothed central path using spline interpolation, providing a continuous and optimized trajectory.

Additional Considerations for Delaunay Triangulation

- **Iterative Operation:** The algorithm is designed to improve iteratively, making it useful during initial laps when the map is incomplete. As more data is collected, the path is refined in subsequent laps.
- **Handling Missing and Misidentified Cones:** The algorithm can accommodate some missing cones by using available data to maintain a feasible path, but its performance drops as more cones are missing or misidentified. Misidentifications are particularly problematic, as the algorithm relies on accurate cone color information, and errors can lead to incorrect path calculations, potentially causing the vehicle to deviate from the intended route.
- **Dependence on Colored Maps:** This algorithm requires colored maps to function, as cone color is used to filter triangles and identify internal edges. Without color data, the algorithm cannot accurately determine track boundaries or produce a reliable path.

Berlin Team's Open-Source Path Planning Project

One significant project we integrated into our research was an open-source path planning algorithm developed by the FaSTTUBe team from Berlin [15]. This algorithm is designed to handle both colored and uncolored maps, making it versatile for different racing scenarios. Additionally, it addresses the challenge of missing cones, allowing the vehicle to navigate effectively even when some track data is incomplete.

Custom Trajectory Building Based on Academic Research

Inspired by the paper [16], we also explored a custom trajectory building approach. This method emphasizes the construction of the track's path by taking into account specific characteristics of the racing environment.

i. **Initial Cone Detection and Classification**:
   - Identify the two closest cones.
   - Classify the cones as left (yellow) or right (blue).
   - Add the classified cones to LeftBoundaryCones and RightBoundaryCones lists.

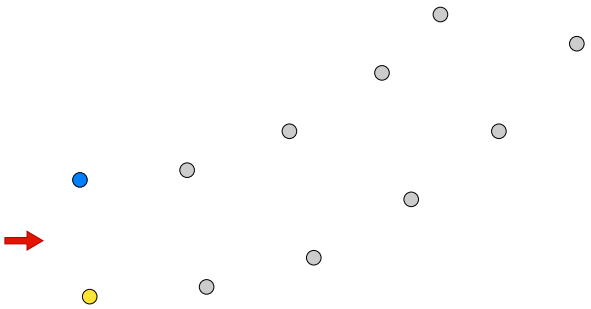


Fig. 17(a). Initial detection of left (blue) and right (yellow) cones for boundary setup.

ii. **Iterative Cone Matching**:
   - For each cone in LeftBoundaryCones and RightBoundaryCones, define a search region based on the current angle and distance.
   - Search within this region for the next cone, classifying it based on its relative position.
   - Add matched cones to the corresponding boundary lists (LeftBoundaryCones or RightBoundaryCones) and repeat until no more new cones can be classified.

iii. **Curvature Adjustment**:

- Adjust the search region's orientation according to the angle between the most recently added cone and its predecessor to account for track curves.

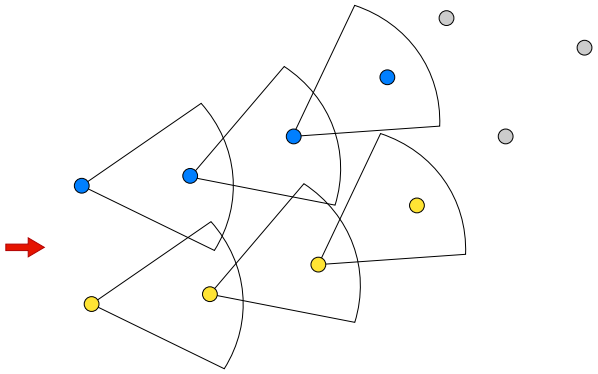


Fig. 17(b). Iterative cone matching to extend the left and right boundaries.

iv. **Perpendicular Cone Detection**:

- For each segment between consecutive cones in the left and right boundary lists, compute the perpendicular bisector.
- Search along this bisector for a cone from the opposite side to pair and form a boundary segment.

v. **Midpoint Calculation**:

- For each matched boundary pair, calculate the midpoint and store these midpoints in MidlinePoints as the candidate points for the track's centerline.

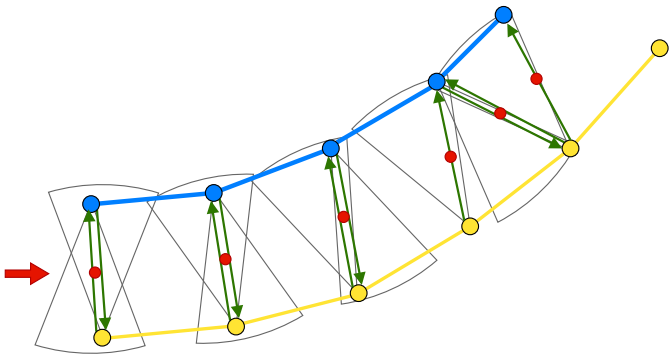


Fig. 17(c). Perpendicular cone detection and midpoint calculation for the centerline.

vi. **Path Smoothing**:

- Apply spline interpolation or other smoothing techniques to MidlinePoints to generate a continuous and smooth centerline.

vii. **Reclassification and Adjustment**:

- Revalidate the classification of cones periodically and make necessary adjustments to ensure the path's integrity.

viii. **Final Output**:

- Output the smoothed midline as the vehicle's driving path trajectory.

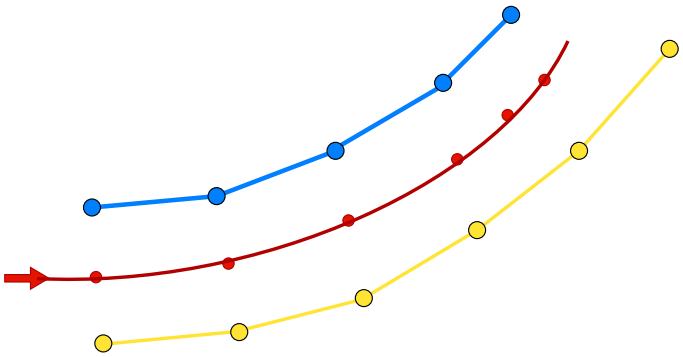


Fig. 17(d). Path smoothing using interpolation to generate the optimized centerline.

4. Control Module

The control system is a critical component in ensuring that the autonomous vehicle accurately follows the planned path while maintaining stability and performance. For the Formula Student competition, where precision and speed are paramount, a robust control system is necessary to manage the vehicle's dynamics as it navigates the track.

Kinematic Bicycle Model

The KBM plays a crucial role in our control system by offering a simplified yet effective way to simulate the vehicle's motion. As detailed in the localization section, this model captures the essential dynamics of the vehicle's movement, focusing on the relationship between the steering angle, velocity, and position, key factors in accurately following the planned path.

Pure Pursuit Control

The Pure Pursuit algorithm is a geometric control method used for path tracking in autonomous vehicles. Its purpose is to calculate the steering angle (δ) required for the vehicle to follow a designated path by continuously targeting a look-ahead point.

How Pure Pursuit Works

i. **Look-Ahead Point Selection**

The algorithm identifies a look-ahead point (TP) on the path, positioned at a fixed distance ahead of the vehicle, known as the look-ahead distance (l_d).

This point serves as a reference for steering adjustments, ensuring that the vehicle follows the intended path.

ii. **Calculating the Steering Angle**

The necessary steering angle δ is calculated to direct the vehicle along a circular path toward the look-ahead point.

The vehicle's path curvature is defined by the radius R, determined using the formula:

$$R = \frac{l_d}{2\sin(\alpha)} \tag{9}$$

α is the angle between the vehicle's current heading and the line to the target point, calculated as:

$$\alpha = \arctan\left(\frac{y_{tp} - y_{rear}}{x_{tp} - x_{rear}}\right) \tag{10}$$

iii. **Steering Angle Formula**

Finally, the steering angle δ is computed using:

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{l_d}\right) \tag{11}$$

Where L is the vehicle's wheelbase.

This method effectively keeps the vehicle on the desired trajectory by dynamically adjusting the steering angle as the vehicle moves. The Pure Pursuit algorithm is particularly effective for smooth paths, ensuring precise path following and stability during navigation.

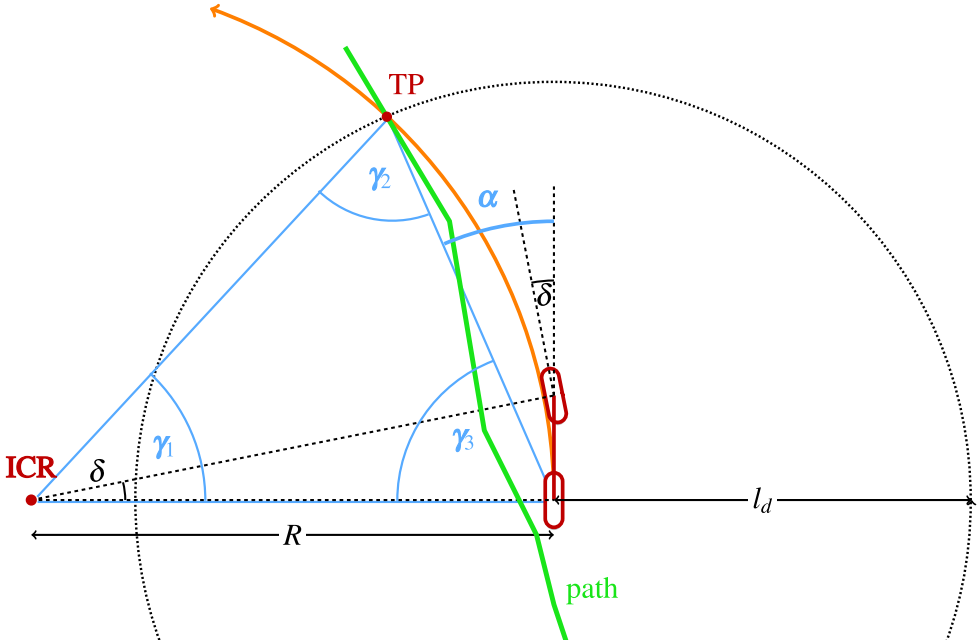


Fig. 16. Pure Pursuit control diagram illustrating the relationship between the look-ahead distance (l_d), target point (TP), and path curvature for trajectory tracking.

Speed Control using PID

The PID controller regulates the vehicle's speed by minimizing the error between the desired speed v_{target} and the actual speed v_{actual}. The control input $u(t)$ is calculated as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{12}$$

Where $e(t) = v_{target} - v_{actual}$ is the speed error. K_p, K_i, K_d are the proportional, integral, and derivative gains.

The PID controller adjusts the throttle and brake inputs to maintain the target speed, ensuring smooth and stable motion. This speed control is integrated with the Pure Pursuit algorithm, enabling accurate path tracking by continuously adjusting the look-ahead point based on the vehicle's speed.

Model Predictive Control (MPC)

Model Predictive Control is an advanced control strategy used to optimize control inputs by predicting the future states of a system over a finite time horizon. It is widely applied in dynamic environments where maintaining control while respecting system constraints is crucial.

Core Concepts of MPC

i. **Prediction Horizon:**

MPC predicts the future states of a system over a finite time horizon, using a dynamic model of the system. The horizon is divided into discrete time steps, where the system's behavior is simulated.

ii. **Dynamic Model:**

A mathematical model, like the kinematic bicycle model, is used to forecast the vehicle's state evolution, which typically includes position, velocity, and orientation. The state update is given by:

$$x_{k+1} = Ax_k + Bu_k + C \tag{13}$$

where A, B, and C are system matrices.

iii. **Cost Function:**

MPC minimizes a cost function J, which penalizes deviations from the desired path, excessive control inputs, and large changes in inputs:

$$J = \sum_{k=0}^{T} (\left(x_k - x_{ref}\right)^T Q \left(x_k - x_{ref}\right) + {u_k}^T R \, u_k \tag{14}$$

where x_{ref} is the reference trajectory.

iv. **Optimization and Constraints:**

The control inputs u_k are optimized to minimize J while respecting constraints, such as limits on steering, acceleration, and speed. The optimization is recalculated at each time step, applying only the first control input.

v. **Receding Horizon:**

After applying the first input, the horizon "recedes," and the process repeats, allowing continuous adaptation to changes.

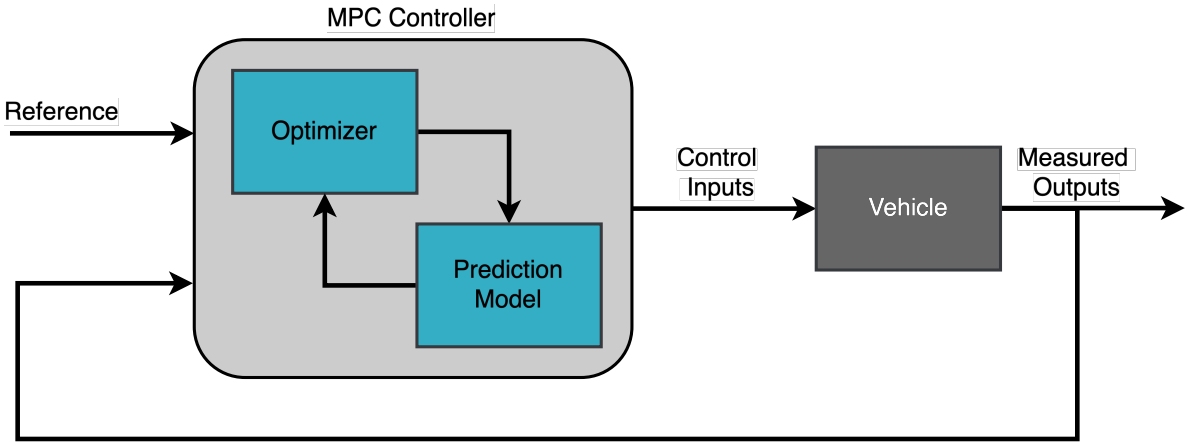


Fig. 17. Model Predictive Control (MPC) architecture showing the interaction between the optimizer, prediction model, and vehicle.

While MPC is effective, its computational demands can lead to lengthy run times, making it challenging for real-time applications in high-speed environments. We have initiated work on implementing MPC, but its current run times are too long for practical use in our autonomous vehicle, necessitating further optimization.

Final Product Acceptance Tests

The final acceptance tests were conducted through a series of data collection sessions at the Motor City track. These sessions involved capturing sensor data under real racing conditions and replaying it later for analysis and system validation. The primary goal was to test the system's performance in realistic environments, allowing us to identify areas for improvement and verify that our system meets competition requirements.

For data collection, we used a custom recording method instead of the standard rosbag approach. Our custom Python script captures data from the vehicle's sensors (camera, LiDAR, IMU) and republishes it within ROS, simulating real-time conditions. This setup allowed us to precisely replicate the behavior of the autonomous system without the need for repeated live testing on the track.

Fig. 20. Data recording and testing session at Motor City racetrack during a test drive day.

Testing tools included:

- **Custom Data Recorder:** A script designed to log sensor data, enabling repeated tests and analysis.
- **Simulation and Replay:** Sensor data replayed in ROS to evaluate algorithm performance in scenarios like cone detection and mapping.
- **RViz and ROS 2 Visualization:** Used for real-time and post-processing analysis of cone detection, localization, and path planning.

The following sections will detail the procedures and results for each major component, starting with the cone detection tests, which focus on evaluating the performance of both camera and LiDAR-based detection systems.

Camera-Based Cone Detection

The first test focuses on evaluating our trained YOLOv8 model. The recorded data was processed using our trained model, showing excellent cone detection and segmentation performance Fig. . The system is also capable of running in real-time within ROS2, functioning efficiently and reliably.

Fig. 21. YOLOv8 model detecting and classifying cones during post-processing of recorded sensor data from Motor City racetrack.

LiDAR-Based Cone Detection

This experiment aimed to evaluate the LiDAR-based cone detection algorithm using pre-recorded data from a drive at the racetrack. The main stages: spatial filtering, ground filtering, clustering, and final cone detection were each visualized in RViz.

Experiment Procedure:

i. **Spatial Filtering (Distance-Based):**

The initial step involved filtering LiDAR points based solely on distance, retaining points within a 50-meter range from the vehicle. This was designed to focus on relevant points while ignoring distant, unnecessary data.

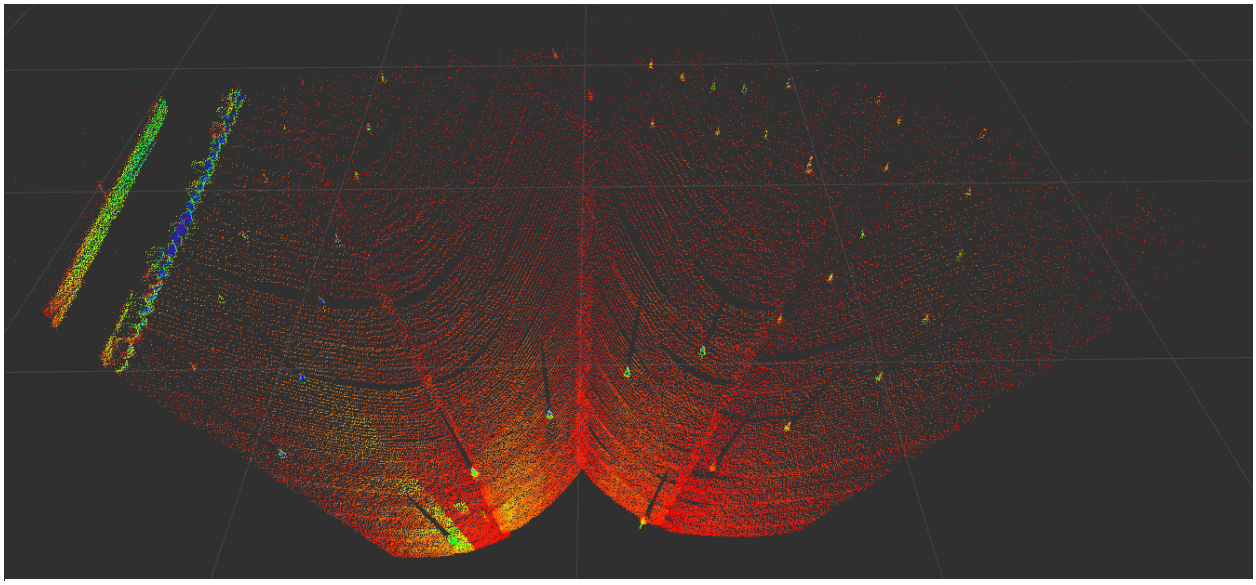


Fig. 22. Spatial filtering result, points filtered based on distance, retaining only those within 50 meters of the vehicle.

Ground Filtering:

The Patchwork++ algorithm was used to remove ground points. However, due to calibration issues, some ground points remained, slightly affecting the subsequent clustering step.

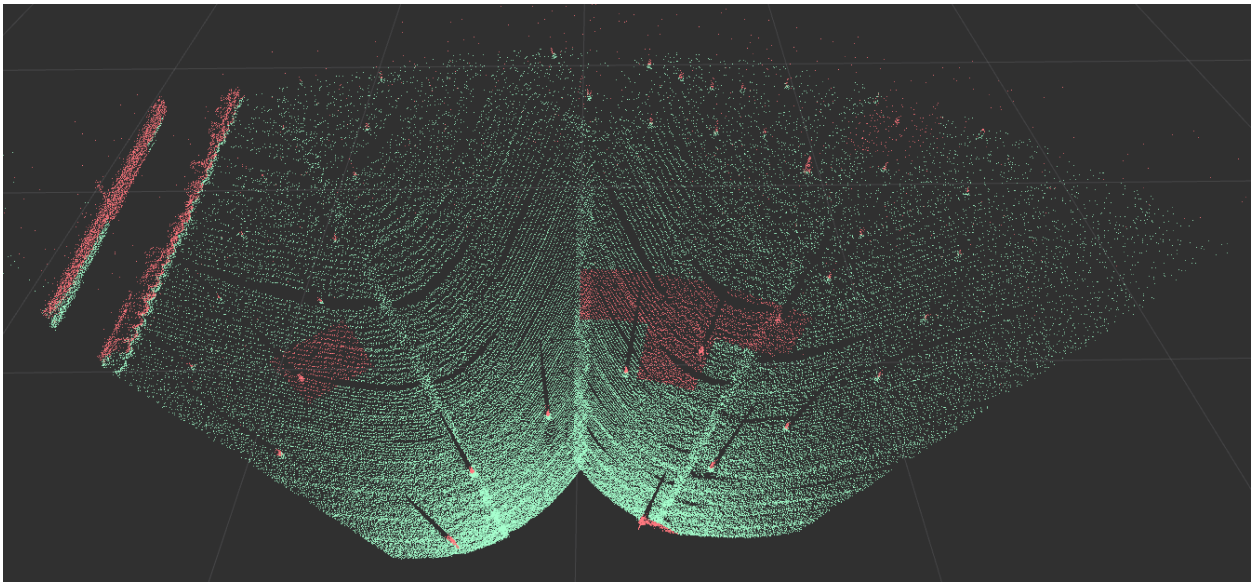


Fig. 23. Ground Filtering (Patchwork++). Split the Point Cloud to ground and nonground points.

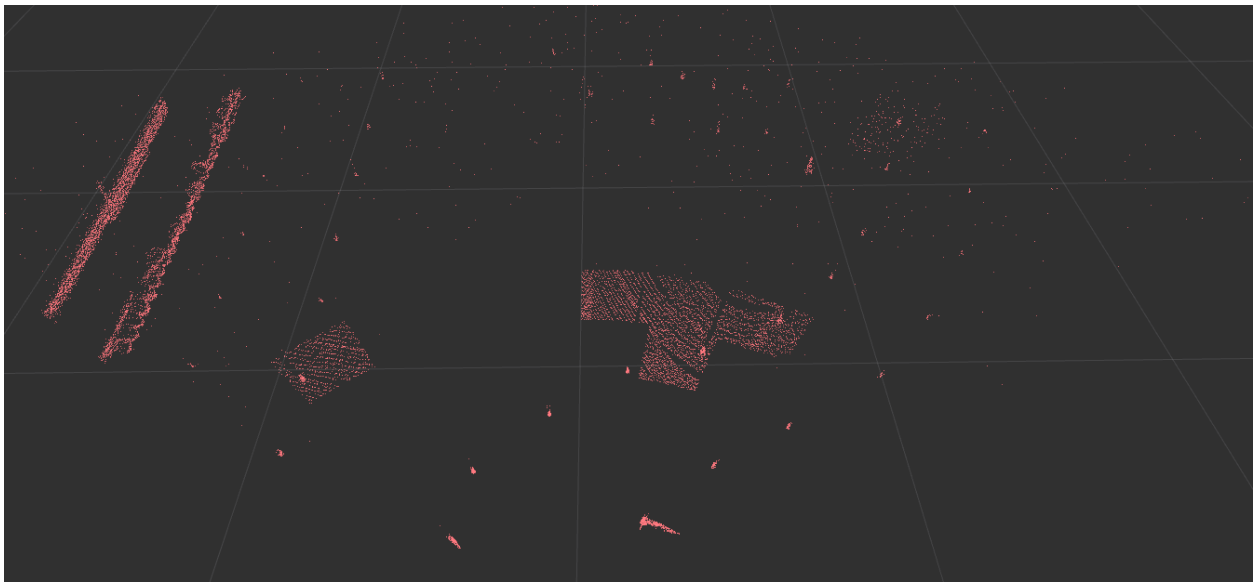


Fig. 18. Ground points removed, though some noise remains due to calibration issues.

ii. **Clustering:**

Euclidean clustering was applied to the filtered points to identify potential cones. The results were generally accurate, though the remaining ground points led to minor misclassifications.

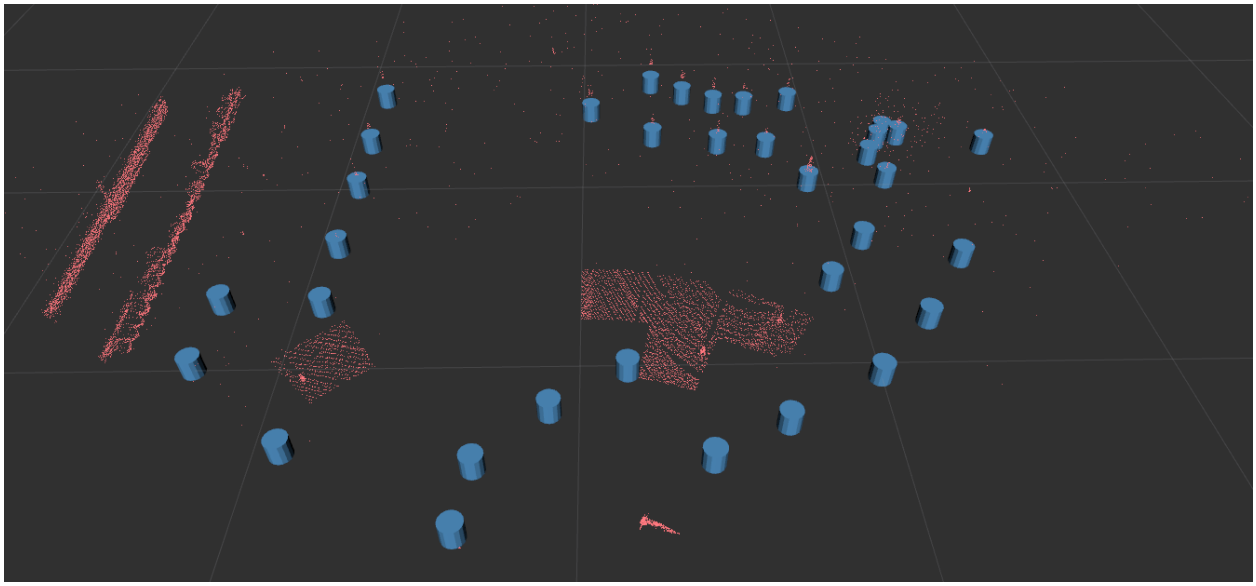


Fig. 19. Clustering algorithm applied, identifying potential cone clusters visualized as cylindrical markers representing detected cones.

Final Detection

The detected cones were visualized with markers indicating their positions relative to the track. While the overall detection results were promising, sensor calibration needs improvement, particularly in the ground filtering stage, to enhance detection accuracy.

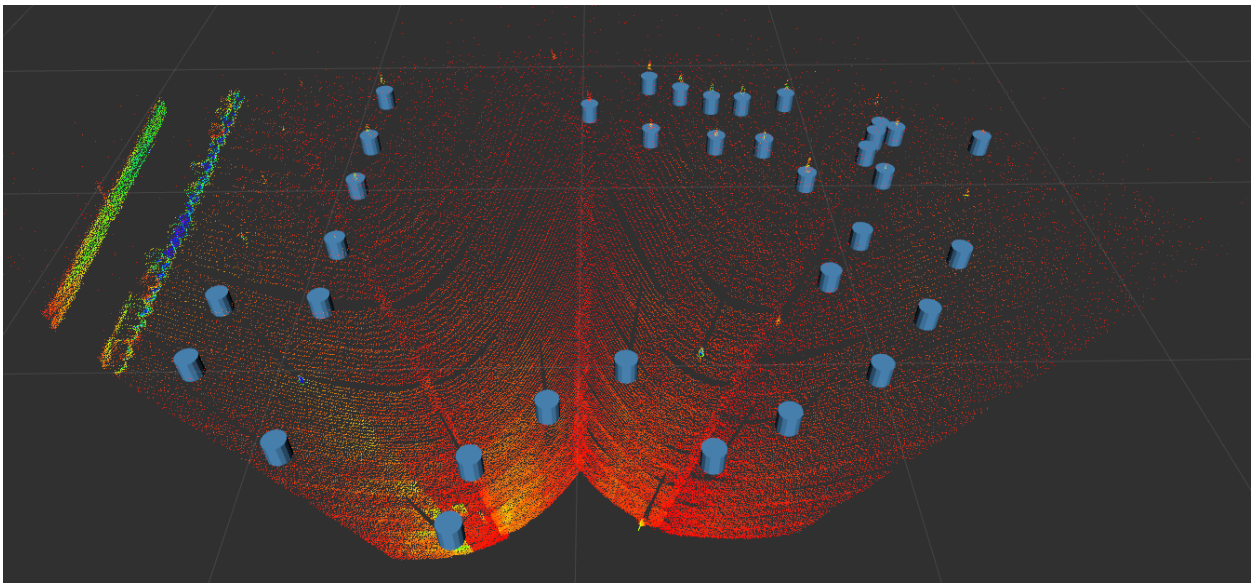


Fig. 20. Final output showing both the raw LiDAR data and the detected cone markers.

State Estimation Testing

In the state estimation section, the code includes a minimal simulation to validate its correctness. However, due to challenges in synchronizing recorded RPM and IMU data, we were unable to fully test the accuracy of our code with real-world data. We are currently working on acquiring synchronized recordings, but in manual simulations, the code performs well and shows promising results.

SLAM Research

In this project, the primary objective was to explore various SLAM approaches and identify the requirements for implementing SLAM in autonomous racing. Given the complexity and scope, the focus was on conducting research and defining the prerequisites rather than fully integrating SLAM into the current system. The team does not yet have the capability to perform SLAM in real-world conditions, but initial research and development efforts have begun, laying the foundation for continued work in future projects within the group.

Path Planning and Control Testing

Due to the current inability to generate a global map in real-world conditions, a simulation environment was developed in Python for easier development and rapid testing. This simulation replicates the mapping process and continuously updates the vehicle's state, allowing us to test and refine the path planning algorithm integrated with the Pure Pursuit controller and PID speed control.

While the primary simulation was developed in Python, a ROS-based version has also been implemented to ensure compatibility with the broader system. The simulation includes the entire process, from generating the path based on detected cones to applying the Pure Pursuit algorithm for steering and PID control for maintaining the desired speed.

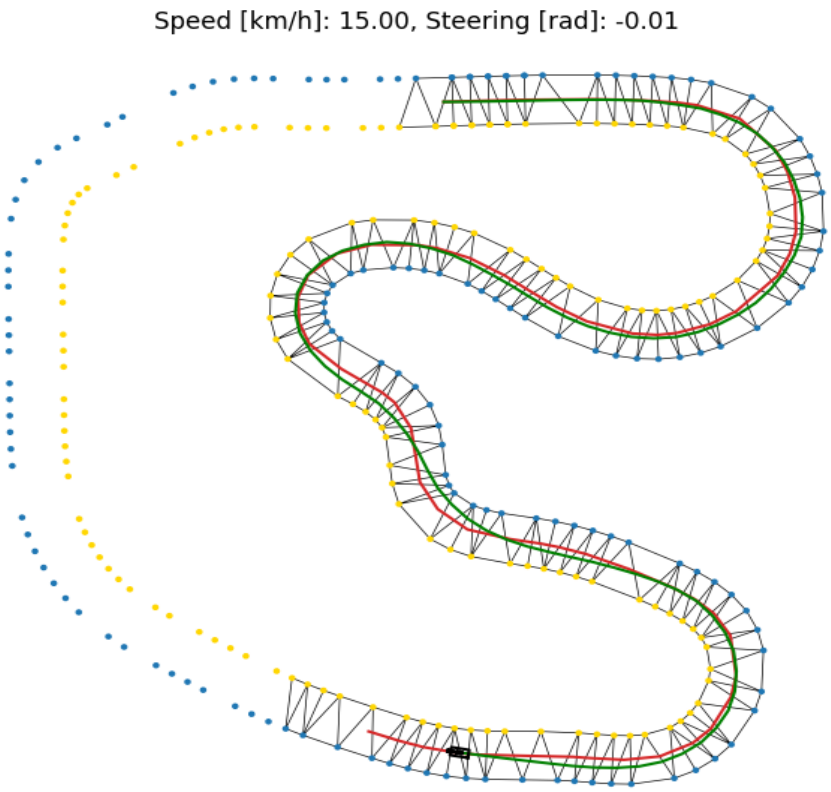


Fig. 21. Simulation showing Delaunay Triangulation planning with Pure Pursuit with PID control for path tracking.

Simulation result showing the vehicle's path tracking on a curved track, with visible steering and acceleration commands, demonstrating accurate following of the centerline using Pure Pursuit control and PID speed adjustment. An MPC-based controller version is also implemented, but it runs slower despite being functional.

Problems and Solutions

Data Collection and Sensor Synchronization: One of the primary challenges was the synchronization of data collected from different sensors (e.g., LiDAR, camera, IMU) both temporally and spatially. Misalignment in this data could result in significant errors in perception and navigation. To address this, we utilized ROS and its packages, which are specifically designed to handle such issues by managing coordinate transformations and ensuring real-time synchronization.

Limited Testing Days for Data Collection: Another challenge was the limited number of testing days available for data collection and system validation in real-world conditions. This restricted our ability to conduct comprehensive tests under realistic scenarios. The solution was to develop a simulation environment that allowed for controlled testing and system improvements, benefiting not just the project but the entire team.

Connecting Different Code Segments: Integrating different code segments was also a complex task. Each segment was developed with a different objective in mind, necessitating seamless data transfer between processes. By leveraging ROS2, we were able to create a structured and efficient framework that streamlined the integration of these diverse components.

Integration of Hardware Systems: Itegrating the hardware systems into the vehicle posed significant challenges. It required extensive efforts to ensure that all components worked harmoniously and were correctly installed. After considerable work on the vehicle, the system was finally able to function as intended.

Conclusions and Recommendations

The project successfully met its primary objectives by proposing a software architecture for an autonomous racing car, capable of competing in the dynamic events of the Formula Student Driverless competition. The integration of sensors like LiDAR, cameras, and IMU, offers a solid foundation for perception and localization. The implementation of advanced algorithms, including YOLOv8 for object detection and Delaunay Triangulation for path planning, resulted in a working first version, though it is still far from optimal and has room for significant improvement. The use

of ROS 2 as the framework enabled seamless communication between different modules, allowing the system to handle complex tasks such as real-time decision-making and vehicle control.

However, the project also highlighted several areas for improvement. Future work should focus on enhancing the integration of additional sensors, such as GPS, to provide better global reference points and reduce localization drift. Moreover, the current SLAM implementation is still in the research phase, and further development is needed to fully integrate SLAM into the system. Algorithm optimization, particularly in path planning and control, could also lead to smoother and more efficient performance in high-speed environments. Exploring more advanced control strategies like Model Predictive Control (MPC) and refining the simulation environment would further improve the system’s reliability and scalability.

Additionally, upgrading the computing unit to an NVIDIA Jetson AGX Orin is recommended, as it offers better performance for real-time processing in a competition setting. The cameras should also be upgraded to GMSL cameras, which are more robust and suitable for competitive use compared to the USB cameras that were used primarily for development purposes.

In summary, the project successfully proposed the foundation for an autonomous racing system that meets the competition's requirements while also providing a platform for ongoing development and future improvements. By continuing to refine sensor integration, algorithm efficiency, and overall system architecture, the vehicle’s performance in future competitions can be further enhanced.

References

[1] Formula Student Germany, "Rules 2024," 2024. [Online]. Available: https://www.formulastudent.de/fsg/rules/

[2] C. Stiller et al., "The Autonomous Racing Software Stack of the KIT19d," Karlsruhe Institute of Technology, 2020.

[3] A. Alvarez et al., "The Software Stack That Won the Formula Student Driverless Competition," presented at the Formula Student Germany, 2021.

[4] T. Friedrich et al., "Winning Through Simplicity: Autonomous Car Design for Formula Student," Elefant Racing e.V., University of Bayreuth, 2021.

[5] R. Heess et al., "AMZ Driverless: The Full Autonomous Racing System," ETH Zurich, 2020.

[6] Foxglove, "Understanding ROS Transforms," Foxglove Blog, [online]. Available: https://foxglove.dev/blog/understanding-ros-transforms.

[7] OpenCV, "Camera Calibration," OpenCV Documentation, [online]. Available: https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html.

[8] N. Vödisch, D. Dodel, and M. Schötz, “FSOCO: The Formula Student Objects in Context Dataset,” SAE Int. J. of CAV, vol. 5, no. 1, 2022.

[9] "Ultralytics ROS Integration," GitHub Repository. [Online]. Available: https://github.com/Alpaca-zip/ultralytics_ros/tree/humble-devel.

[10] S. Lee, H. Lim, and H. Myung, "Patchwork++: Fast and Robust Ground Segmentation Solving Partial Under-Segmentation Using 3D Point Cloud," IEEE Transactions on Automation Science and Engineering, 2023.

[11] L. A. N. C. Lopes, "A SLAM Method for the Formula Student Driverless Competition," M.Sc. thesis, Instituto Superior Técnico, Universidade de Lisboa, Oct. 2021.

[12] "A comparison of different approaches to solve the SLAM problem on a Formula Student Driverless race car," 2021.

[13] S. Chary, "A Brief Introduction to GraphSLAM," Medium, Aug. 2020. [Online]. Available: https://shivachandrachary.medium.com/a-brief-introduction-to-graphslam-4204b4fce2f0.

[14] MathWorks, "Path Planning for Formula Student Driverless Cars using Delaunay Triangulation," MathWorks Student Lounge Blog, Oct. 3, 2022. [Online]. Available: https://blogs.mathworks.com/student-lounge/2022/10/03/path-planning-for-formula-student-driverless-cars-using-delaunay-triangulation/

[15] "FT-FSD Path Planning Project," GitHub Repository. [Online]. Available: https://github.com/papalotis/ft-fsd-path-planning/tree/main.

[16] A. Slomoi, Path Planning and Control in an Autonomous Formula Student Vehicle, supervised by Professor T. Drummond, Dept. of Electrical and Computer Systems Engineering, 2018.

Appendices

The complete code for this project is available on GitHub:

GitHub Repository: System Architecture for Formula Student Driverless

המלצת ציון לדו''ח מסכם

מספר הפרויקט: P-2024-057, שם הפרויקט: System architecture in formula student car

שם הסטודנט: מיכאל לייב ת.ז.: 319095832

קריטריון	1 - חלש	2 - בינוני	3 - טוב	4 - ט״מ	5 -מצוין
הגדרת המטרה - האם מטרת הפרויקט ברורה? המטרה צריכה לכלול גם את התרומה הצפויה מהשלמת הפרויקט.					
ארכיטקטורת הפתרון - האם הובהר מדוע הארכיטקטורה המוצעת לפתרון ההנדסי מתאימה לפתרון הבעיה? האם הייתה התייחסות לאלטרנטיבות?					
הצגת התוצאות - האם הוסברו המשמעויות של התוצאות, או המסקנות הנובעות מהן, האם רק הייתה הצגה עובדתית של תוצאות?					
מבנה – מצגת - האם יש מבנה הגיוני להצגת העבודה? מבוא, רקע ו-SoTA, מטרות, שיטה, פתרון, תוצאות, מסקנות.					
רמת קושי - כיצד הנך מעריך את הפרויקט בהשוואה לפרויקטים אחרים השנה או בשנים קודמות?					
איכות כתיבה / הצגה - נא להתייחס לאיכות הצגת העבודה: תחביר, מבנה וקשר הגיוני בין החלקים במהלך הצגת העבודה					
שורה תחתונה - האם מטרות הפרויקט הושגו? יש להתייחס למטרות כפי שהוגדרו בתחילה.					
הבנה - האם נראה שהסטודנט מבין את העבודה שנעשתה? האם עומק הדיון מספק?					
השקעה - האם ניכר שהסטודנט השקיע בעבודה בפרויקט ו/או בהכנת ההצגה של העבודה? האם נראה שהעבודה נעשתה בחופזה, כדי לצאת ידי חובה?					
דירוג - כיצד הנך מעריך את איכות הפרויקט ביחס לכלל הפרויקטים שראית?					
המאפיינים העיקריים בפרויקט - יש לסמן את כמות המאפיינים שהושגו (ולו באופן חלקי). מאפיין אחד בעמודה 1 וכן הלאה... המאפיינים: * מימוש/יצור מערכת, * אנליזה מתמטית, * תכנון אלגוריתמי, * אינטגרציה עם מערכת קיימת, * חקר ביצועים. בקביעת הציון: העדר מאפיינים מזכה בציון 1, מאפיין אחד בלבד מזכה בציון 3, ושני מאפיינים ומעלה מזכים בציון 5.					

אם יש כוונה לפרסם/ יפורסם מאמר, שם כתב העת ומועד משוער להגשה:

ציין אם יש כוונה לשקול המלצה כפרויקט מצטיין: