VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in order to be awarded the degree of
Master of Science in Engineering Technology:
Electronics-ICT - networks

DESIGN AND IMPLEMENTATION OF AN AUTONOMOUS RACING SYSTEM FOR A FORMULA STUDENT RACING CAR
Efficient path-planning through LiDAR-based SLAM

Valentin Quevy

2022-2023

Promoters: Prof. Dr. Ir. Jan Lemeire & Prof. Dr. Ir. Adrian Munteanu

ENGINEERING SCIENCES

Acknowledgements

Without the awesome resources and advice that my promotor Jan and copromotor Adrian provided, I could not have completed my master's thesis. It was crucial for me to gain knowledge from other perspectives in order to widen my education and grow as a person.

VUB-Racing was also very involved; they explained the issue and gave me the opportunity to develop their driverless system. My professional and interpersonal abilities have significantly improved as a result of the numerous meetings and events.

My last words belong to my family and friends, for whom I am truly thankful of the support they provided. Thanks to them, I was able to easily overcome the great challenges.

Valentin Quevy

1040 Etterbeek, June 12, 2023

Abstract

English

This master's thesis aims to provide a comprehensive account of the design and implementation of an autonomous system specifically tailored for a Formula Student (FS) racing car. The primary focus of this research involves an in-depth exploration of various critical aspects, including sensor selection, test platform development and design of the software-stack which includes the establishment of a robust perception pipeline, efficient path-planning strategies, and precise control mechanisms.

Comparative analyses within each of these areas will assess the effectiveness and efficiency of different approaches. Through these analyses, valuable insights will be gained into the strengths and weaknesses of each approach, facilitating informed decisions when selecting the optimal sensors for the system. The sensors under consideration include LiDAR, camera, radar, and GPS.

In addition, the thesis will present a comprehensive test and simulation platform, along with the methodology employed for evaluating the performance of the autonomous racing system. Gazebo will provide a realistic environment for testing and validating the system, ensuring accurate and reliable results.

Furthermore, we will address the design and integration of various software-stack components. These components play a crucial role in the system's overall functionality and include modules for perception, path-planning, and control. We will delve into the details of these software-stack components, exploring their design principles and their interplay in achieving seamless autonomy.

Additionally, a thorough performance analysis will evaluate the system's overall capabilities and limitations. This analysis will provide a comprehensive understanding of the system's performance in terms of accuracy, speed, and robustness.

To achieve the objectives, this thesis employs LiDAR-based Simultaneous Localization and Mapping (SLAM) techniques utilizing the Cartographer package. These techniques enable efficient path-planning through triangulation. By incorporating a LiDAR sensor into the perception pipeline, the system can generate real-time maps of the environment. These maps serve as the foundation for the effective path-planning strategy, enabling the racing car to navigate the track efficiently. Computed waypoints and a pure-pursuit control system are utilized to guide the car along the optimal racing line.

The study and its results are expected to significantly advance the knowledge and capabilities of autonomous systems for the VUB Formula Student Driverless racing team. The insights gained from this research will inform and guide future developments, empowering the team to participate in the future Formula Student Driverless (FSD) competitions.

Dutch

Deze masterthesis heeft als doel een uitgebreid verslag te bieden van het ontwerp en de implementatie van een autonoom systeem dat specifiek is afgestemd op een Formula Student-raceauto. Het belangrijkste doel van dit onderzoek omvat een diepgaande verkenning van verschillende kritieke aspecten, waaronder sensorenselectie, ontwikkeling van een testplatform en het ontwerp van de softwarestack, waarbij de nadruk ligt op het opzetten van een robuuste perceptiepijplijn, efficiënte padplanningsstrategieën en nauwkeurige besturingsmechanismen.

Vergelijkende analyses binnen elk van deze gebieden zullen de effectiviteit en efficiëntie van verschillende benaderingen beoordelen. Door middel van deze analyses worden waardevolle inzichten verkregen in de sterke en zwakke punten van elke benadering, wat bijdraagt aan geïnformeerde besluitvorming bij het selecteren van de optimale sensoren voor het systeem. De overwogen sensoren zijn onder andere LiDAR, camera, radar en GPS.

Daarnaast zal de thesis een uitgebreid test- en simulatieplatform presenteren, samen met de gebruikte methodologie voor het evalueren van de prestaties van het autonome racesysteem. Gazebo zal een realistische simulatie omgeving bieden voor het testen en valideren van het systeem, wat zorgt voor nauwkeurige en betrouwbare resultaten.

Verder zullen we ons richten op het ontwerp en de integratie van verschillende componenten van de softwarestack. Deze componenten spelen een cruciale rol in de algehele functionaliteit van het systeem en omvatten modules voor perceptie, padplanning en besturing. We zullen ingaan op de details van deze softwarestack-componenten, waarbij we hun ontwerpprincipes en hun onderlinge samenwerking onderzoeken om naadloze autonomie te bereiken.

Daarnaast zal er een grondige prestatieanalyse worden uitgevoerd om de algehele mogelijkheden en beperkingen van het systeem te evalueren. Deze analyse zal een grondig begrip verschaffen van de prestaties van het systeem op het gebied van nauwkeurigheid, snelheid en robuustheid.

Om de doelstellingen te bereiken, maakt deze thesis gebruik van LiDAR-gebaseerde Simultane Localisatie and Mapping (SLAM) technieken met behulp van de software pakket Cartographer. Deze technieken maken efficiënte padplanning mogelijk door middel van triangulatie. Door een LiDAR-sensor in de perceptiepijplijn op te nemen, kan het systeem realtime kaarten van de omgeving genereren. Deze kaarten vormen de basis voor een effectieve padplanningsstrategie, waardoor de raceauto efficiënt het parcours kan afleggen. Berekende waypoints en een pure-pursuit besturingssysteem worden gebruikt om de auto langs de optimale racelijn te begeleiden.

Dit onderzoek en de resultaten ervan zullen naar verwachting de kennis en mogelijkheden van autonome systemen voor het VUB Formula Student Driverless-raceteam aanzienlijk bevorderen. De inzichten die we verkrijgen uit dit onderzoek zullen toekomstige ontwikkelingen informeren en leiden, waardoor het team zal kunnen deelnemen aan de toekomstige Formula Student Driverless (FSD) racewedstrijden.

French

Ce mémoire de master vise à fournir un compte rendu complet de la conception et de la mise en œuvre d'un système autonome spécialement conçu pour une voiture de course Formula Student. La recherche se concentre principalement sur l'exploration approfondie de divers aspects critiques, tels que la sélection des capteurs, le développement de la plateforme de test et la conception de la pile logicielle. Cela comprend la mise en place d'un pipeline de perception robuste, de stratégies de planification de trajectoire efficaces et de mécanismes de contrôle précis.

Des analyses comparatives seront réalisées dans chacun de ces domaines pour évaluer l'efficacité et l'efficience des différentes approches. Ces analyses fourniront des informations précieuses sur les avantages et les limites de chaque approche, ce qui facilitera la prise de décisions éclairées lors du choix des capteurs optimaux pour le système. Les capteurs pris en considération comprennent le LiDAR, la caméra, le radar et le GPS.

De plus, le mémoire présentera une plateforme complète de test et de simulation, ainsi que la méthodologie utilisée pour évaluer les performances du système de course autonome. Gazebo fournira un environnement réaliste pour tester et valider le système, garantissant ainsi des résultats précis et fiables.

Par ailleurs, nous aborderons la conception et l'intégration des différents composants de la pile logicielle. Ces composants jouent un rôle crucial dans la fonctionnalité globale du système et comprennent des modules de perception, de planification de trajectoire et de contrôle. Nous examinerons en détail ces composants de la pile logicielle, en explorant leurs principes de conception et leur interrelation pour assurer une autonomie fluide.

En outre, une analyse approfondie des performances évaluera les capacités globales du système ainsi que ses limites. Cette analyse permettra de mieux comprendre les performances du système en termes d'exactitude, de vitesse et de robustesse.

Pour atteindre ces objectifs, ce mémoire utilise des techniques de localisation et de cartographie simultanées (SLAM) basées sur le LiDAR à l'aide du pacquet logiciel Cartographer. Ces techniques permettent une planification de trajectoire efficace par triangulation. En intégrant un capteur LiDAR dans le pipeline de perception, le système est capable de générer des cartes en temps réel de l'environnement, qui serviront de base à une stratégie de planification de trajectoire efficace permettant à la voiture de course de naviguer efficacement sur la piste. Des waypoints calculés et un système de contrôle de poursuite pure sont utilisés pour guider la voiture le long de la trajectoire de course optimale.

Cette étude et ses résultats sont censés contribuer de manière significative à l'avancement des connaissances et des capacités des systèmes autonomes pour l'équipe de course VUB Formula Student Driverless. Les enseignements tirés de cette recherche orienteront les développements futurs, permettant ainsi à l'équipe de participer au futures compétitions Formula Student Driverless (FSD).

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ADAS	Advanced Driver Assistance Systems
AMCL	Adaptive Monte-Carlo Localizer
AV	Autonomous Vehicle
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
EKF	Extended Kalman Filter
FS	Formula Student
FSAE	Formula Society of Automotive Engineering
FSD	Formula Student Driverless
FSM	Finite State Machine
FSOCO	Formula Student Objects in Context dataset
GPS	Global Positioning System
HSV	Hue Saturation Value
IMU	Inertial Measurement Unit
IR	Infrared light
LiDAR	Light Detection and Ranging
SBC	Single Board Computer
SLAM	Simultaneous Localization and Mapping
MP	Motion Planning
MPC	Model Predictive Control
NIR	Near-Infrared Light
PID	Proportional Integral Derivative
ROS	Robot Operating System
ToF	Time of Flight
UI	User Interface
VUBR	Vrije Universiteit Brussel Racing team
YOLO	You Only Look Once algorithm

Chapter 1

Introduction

In the ever-evolving world of autonomous vehicles, I have always been captivated by the cutting-edge technologies that enable these vehicles to navigate and interact with their surroundings. The ability of a camera to capture the world not only in 2D images but also in 3D has truly fascinated me. This fascination led me to discover the realm of Formula Student racing, where the challenges faced in terms of perception and navigation are particularly intriguing. Recognizing this as the perfect opportunity to merge my passion for autonomous vehicles with real-world applications, I eagerly joined the VUB racing team to contribute my skills and knowledge.

The primary goal of my Master's thesis is to explore state-of-the-art techniques in LiDAR perception and investigate their applicability to the specific requirements of Formula Student race cars. One of the most crucial challenges lies in accurately detecting, separating, and precisely localizing the cones on the racing track. This complex task requires an innovative solution, and I am determined to find the optimal approach given our available resources.

In pursuit of this goal, I will dive into various algorithms for object recognition and path planning. The focus will be on identifying the most suitable algorithms for FS race cars. This research endeavor will culminate in a comprehensive study of the perception pipeline and path planning algorithms, with a strong emphasis on innovation.

This thesis aims to achieve two main objectives. Firstly, I aim to make a significant contribution to the research and development efforts of VUB-Racing, enhancing their existing knowledge and capabilities. Secondly, I will propose an effective path planning strategy specifically tailored for the discovery lap, the initial phase of the racing competition.

To accomplish these objectives, the thesis follows a structured approach. The structure encompasses several key sections, including an in-depth review of relevant literature on FS racing, SLAM, and LiDAR technology. The materials and methods section will provide detailed insights into the hardware and software choices made, as well as the methodology employed. This includes the utilization of SLAM, Delaunay triangulation, waypoint generation, and the implementation of a pure-pursuit controller.

Throughout the research, a series of experiments will be conducted to evaluate the proposed methods. The results obtained from these experiments will be rigorously analyzed and discussed, providing valuable insights into the performance and effectiveness of the chosen approaches. Furthermore, the discussion section will offer a comprehensive interpretation of the findings.

In conclusion, this Master's thesis embarks on an exciting journey to design and implement an autonomous racing system for a Formula Student race car. With a specific focus on efficient path planning through LiDAR-based SLAM, we aim to push the boundaries of perception and navigation. Through our contributions to VUB-Racing and advancements in autonomous racing, we strive to make a big impact in this thrilling field.

The autonomous system for the Formula Student Driverless (FSD) competition must be able to race through the dynamic events in various track layouts and possible weather conditions, including dry, damp, and wet. To showcase engineering skills, the competition challenges teams to perform well in both static and dynamic events. The static events include business plan development, cost and manufacturing analysis, and engineering design.

2.1 Dynamic events

The dynamic events test the vehicle's performance and reliability under different angles in circuits marked with cones. The cones guide the vehicle along the track, with small blue cones marking the left border and small yellow cones marking the right border of the track. Small orange cones designate the entry and exit lanes, and big orange cones are placed before and after the start, finish, and timekeeping lines.

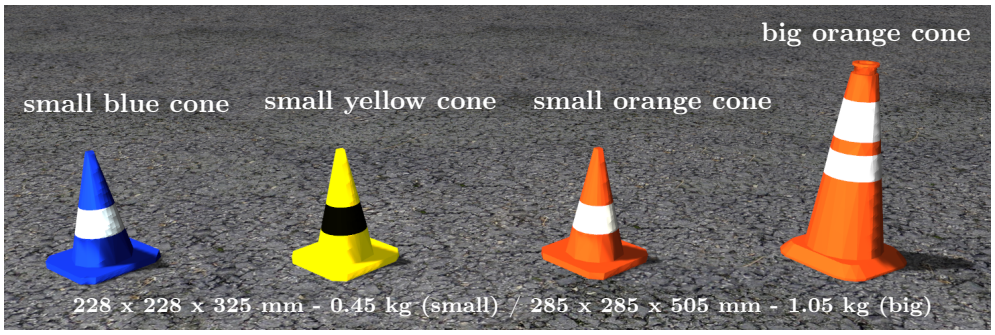

Figure 2.1: FSD cones specifications

The driverless cars are tested in various events during the FSD competitions to evaluate their performance and reliability. The Acceleration event evaluates the car's acceleration capability. The Skidpad event measures the car's lateral grip and cornering performance. The Autocross and Trackdrive events evaluate the car's ability to navigate through an unknown track and complete multiple laps of a fixed track, respectively.

2.1.1 Acceleration event

The acceleration track is **a straight line** with a length of 75 m from starting line to finish line. The track is at least 3 m wide. Cones are placed along the track at intervals of about 5 m. Cone locations are not marked on the pavement.

Each team gets two attempts to complete the autonomous run. The vehicle starts 0.30 meters behind the starting line and accelerates from a standing position. A go-signal from RES signals the beginning of the run, and timing starts once the vehicle crosses the starting line and stops once it crosses the finish line. After crossing the finish line, the vehicle must come to a complete stop within 75 meters inside the marked exit lane to enter the finish-state.

2.1.2 Skidpad event

The skidpad track has **a figure-eight pattern** with two pairs of concentric circles. The inner circles have a diameter of 15.25 m, and the outer circles have a diameter of 21.25 m. Sixteen cones are placed around the inside of each inner circle, and thirteen cones are positioned around the outside of each outer circle. The driving path is the 3 m wide path between the inner and outer circles, which is marked by a line inside the outer circle and outside the inner circle. Vehicles enter and exit the track through gates on a 3 m wide path that is tangent to the circles where they meet, and the centers of the circles are 18.25 m apart. On fig. 2.3 we see an example of a possible skidpad track.

During the driverless procedure for racing the skidpad track, each team gets two chances to compete in autonomous mode. The vehicle starts 15 m from the timekeeping line, enters perpendicular to the figure eight, and takes one full lap on the right circle to establish the turn. The second lap on the right circle is timed, followed by the third and fourth laps on the left circle. After finishing the fourth lap, the vehicle exits the track and must come to a full stop within 25 m behind the timekeeping line in the marked exit lane.

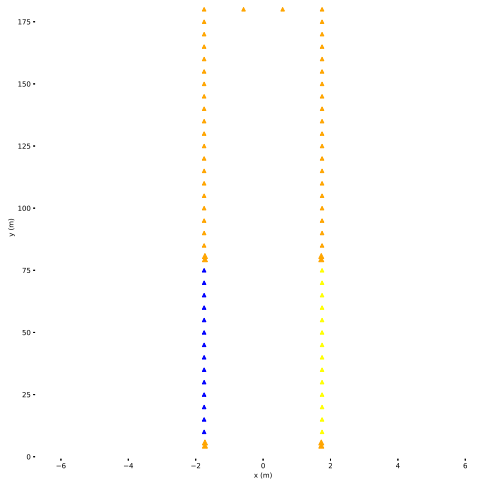

Figure 2.2: Acceleration track

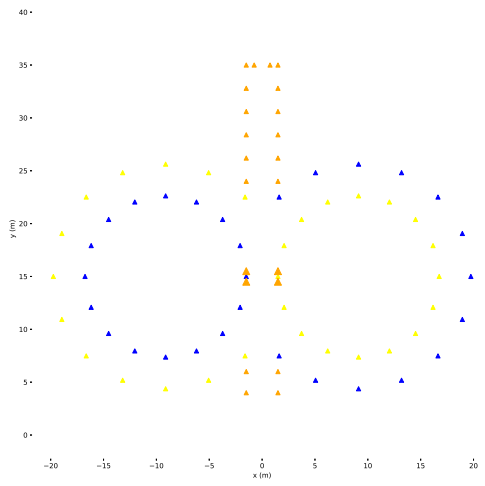

Figure 2.3: Skidpad track

2.1.3 Autocross & Trackdrive event

The Driverless autocross event takes place on the same track as driverless trackdrive event.

The track layout is **a closed loop circuit** that includes straights no longer than 80m, constant turns up to 50m in diameter, hairpin turns with a minimum of 9m outside diameter, and miscellaneous features such as chicanes, multiple turns, and decreasing radius turns. The minimum track width is 3 m, and the lap length is typically between 200 m and 500 m. On fig. 2.4 we can see examples of such a track layout.

In Autocross, the driverless system must navigate the car through the unknown track as quickly as possible, while in Trackdrive, the car must complete ten laps of the same track.

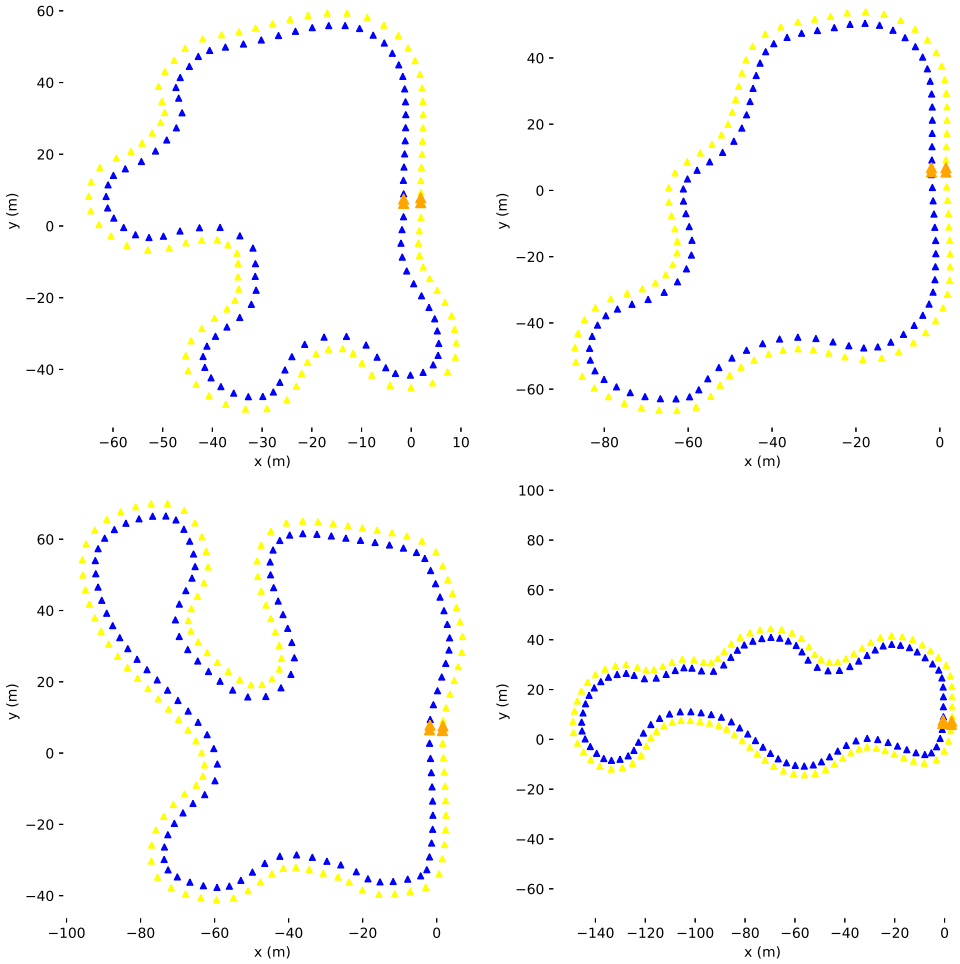

Figure 2.4: Examples of trackdrive tracks layouts

2.2 Identification of the autonomous system capabilities

Based on the problem statement stated in section 2.1, the autonomous racing system must be able to perceive the track through various sensors, analyze the information gathered, make quick and accurate decisions, and finally control the car in order to successfully complete those events.

Below is a list of the essential macro-abilities that are required for an autonomous system to perform effectively in the competitions.

- Perception
  - Detect and recognize cones of different colors and sizes
  - Identify the boundaries of the track
  - Identify track irregularities
- Path-planning
  - Plan a path that avoids obstacles and follows the track
  - Compute an optimal race path that adapts to the choosen speed and acceleration profile
- Control
  - Implement algorithms to steer the vehicle
  - Implement control algorithms to throttle and brake
  - Implement stability and smoothness of the vehicle's motion
  - Implement fail-safe mechanisms to ensure safe operation of the vehicle

3.1 Driverless package main building blocks

In an autonomous race car, there is no driver. Instead, we rely on advanced hardware and logic to replace the driver's senses and brain. This enables the car to perceive its surroundings, make decisions, and navigate the race tracks.

To replicate the driver's senses, we can use various sensors like cameras, LiDAR, and radar. These sensors provide valuable information about the car's environment, allowing it to "see" and understand its surroundings. Each sensor has its own advantages and limitations, which we carefully analyze to ensure optimal performance.

The brain of the driver is emulated by a powerful mini computer. This computer processes the sensor data in real-time and computes the critical decisions based on the written algorithms, such as choosing the best racing path. For this the mini computer must be capable of handling complex algorithms quickly and efficiently.

Designing the autonomous race car involves considering the car as a whole. It consists of different packages, including aerodynamics, mechanics, powertrain, electronics, and the driverless package. The driverless package is particularly important, as it houses the autonomous technology. For this reason, we strive to make this package small, energy-efficient, and cost-effective while still meeting our performance goals.

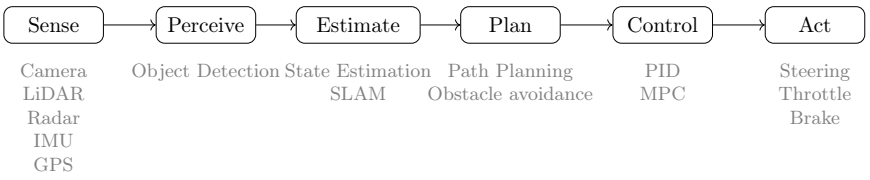

Figure 3.1: State-of-the-Art logic behind an FSD race car

3.2 Perception

The perception system of a FSD car uses a range of sensors and algorithms to collect data about the race track and interpret it for mapping the cones accurately in space in real-time. The algorithms employed in the perception system must possess the capability to detect and classify the four different cone types found during the competitions. This can be achieved through feature extraction techniques that take into account the conical shape and distinguishable yellow and blue colors of the cones. Additionally, the relative position of the cones with respect to the car's pose can also aid in their identification and classification.

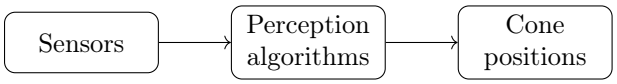

Figure 3.2: Perception flow

3.2.1 Sensor selection

When choosing sensors for a perception system, it's important to carefully evaluate each sensor as a possible sensing device. Each sensor has its own strengths and limitations, and understanding their capabilities is crucial in making a good choice.

Different sensors, like cameras, LiDAR, and radar, provide different types of information about the environment. Cameras capture detailed visual data, showing colors and textures. LiDAR sensors give accurate 3-D information by using laser beams to form pointclouds. Radar sensors detect and track objects based on their motion and reflection using radio waves. Figure 3.3 gives a visualization of what the different sensor modalities will perceive, including 2-D detection with a camera, 3-D detection with LiDAR, and range-based detection with RADAR.

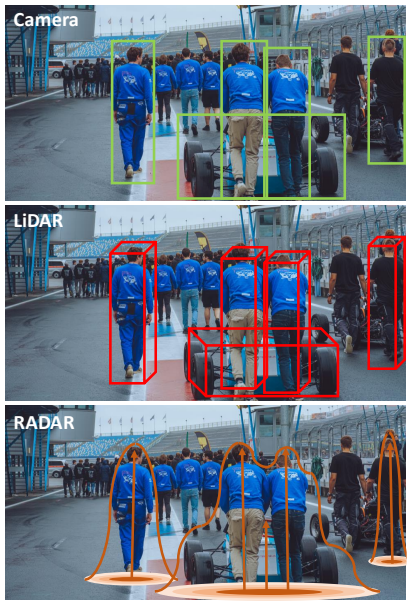

Figure 3.3: Visualization of camera, LiDAR and RADAR measurements in a typical autonomous driving use- case, FSN 2021.

By considering each sensor separately, we can assess how well they work for specific perception tasks. For example, cameras are good for detecting and recognizing objects, while LiDAR is great for getting precise spatial information. Radars are useful for object detection in bad

weather and for moving objects. On table 3.1 the pros and cons of each sensor are listed and on fig. 3.5 a comparison is made based on several factors.

However, the real power of perception systems often comes from combining sensors. When we fuse data from multiple sensors, we can take advantage of their strengths and compensate for their weaknesses. The fusion may typically first require to synchronize the sensor data in time, preprocess and register spatially before the final fusion. The fusion can take place at various stages of the perception pipeline, to finally combine data from multiple sensors. Early fusion occurs at the beginning, combining raw data for high accuracy but potentially reduced robustness. Late fusion applies sensor-specific processing pipelines, while intermediate fusion happens at an intermediate stage, striking a balance between complexity and accuracy. However, sensor fusion comes at the cost of increased complexity in terms of system architecture and additional processing requirements, as well as from a financial perspective. On fig. 3.4, a complete estimation driverless package is proposed on their race car of 2023.

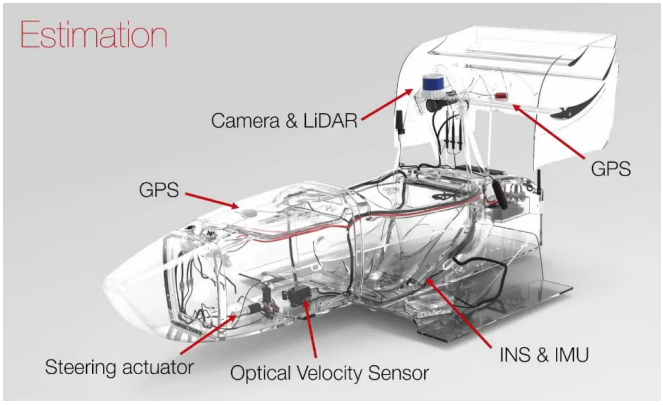

Figure 3.4: AMZ Castor 2023 estimation driverless package [1]

For example, combining camera and LiDAR data can improve object detection accuracy and robustness by using the visual details from cameras and the depth information from LiDAR. This fusion helps the perception system accurately detect and locate objects in a different situations. Similarly, integrating radar data with other sensors provides a more complete understanding of the environment, especially when visibility is poor or objects are partially hidden, contributing to the robustnes of the perception.

The choice of sensors and the level of sensor fusion is influenced by various factors, such as accuracy, computational demands, cost, and system constraints. Since this is our first year of research and development as a team of five students for the VUB FS team, we opted to pursue two non-fused pipelines: one utilizing a camera and the other incorporating a LiDAR camera. This decision was made to manage the initial project complexity effectively and work with affordable materials to control costs. Furthermore, we utilized simulations to establish a solid groundwork for future advancements and enhancements.

Sensor	Pros	Cons
Camera		
	+ Rich visual information	– Limited range
	+ Color perception	– Depth perception challenges
	+ Object recognition	– Vulnerability to occlusion
LiDAR		
	+ Accurate distance measurements	– Limited color and texture information
	+ 3D perception	– Cost and complexity
	+ Robustness to lighting conditions	
Radar		
	+ Long-range detection	– Lower resolution
	+ Weather resistance	– Limited object classification
	+ Speed measurement	

Table 3.1: Pros and cons of Camera, LiDAR, and Radar sensors

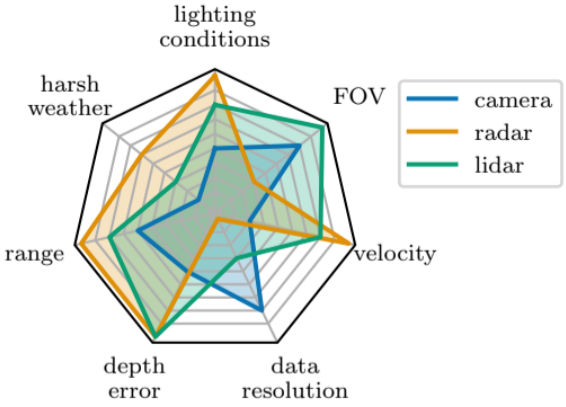

Figure 3.5: Sensor modalities quantitatively compared on a scale from bad to excellent, [6]

3.2.1.1 Camera

A camera can be a very good choice for a FS team, despite some challenges. Depending solely on a single camera for object detection may pose risks due to various factors that affect camera images. For instance, noise can interfere with image quality during nighttime operations, while hazy, foggy, or rainy conditions can lead to a loss of contrast. Additionally, the presence of glare when facing bright light sources and the accumulation of dirt on camera lenses over time can further impact performance. Cone detection could simply be achieved by converting the video stream into Hue Saturation-Value (HSV) format and applying bounding boxes based on the color code of the cones. However, this method would require continuous adjustment as the brightness levels change, affecting the reliability of the detection process. A more sophisticated approach would be required to achieve more precise cone detection. One potential solution is to use a Convolutional Neural Network (CNN) such as YOLO, trained with labeled data like the FSOCO dataset.

Furthermore, measuring the distance to objects using cameras can be ambiguous due to the inherent loss of depth in the image formation process but can be achieved with a more expensive dual-lense camera. However, despite these limitations, cameras offer many advantages. They provide rich visual information, enabling detailed object recognition, color perception, and texture analysis. Cameras also offer a wide field of view, enhancing situational awareness on the race track. Furthermore, cameras are relatively affordable and widely available, making them a cost-effective solution for FS teams with limited resources. For those reasons a camera-based approach must be considered.

3.2.1.2 LiDAR

Choosing LiDAR as a sensor for a FSD car can be an excellent choice. LiDAR provides highly **accurate distance measurements**, enabling precise navigation on the track. It can accurately determine the distance between the car and track boundaries, ensuring safe maneuvering around cones and obstacles.

The ability of LiDAR to generate detailed **3-D spatial information** is particularly advantageous for object detection and obstacle avoidance. By creating a comprehensive point cloud representation of the environment, LiDAR can precisely identify and track cones and other objects on the track. This allows the FSD car to make informed decisions and execute maneuvers with precision.

LiDAR's reliable **performance in various lighting conditions** is another valuable attribute. Whether it's a bright sunny day or a low-light evening race, LiDAR can consistently detect and measure distances without significant variations. LiDAR sensors utilize different wavelengths of light, including near-infrared or even infrared, which are less affected by ambient lighting variations compared to visible light.

However, there are considerations to address. LiDAR's **large volume of data** output requires robust processing capabilities. Advanced algorithms and powerful computing hardware are needed to efficiently handle and analyze the vast amount of data produced by the LiDAR sensor in real-time.

When it comes to implementation costs, LiDAR can be a **significant investment** for an FSD team. However, its benefits in terms of accurate perception and enhanced safety on the track can outweigh the initial financial outlay.

Regarding the energy consumption, regular cameras generally rely on passive light detection, which requires lower power consumption. The laser emission, ranging, and signal processing components of LiDAR sensors contribute to their **higher energy demands**. However, it's important to note that power consumption can vary depending on the specific models and designs of both LiDAR sensors and regular cameras.

LiDAR sensors are **sensitive to vibrations** especially scanning-state type ones, which are common in FSD cars due to their wide horizontal FOV. To mitigate this, proper mounting and damping techniques must be employed to minimize the impact of vibrations on the sensor's performance. This ensures reliable and accurate measurements even under challenging racing conditions.

Lastly, while LiDAR performs well in most weather conditions, it can be **affected by adverse weather such as heavy rain or dense fog**. In such cases, supplementary sensors or fusion with other perception technologies may be necessary to maintain reliable perception capabilities for the FSD car.

3.2.1.3 LiDAR technology

LiDAR technology is a sensing method used to measure distances and create detailed 3D maps of the environment. Integrating LiDAR offers enhanced perception capabilities and enables safer navigation and advanced autonomous functionalities. This technology involves emitting laser pulses, measuring their time of flight, calculating distances, creating point clouds, processing data, and utilizing it for object recognition and mapping.

1. **Laser pulse emission**
   LiDAR systems consist of laser diodes that emit short pulses of laser light. These pulses are typically in the form of infrared light and have a known wavelength and energy level. The laser pulses are emitted in a controlled manner, either by using a rotating mirror (scanning type) to scan the laser beam or by employing multiple fixed lasers (solid-state type).

2. **Light Propagation**
   Once emitted, the laser pulses propagate through the surrounding environment. They travel at the speed of light until they encounter an object or surface. When the laser pulse hits an object, it gets reflected back towards the LiDAR sensor.

3. **Time-of-Flight (TOF) Measurement**
   LiDAR sensors measure the time it takes for the laser pulse to travel from the sensor to the object and back. This measurement is known as Time-of-Flight. By knowing the speed of light, the TOF value can be converted into a distance measurement.

4. **Distance Calculation**
   Using the TOF measurement, the LiDAR system calculates the distance between the sensor and the object. By multiplying the TOF value by the speed of light and dividing it by two (to account for the round trip), the distance can be determined accurately.

5. **Point Cloud Creation**
   The LiDAR sensor emits laser pulses in various directions and captures the reflections from objects within its field of view. These measurements are then combined to generate a 3D representation known as a point cloud. In the point cloud, each point corresponds to a specific distance and angle relative to the LiDAR sensor. This enables accurate mapping of the cones and other track features.

6. **Data Processing**
   The raw data from the LiDAR sensor needs to be processed to extract meaningful information. This involves applying filtering techniques to remove noise, compensating for sensor imperfections, and converting the point cloud data into a usable format. In the case of FSD racing, we can consider the tilt that the car experiences during acceleration by projecting the points back onto the correct axis and filtering the ground plane out.

7. **Application**
   With the processed LiDAR data, cones can be detected with object recognition algorithms that analyze the conical shapes out of the point cloud data. Mapping algorithms utilize the point cloud data to create detailed 3D maps of the environment or 2D-occupancy grid maps.

3.2.1.4 Desired LiDAR specifications

To optimize a LiDAR sensor for a FSD car, real-time operation is essential, necessitating rapid data capture, processing, and analysis. Achieving real-time performance involves specialized hardware and algorithms. Object detection algorithms should be flexible, fast, and accurate.

Hardware selection should consider factors such as computation speed, memory capacity, power consumption, and compatibility. Other critical specifications include range, accuracy, field of view, scan rate, data rate, laser type, point density, size, weight, power consumption, connectivity, and cost. Integration with power, control, sensors, and communication systems is crucial. LiDAR is commonly combined with cameras and GPS/IMU for a comprehensive and precise perception system in the FSD car.

Specification	**Desired Value (Example)**
Type	Solid-state
Range	Extended (50-100 meters)
Accuracy	High ($\pm$1 centimeter)
Field of View (FOV)	Horizontal: Wide ($>$120°), Vertical Narrow (30°)
Scan Rate	Sufficient (20-30 Hz)
Data Rate	High
Laser Type	NIR (905nm)
Point Density	Sufficient
Size	Compact
Weight	Lightweight
Power Consumption	Low (10 W)
Connectivity	Reliable
Cost	Cost-effective
Integration	Seamless (ROS2 & plugins)
Environmental Protection	High (IP67)
Vertical Angular Resolution	Fine ($<$0.05°)
Horizontal Angular Resolution	Fine ($<$0.05°)

Table 3.2: Desired Specifications for LiDAR Sensor in Formula Student Race Car

Table 3.2 outlines desired specifications for the LiDAR sensor, carefully chosen to optimize its performance in the context of FSD racing. Let's justify these chosen specifications:

Firstly, a wide horizontal field of view (HFOV) of approximately 120° is desired to ensure comprehensive track coverage. This allows the LiDAR sensor to capture a wide range of data, enabling the detection of cones placed at different angles and build the race-track map fast. While a 360-degree field of view (FOV) might seem desirable, it is unnecessary in the context of FSD. Placing the LiDAR sensor underneath the car would be impractical and limit its ability to detect obstacles and map the environment accurately. A more effective placement would be at the nose of the car or tilted on the rear-wing, providing a forward-facing view and covering the area directly in front of the vehicle.

A narrow vertical field of view (VFOV) of 20°-30° is preferred to focus on the track surface. By excluding unnecessary measurements in higher areas, the sensor can prioritize detecting cones and other track features, decreasing the generated data amount to enable efficient decision-making.

A high scan rate of 20-30 Hz is essential to capture rapidly changing surroundings effectively. For instance, at a speed of 100 km/h, this translates to a scan being performed every 93 centimeters. The sensor's ability to gather real-time data at such frequencies allows the car to promptly adapt to the dynamic racing environment and make split-second decisions based on up-to-date information. Faster mapping and localization imply that the car can increase its speed, particularly during the discovery lap.

A high environmental protection rating of IP67 is chosen to ensure the sensor's durability against dust and water. Formula Student races often expose the car to various weather and track conditions, and the sensor needs to operate reliably in such demanding environments, without compromising its performance.

Keeping the LiDAR sensor lightweight and compact is crucial to integrate seamlessly into the race car without interfering with its aerodynamics. By minimizing weight and size, the sensor can be mounted strategically while preserving the car's optimal performance and handling on the track.

Achieving a fine angular resolution of 0.05°-0.1° allows the sensor to capture detailed and accurate point cloud data. This level of precision is vital for object detection and mapping, enabling the car to identify and precisely locate cones, track markings, and other important elements on the race track.

Lastly, compatibility with ROS2, the Robot Operating System, ensures seamless integration into the car's software stack. This allows for efficient data processing, fusion with other sensor inputs (such as cameras or GPS/IMU), and easy integration into navigation and control algorithms, facilitating a holistic and cohesive autonomous driving system.

3.3 Moddeling the environment with SLAM

In the field of FSD, the significance of SLAM cannot be overstated. It answers a fundamental question: Can we place a mobile robot in an unknown environment and have it construct an accurate map of its surroundings while simultaneously determining its own location within that map? Solving the SLAM problem paves the way for true robotic autonomy. By leveraging SLAM, vehicles gain the ability to navigate and operate on tracks without any prior knowledge of the environment.

The most suitable SLAM approach for FSD competitions can be determined by several factors, the available sensor suite, and the required trade-offs between accuracy, computing complexity, and real-time performance. There are multiple existing SLAM algorithms available that incorporates LiDAR, camera, or a combination of both sensors. When using a 2D LiDAR system, popular choices include Cartographer, SLAM-toolbox, and GMapping. These algorithms generate 2D occupancy grid maps to represent the environment. Alternatively, for monocular and dual-lens cameras, the latest and highly regarded open-source visual SLAM solution is ORB SLAM 3. It offers robust and accurate localization and mapping capabilities.

3.3.1 Localization

Various techniques exist for localization in autonomous systems. Camera-based localization relies on extracted visual features but can be affected by factors such as poor lighting, occlusions, and low-textured environments. GPS-based localization relies on satellite signals, but its accuracy is limited, particularly in urban environments due to multipath interference. Odometry utilizes internal motion sensors to calculate position changes over time but is prone to error accumulation. In contrast, LiDAR is a reliable choice for localization as it measures ranges. Techniques like scan matching and Iterative Closest Point (ICP) are employed to localize the vehicle within a reference map. The most accurate solution often involves a sensor fusion approach using Kalman filters, particle filters, or graph-based optimization algorithms to merge measurements from multiple sensors. However, this approach increases system complexity.

3.3.2 Mapping

Mapping involves creating a model or representation of the physical world using sensor data. LiDAR-based mapping algorithms process 2D or 3D LiDAR data to generate maps that capture the geometric and spatial information of the environment. There are different mapping algorithms available, such as those that discretize the environment into cells and assign occupancy probabilities to each cell. Examples include 2D occupancy grid maps or 3D maps.

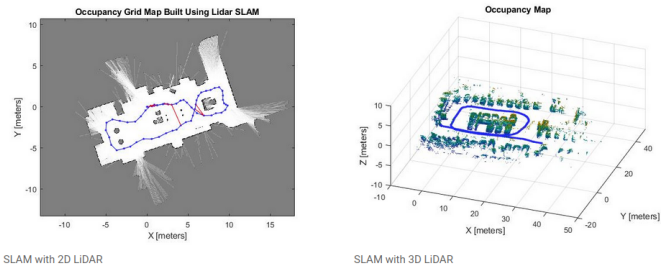

Figure 3.6: 2-D and 3-D occupancy built using LiDAR-SLAM [7]

FastSLAM, GraphSLAM, and EKF-SLAM are popular mapping algorithms used in Simultaneous Localization and Mapping (SLAM) applications. FastSLAM is a particle filter-based algorithm that combines particle representation with Rao-Blackwellized approach. EKF-SLAM, on the other hand, employs an Extended Kalman Filter to estimate the robot's pose and update the map based on sensor measurements, providing a smoother and more continuous mapping approach. GraphSLAM utilizes a graph structure to represent the environment, robot poses, and sensor measurements, allowing for optimization and creation of a consistent map.

Graph-SLAM, including approaches like ISAM2, has emerged as a popular choice over EKF-SLAM and FastSLAM due to its distinct advantages and superior performance in FSD competitions. Unlike EKF-SLAM, which relies on linearization and Gaussian noise assumptions, Graph-SLAM's graph-based approach, including ISAM2, offers higher accuracy and flexibility in handling diverse sensor data. ISAM2, in particular, combines the benefits of incremental smoothing and mapping, allowing for real-time updates of the map as the robot explores the environment. This capability makes ISAM2 a viable option for FSD competitions where dynamic changes occur frequently. Additionally, GRAPH-SLAM's loop closure detection and optimization-based trajectory estimation, found in approaches like ISAM2, overcome the limitations of FastSLAM, such as inconsistent map representation and computational complexity. These advantages make GRAPH-SLAM, including ISAM2, the preferred option for FSD competitions, as it excels in dynamic and complex environments, ensuring accurate mapping, robust localization, and efficient trajectory estimation.

3.3.3 Cartographer SLAM

The Cartographer algorithm, when used with a 2D lidar sensor, works by combining multiple scans of the environment to create accurate and detailed maps. It employs a technique called "scan-matching" using the Ceres Solver library, which aligns each new scan with the previous ones to determine the robot's position and orientation.

To build the map, Cartographer divides the environment into small units called "voxels". These voxels represent a discrete portion of space and allow the algorithm to efficiently organize the gathered data. By assigning occupancy probabilities to each voxel based on lidar measurements, Cartographer creates a global map that captures the overall structure of the environment.

Additionally, Cartographer maintains a local map that focuses on the immediate surroundings of the robot. It uses the lidar's odometry, which estimates the robot's motion based on its

internal sensors, to track its movement and update the local map accordingly. This approach helps to handle real-time mapping and adapt to dynamic environments.

By combining the global map and the local map, Cartographer creates a comprehensive representation of the environment that accounts for both long-term structure and short-term changes.

3.3.4 SLAM challenges

SLAM algorithms, much like navigating a challenging Formula Student track, face a multitude of obstacles that require careful consideration to ensure accurate mapping and localization. One significant challenge is loop closure detection, vital for correcting errors and maintaining a consistent map. If the algorithm fails to recognize revisited locations, errors accumulate, leading to significant deviations from the actual values. Similar to losing track of the racing car's position on the track, localization may fail entirely, causing the robot to lose its place on the map.

Scalability and memory efficiency are also crucial factors for SLAM algorithms. As data accumulates over time, managing memory usage and ensuring scalability become critical. This becomes especially important for large-scale mapping and long-term operation, necessitating efficient data storage and optimization techniques to maintain real-time performance. Just like a race team needs to efficiently manage their resources and make strategic decisions for success on the track, SLAM algorithms must handle data effectively to sustain optimal performance.

Dynamic environments present yet another challenge for SLAM algorithms. The presence of moving objects can interfere with mapping and localization. Similar to unexpected obstacles hindering a racing car's progress, these dynamic objects introduce errors and inconsistencies into the SLAM process, impacting its accuracy.

Furthermore, SLAM algorithms heavily rely on sensor data, akin to how a driver depends on their senses to navigate a Formula Student track. However, sensors have their limitations. They can be susceptible to noise, possess limited range, experience occlusions, or encounter varying environmental conditions. These factors directly influence the accuracy and reliability of the SLAM process, much like adverse weather conditions or visibility issues impacting a driver's performance on the track.

In addition to these challenges, SLAM algorithms entail complex computations, including feature extraction, data association, mapping, and optimization. Achieving a delicate balance between accuracy and real-time performance poses a considerable challenge, particularly for resource-constrained systems. It mirrors the quest for finding the perfect equilibrium between speed and precision on the racing track, where both are crucial for achieving success.

4.1 Hardware selection

The primary goal of our project is to develop autonomous software that forms the foundation for a future driverless car. To achieve this, we adopted a strategy of utilizing simple and affordable materials in the initial stages, enabling the development of a solid autonomous navigation system capable of performing a discovery lap and mapping the racing track using LiDAR technology. By prioritizing simplicity, we avoid the need for expensive and complex materials that would introduce computational overhead and hinder direct handling.

For the initial development, we selected a 2-wheeled robot with a bicycle kinematic model, specifically the Turtlebot3 (TB3), as it perfectly aligns with our requirements. The TB3 is equipped with relatively basic sensors, including the (LDS-01) 2-D LiDAR sensor and the (MPU9250) Inertial Measurement Unit (IMU) that contains a 3-axis accelerometer, gyroscope, and magnetometer for generating odometry data. This sensor configuration provides a suitable platform for focusing on perception rather than control, which is essential for efficient path-planning through LiDAR-based SLAM.

The computational tasks are performed on a Raspberry Pi 4B single-board computer (SBC), which also offers teleoperation capabilities through a Wi-Fi connection to a local computer. This arrangement allows for live visualization of the navigation, enabling real-time monitoring and adjustments. The Raspberry Pi 4B communicates the commands to the OpenCR1.0 board, which controls the 2 Dynamixel wheels, enabling precise and coordinated motion.

Specification	**Value**
Size	138mm x 178mm x 192mm
Weight (+ SBC + Battery + Sensors)	1kg
Battery	11.1 V Li-Po Battery
Operating Time	2 hours
CPU	ARM Cortex-M7 (168 MHz)
Sensors	2D LiDAR LDS-01 and IMU
SBC (Single Board Computer)	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU
Communication	USB, Wi-Fi
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)

Table 4.1: TurtleBot3 Burger Specifications

This table outlines the various components and features of the Turtlebot3, which contribute to its suitability for our autonomous racing system.

This setup enables efficient development iterations, cost-effectiveness, and a comprehensive understanding of the fundamental components.

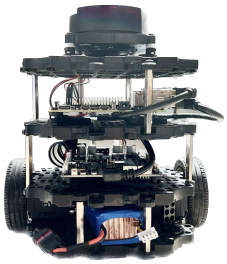

Figure 4.1: Test setup, TurtleBot3 from ROBOTIS

4.2 Software stack and selection

Opting for ROS2 as the software framework for our Formula Student driverless race car brings a range of advantages perfectly suited to this specific application. ROS2's **modularity** is a standout benefit, allowing us to create separate packages with reusable **interoperable** nodes for different modules such as perception, motion planning, control, and communication. This approach enables efficient development, quick adjustments, and easy integration of advancements or new technologies into the system without starting from scratch.

The **performance** capabilities of ROS2 are vital in the context of a driverless race car. It efficiently processes data in real-time, enabling rapid decision-making and responses. This speed is essential during races where timely actions are required. By quickly processing incoming information from sensors, ROS2 efficiently handles acceleration, braking, and steering adjustments during races, resulting in safer maneuvering and improved safety.

Reliability is another advantage of ROS2 in the Formula Student driverless race car. With robust error handling mechanisms and clear error messages, ROS2 simplifies problem identification and resolution. These clear error messages help us quickly identify issues and keep the car running smoothly during races, where reliability is crucial.

ROS2's **flexibility** is particularly beneficial in the FSD race car setting. Through YAML files, we can easily customize parameters without directly modifying the core code. This adaptability allows us to fine-tune the car's settings to varying track conditions, racing strategies, and unexpected challenges. We can adjust sensor configurations, trajectory planning algorithms, and control parameters using YAML files, optimizing the car's performance for specific race scenarios and track layouts.

In addition to these advantages, ROS2 provides advanced analysis tools that greatly assist in developing and refining the driverless race car. Tools like Rqt allow us to monitor various aspects, including the frequency of information exchange between components. Gazebo, a powerful simulation tool, helps us simulate the car's behavior in real-time virtual environments, mimicking the real-world conditions. These tools enable us to examine sensor data, validate algorithms, and simulate race scenarios, providing valuable insights for improvement and performance optimization before actual races take place.

Efficient software development is facilitated by the use of simulations, and in our case, we employ Gazebo, a powerful simulation tool that mimics the real-world environment and the behavior of the robot. Operating the simulated Turtlebot3 in Gazebo replicates the experience of the physical counterpart, enabling effective development and testing of our software. Ad-

ditionally, Jupyter notebooks are utilized to accelerate path-planning experiments, leveraging the map generated in Gazebo for iterative fine-tuning of the navigation solution until achieving satisfactory results.

We chose to use the Foxy Fitzroy version of ROS2 because it is one of the latest releases and is still actively supported. This ensures that we have access to the most up-to-date features, improvements, and bug fixes, making our system more reliable and future-proof. For programming the nodes, we decided to use a combination of Python and C++. Python is easy to use and allows for quick development and prototyping, while C++ provides better performance and control for critical tasks. This combination allows us to benefit from the strengths of both languages, enabling us to develop our system more effectively.

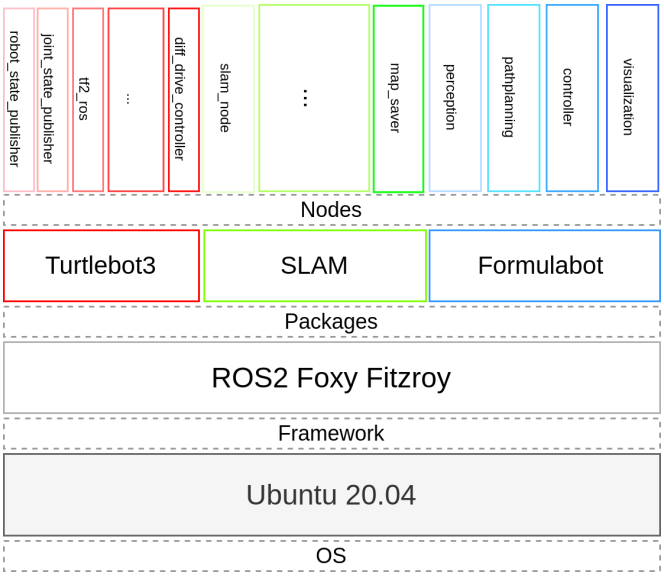

Figure 4.2: Software stack

4.3 System architecture overview

Our system architecture is designed with multiple interconnected nodes that communicate with each other through inputs and outputs. It is built upon three primary packages: TurtleBot3, SLAM (which utilizes Cartographer and SLAM-toolbox), and FormulaBot. Let's dive into the details of each package and how they contribute to the overall functionality of our system.

The TurtleBot3 package plays a crucial role in providing various modules responsible for robot functions. These modules include the robot-state-publisher, tf2-ros, joint-state-publisher, and diff-drive-controller. They are located near the robot and are responsible for tasks such as publishing the robot's state, handling transformations between coordinate frames, publishing joint states, and controlling the robot's differential drive system.

On the other hand, the SLAM packages are responsible for mapping and localization tasks. These packages utilize advanced algorithms like Cartographer and SLAM-toolbox to create accurate maps of the environment and determine the robot's position within that map.

Our custom logic is implemented within the FormulaBot package, which acts as a host for four core nodes: Perception, Path Planning, Control, and Visualization. Each of these nodes has its specific role in the system.

The Perception node receives crucial updates from the root nodes in the TurtleBot3 suite. These updates include information like the robot's odometry and LiDAR scans. The Perception module then processes this data to extract cone positions, which are essential for the subsequent steps in the system.

The extracted cone positions are used by the Path Planning node to compute a centered path. This path serves as the navigational input for the Control component. The Control node processes the waypoints provided by the Path Planning unit and generates steering and velocity instructions. These instructions are vital for guiding the robot along the desired path and ensuring precise control over its movements.

The seamless exchange of information is a key aspect of our system architecture. Once the updated map reaches the SLAM module via the FormulaBot perception unit, cone positions are extracted and utilized by the Path Planning unit. The computed centered path is then sent to the Control component, which generates steering and velocity directives. Finally, these directives are transmitted to the TurtleBot3 nodes for execution, allowing the robot to follow the desired path and navigate effectively.

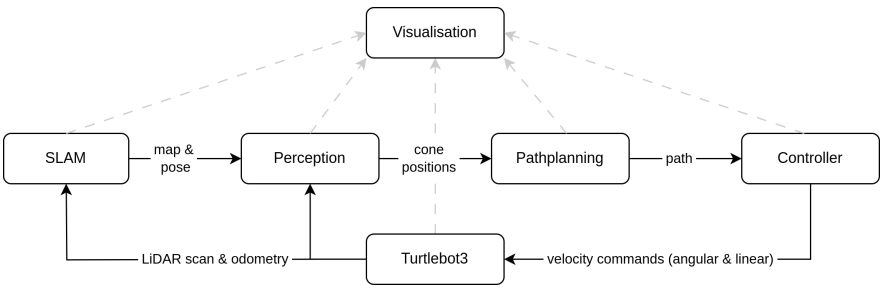

Figure 4.3: Communication flow

4.4 Perception

The initial approach for cone detection involved analyzing a single LiDAR scan. We removed unreliable values and grouped the remaining points to find cone positions. However, this approach had limitations in terms of accuracy and efficiency.

From this initial attempt, we learned that relying only on a single scan posed challenges. One challenge was false positives, where objects other than cones were mistakenly identified as cones. This could lead to incorrect cone positions and navigation errors.

Another challenge was misinterpreting cones during the robot's turns. As the robot changed direction, it could misjudge the cone positions, causing it to deviate from the intended path. This was risky as it could result in collisions or missing the desired targets.

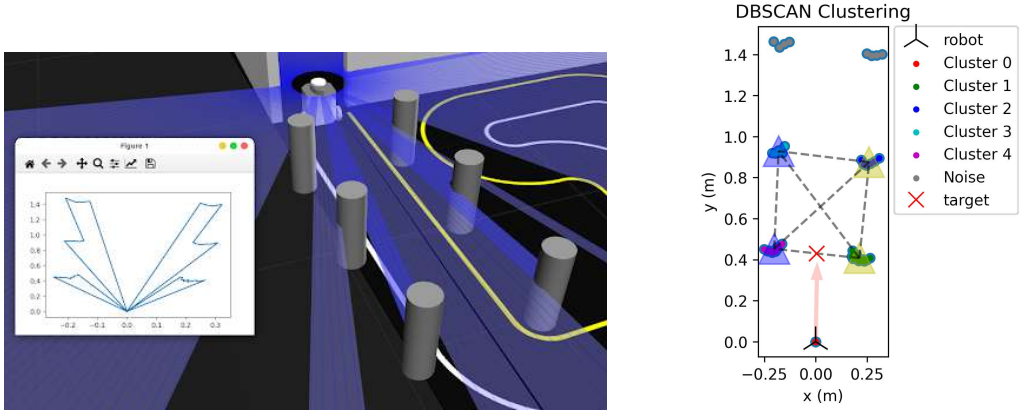

Figure 4.4: Early perception and navigation approach: Directing towards the centroid of the two nearest clusters located ahead

Based on the insights gained from the initial approach, we identified several key improvements for the perception pipeline. We recognized the importance of filtering to effectively eliminate false-positive cone detections. To achieve more accurate cone positions and avoid collisions, a more precise method for approximating cone locations was deemed necessary. Additionally, taking into account the robot's orientation relative to the cones emerged as a crucial consideration. Moreover, we acknowledged the need to utilize more data to gain a comprehensive understanding of the surrounding environment.

To address these challenges, we adopted a proposed approach that involved continuously updating a map of the environment, rather than relying solely on the latest LiDAR scan. This approach utilized the Simultaneous Localization and Mapping (SLAM) technique, which enabled real-time mapping and localization. The output of the SLAM process was an occupancy-grid map represented by gray values, providing insights into the probability of each cell being occupied.

Once the map was obtained and the robot's position was localized within it, the next step was to extract the cone positions. We began by preprocessing the map to ensure correct scaling. To identify potential cone locations, we applied a thresholding method that retained only cells with high probabilities of occupancy. Since the map could be visualized as an image, we employed contouring techniques to divide it into distinct regions. To filter out false positives, we chose a circle-fitting approach. By fitting a minimum enclosing circle around the contours, we could eliminate circles with radius values that were either too small or too large. This method not only helped filter out false positives but also provided an accurate estimation of the circle center.

In addition to the SLAM map, we upgraded the LiDAR scan processing to serve as a com-

plementary method in case the SLAM map failed to provide reliable information. To identify potential cone locations, we divided the LiDAR scan into clusters. Traditional clustering methods like K-means clustering were not suitable due to the unknown number of cones. Instead, we utilized techniques such as Euclidean clustering or Density-Based Spatial Clustering of Applications with Noise (DBSCAN). For instance, DBSCAN randomly selected a data point and expanded clusters by connecting nearby points within a specified distance threshold. Points that couldn't be connected to any cluster were considered noise. Through an iterative process, the algorithm explored the data, forming clusters based on density and effectively distinguishing outliers from the primary clusters. In this approach, only the cluster threshold distance and the minimum number of points needed to be specified, eliminating the need to predefine the number of clusters in advance.

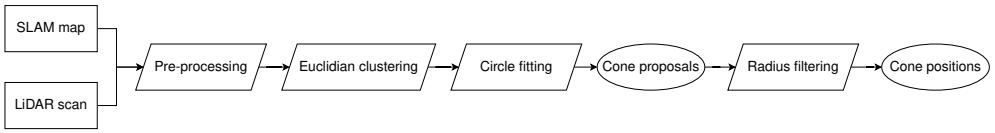

Figure 4.5: Perception pipeline

4.5 Path-planning

The racing strategy we aim to develop initially involves a discovery lap, during which the car explores and maps the track at a relatively low speed. The purpose of this lap is to minimize errors by following a safe path, which means driving right between the cones rather than in close proximity to them. In our previous approach, as mentioned in the previous section, we drove towards the center of the two nearest front clusters. However, we recognized the need for a more effective path-planning methodology. To address this, we have chosen to employ Delaunay triangulation to compute the waypoints.

Delaunay triangulation is a widely used technique in computational geometry. It involves dividing a set of points into triangles in such a way that no point falls within the circumcircle of any triangle. By performing Delaunay triangulation on the cone positions in front of the robot, we can obtain a triangulated representation of the cones. This triangulation provides a structured set of waypoints that can guide the robot along a path through the cones.

In order to determine the appropriate path between the cones, we need to consider the colors of the cones. However, since our 2D LiDAR camera cannot detect colors, we need to rely on alternative methods to estimate the cone colors. We have considered two specific techniques to address this challenge.

The first technique involves extrapolating the previously driven path, which forms a polyline, and utilizing the concept of winding numbers. Winding numbers are a mathematical concept that can be used to determine the number of times a curve winds around a given point. By calculating the winding numbers of the cones with respect to the extrapolated path, we can estimate their colors. However, we found that this method is not entirely reliable, particularly in situations where sharp turns are encountered. The extrapolation may not accurately capture the curvature of the track, leading to incorrect color estimations.

As an alternative, we have adopted a second approach that utilizes a concave hull. A concave hull is a polygon that encompasses a set of points while allowing for concavities in its shape. We fit a concave hull over the front cones, with a certain maximum degree of concavity specified. The concave hull provides a reasonable approximation of the track shape. We then compute the midpoints of the Delaunay edges within the hull. These midpoints serve as potential waypoints for the robot's path.

To filter out any midpoints that may fall outside the hull and are not relevant to the path, we compare each midpoint with the hull or polygon. If a midpoint lies outside the hull, it is discarded from the set of waypoints. This filtering process ensures that the robot remains on a feasible and safe path within the track boundaries.

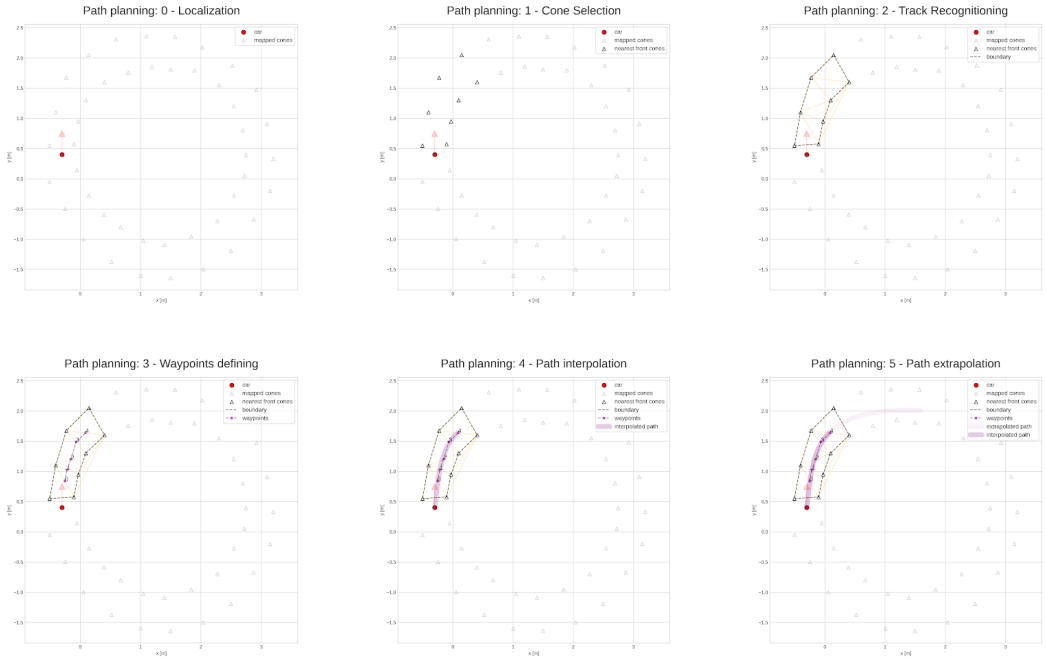

Figure 4.6: The path-planning process

4.6 Control

As the focus of the thesis lies on the perception pipeline and path planning, we wanted to use a simple controller for our robot. Since it utilizes a bicycle model as the kinematic model, it would require modifications for implementation on the future car control system. Initially, our approach involved computing a few waypoints that the car needed to navigate to in order to follow an obstacle-free path. However, we discovered a limitation in our initial implementation: steering and throttling were performed separately. This led to an issue where the car would come to a stop at each waypoint, adjust its orientation, and then accelerate towards the next waypoint. Consequently, during sharp turns, the car's alignment became misaligned as all the steering occurred abruptly instead of gradually. This misalignment caused the car to miss some of the right-front cones during the navigation process.

Therefore, we needed a simple solution to address this non-adaptive and blocking control system. After thorough research, we discovered the pure pursuit controller as a suitable option. We chose it due to its effectiveness, simplicity, and the underlying mathematical principles it employs. The pure pursuit algorithm combines geometry and trigonometry to guide the car along the desired trajectory.

At its core, the algorithm continuously calculates a look-ahead point on the desired path. To achieve this, the computed waypoints are interpolated to obtain a continuous path. The look-ahead point represents the target position that the car should aim for to accurately follow the desired trajectory. The distance to the look-ahead point is typically determined based on the car's current speed and dynamic characteristics. During the experimentation phase, we iterated

multiple times to find an optimal look-ahead distance, considering the robot's maximum speed of 0.22 m/s and the speed of the SLAM algorithm that generates the occupancy grid map used for waypoint computation.

To determine the required steering angle for reaching the look-ahead point, the algorithm takes into account the path's curvature. It calculates the radius of the circle that best fits the car's current position and orientation, intersecting with the desired path. The curvature of this circle, represented by its radius, influences the steering angle adjustment.

Trigonometry is employed in the pure pursuit algorithm to calculate the steering angle. By considering the distance between the car's front axle and the look-ahead point, as well as the radius of the circle described above, the algorithm determines the appropriate steering angle that aligns the car with the desired trajectory.

The elegance and simplicity of the pure pursuit controller lie in its ability to handle various path types, including straight lines and curves, using these mathematical principles. This versatility makes it particularly well-suited for FSD competitions, where the track features challenging geometries.

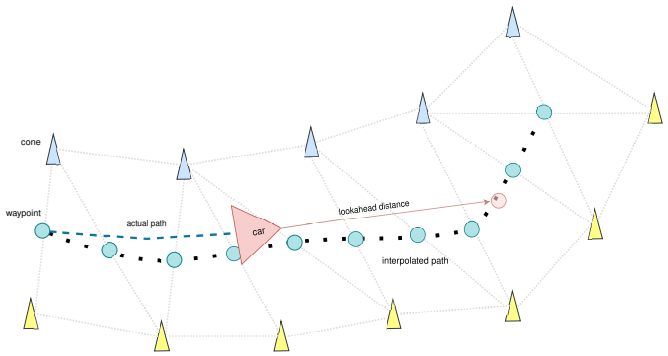

Figure 4.7: Pure pursuit controlled motion on a race track

Furthermore, the algorithm's robustness to disturbances and uncertainties stems from its capability to handle minor deviations from the desired path without significant errors. We observed that even when an off-track waypoint was computed, the car could quickly bypass it and switch to the next correct waypoints without colliding with cones. This resilience is crucial in real-world scenarios where factors like uneven surfaces or external forces can impact the car's trajectory.

$$L = \sqrt{(x_a - x)^2 + (y_a - y)^2} \tag{4.1}$$

$$\alpha = \arcsin\left(\frac{2L\sin(\delta)}{R}\right) \tag{4.2}$$

$$\theta = \text{atan2}(y_a - y, x_a - x) \tag{4.3}$$

$$\delta = \text{atan2}(2 \cdot L \cdot \sin(\alpha), R) \tag{4.4}$$

In these formulas:

- L represents the distance between the vehicle's current position (x, y) and the lookahead point (x_a, y_a).
- R denotes the desired radius of the path or curvature.
- α is the angle between the vehicle's heading direction and the line connecting its position to the lookahead point.
- δ corresponds to the steering angle or the required control input to track the desired trajectory.
- θ represents the angle of the line connecting the vehicle's position to the lookahead point.

These formulas play a fundamental role in calculating the steering angle δ based on the vehicle's position and the desired trajectory, allowing the pure pursuit controller to guide the vehicle along the desired path.

Experimental results and discussion

The primary goal of these experiments is to evaluate the effectiveness of our autonomous navigation system. We want to identify any potential issues or challenges that may arise and improve the system's overall performance. To achieve this, we have used Jupyter Notebooks to fine-tune and select the best navigation strategy. Jupyter Notebooks provide us with an interactive platform to experiment with different algorithms and settings.

In addition, we have employed Gazebo as a simulated environment for our robot. Gazebo allows us to create a virtual world where our robot can navigate and interact with objects and obstacles. By testing our system in this simulated environment, we can quickly detect any errors or problems and make the necessary adjustments.

To further validate our system, we will conduct physical trials on various tracks. This involves deploying our autonomous navigation system on real robots and gathering data in different real-world environments. For example, during these trials, we will evaluate how our system handles high speeds and sharp turns. We will assess its ability to navigate safely, avoid collisions, and make appropriate decisions in complex environments. We will also measure its accuracy and speed across different tracks to analyze its performance.

The data collected from both the simulated environment and physical trials will be crucial for refining our autonomous navigation system. Each experiment was visually recorded and data was captured using rosbags. By analyzing the results and identifying areas for improvement, we can adjust the algorithms, optimize settings, and enhance the system's reliability and efficiency.

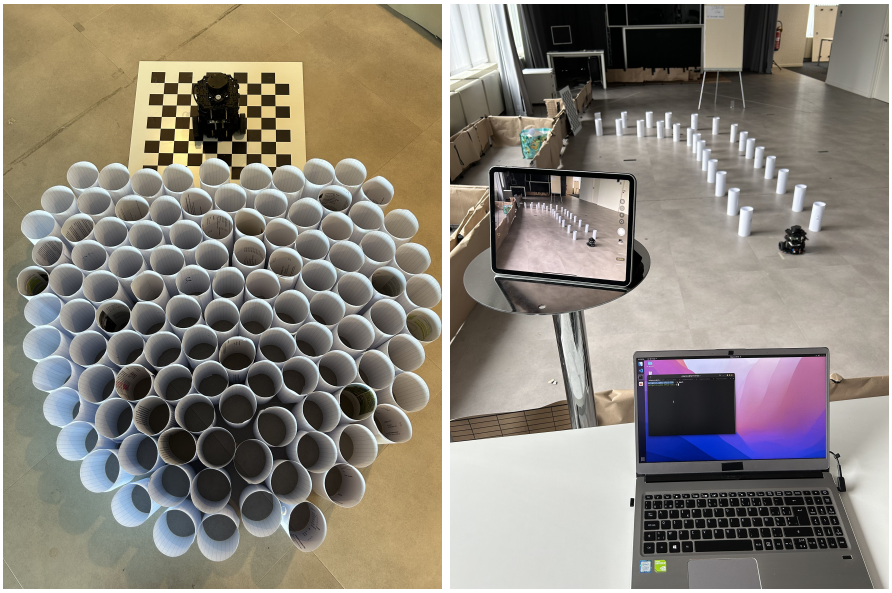

Figure 5.1: Experimental setup at Pleinlaan 9, ETRO-lab

5.1 Experiment A: Gazebo Simulations

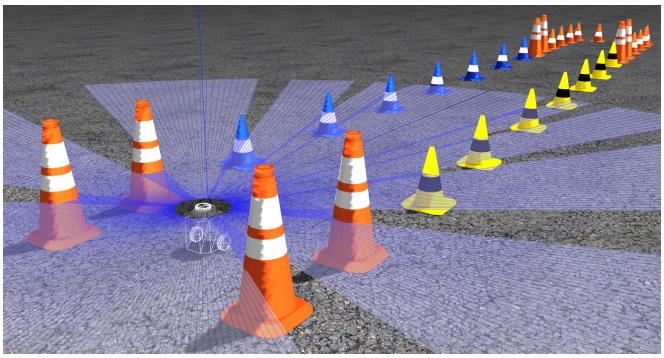

Figure 5.2: Acceleration event simulation in Gazebo

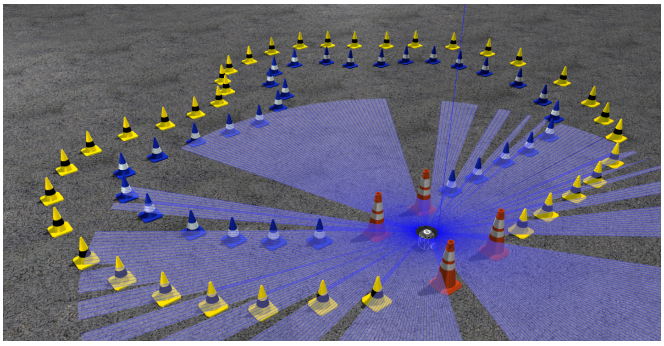

Figure 5.3: Autocross event simulation in Gazebo

The experimental setup aimed to evaluate the performance of our autonomous navigation system in the Gazebo simulation environment. Two specific scenarios were considered: a straight path and a closed-loop track. The objective was to assess the system's ability to navigate accurately and efficiently in an noise-free environment.

For the straight path scenario, a virtual environment was created in Gazebo consisting of a linear path with defined start and end points. The path was free of obstacles, providing a clear and unobstructed route for the autonomous robot. This scenario allowed for the evaluation of the system's basic navigation capabilities, including path following and obstacle avoidance.

In the closed-loop track scenario, a more complex environment was designed, featuring a circuitous track with curves and turns. This scenario enabled a comprehensive assessment of the system's performance in terms of path planning, trajectory tracking, and obstacle detection.

Multiple test runs were conducted for each scenario.

5.1.1 Results

In the straight path scenario, the autonomous robot successfully followed the defined linear path without significant deviations. The mapping accuracy was high, and the generated maps accurately represented the environment.

In the closed-loop track scenario, the system faced more complex challenges, including curved paths and potential obstacles. Despite these complexities, the autonomous robot demonstrated proficient path planning and trajectory tracking capabilities. It successfully maneuvered through the various curves of the track, avoiding collisions with obstacles.

5.1.2 Discussion

The results obtained from the experiments indicate that the autonomous navigation system is capable of reliably navigating in simple, unobstructed environments. In the straight path and closed-loop scenario, the system successfully followed the predefined linear path without significant deviations. This demonstrates the system's ability to effectively process sensor data, perform accurate localization, and generate appropriate control commands to steer the robot along the desired trajectory.

5.2 Experiment B: Track segments: straight, turn and hairpin

In order to assess the robot's navigation capabilities on the track, we conducted a series of experiments involving three critical segments: a straight path, a high-radius turn with a 4m radius, and a hairpin. To create the track setup, A4 paper cones with a diameter of 12 cm were utilized, maintaining a distance of approximately ± 60 cm between the cones. With an angle resolution of one degree, we observed that six points would be reflected when a cone was positioned 30 cm ahead, while only one point would be reflected for a cone located 300 cm ahead. The robot's perception and navigation were visualized by establishing a Wi-Fi connection between the robot and my local computer. This enabled real-time monitoring and control of the robot's operations.

The physical experiments were performed in the laboratory of the ETRO Department of Electronics and Informatics at the Vrije Universiteit Brussel. This laboratory provides a controlled and well-equipped environment for conducting the experiments accurately and reliably. The lab is a spacious open area with an usable flat surface measuring 6 by 11 meters, as depicted in fig. 5.4. Additionally, the lab was mapped, providing a detailed layout representation as shown in fig. 5.10.

5.2.1 Straight path

Figure 5.4: Setup of the straight path

5.2.1.1 Results

During the initial trial, the robot was able to navigate through a straight path using the autonomous navigation. The detection of cones from the SLAM map was initially successful; however, we encountered a challenge with the low frequency of map publication and slow navigation. Also occlusion occured as all the cones were aligned in a straight line.

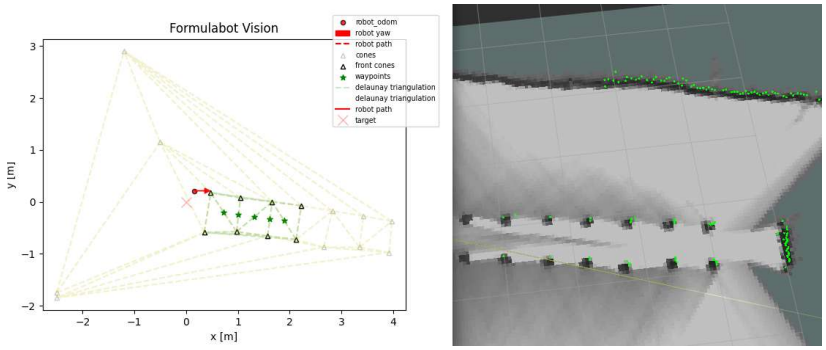

Figure 5.5: Robot vision and mapped track

5.2.1.2 Discussion

In this trial, we evaluated the robot's ability to autonomously navigate through a straight path. Initially, the robot successfully detected the cones from the SLAM map. However, we faced a challenge due to the low frequency of map publication, which resulted in slower navigation. To address this limitation, we adjusted the Cartographer configuration file to increase the map update frequency. Consequently, the robot's navigation through the straight path showed significant improvement in navigation speed.

An interesting observation during this trial was the occurrence of occlusion, where all the cones were aligned in a straight path. It is important to highlight that occlusion is an inherent limitation of technologies such as LiDAR or RGB camera, where objects can obstruct the line of sight. In our case, occlusion caused the robot to map a cone only when it approached it

closely, leading to delayed cone detection. However, it is worth noting that this issue can be mitigated by employing a RADAR sensor, which is less susceptible to occlusion.

5.2.2 High-radius turn and hairpin

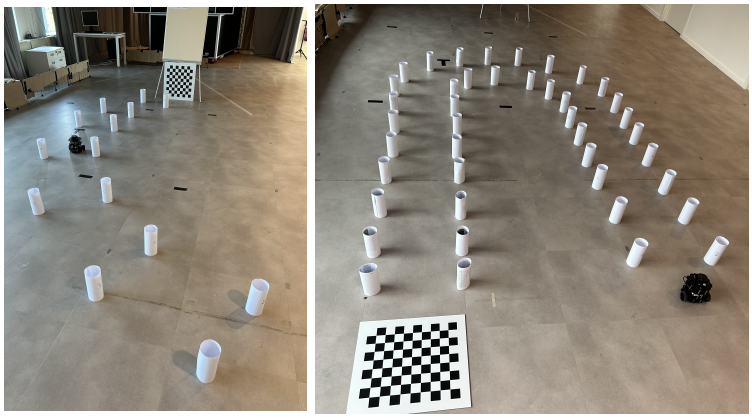

Figure 5.6: Setup of the high-radius turn and hairpin

5.2.2.1 Results

In the following experiment, our autonomous robot was tested on a curved track consisting of two segments: a high-radius turn and a hairpin.

In the first segment, the high-radius turn, the robot successfully navigated through the curved path. It maintained a centered trajectory without colliding with any cones or deviating from the intended path.

However, in the second segment, the hairpin, which involved a sudden and sharp 180-degree turn, the robot initially followed the correct path. Unfortunately, halfway through the turn, the robot mistakenly identified and pursued the wrong cones, leading it off the designated track.

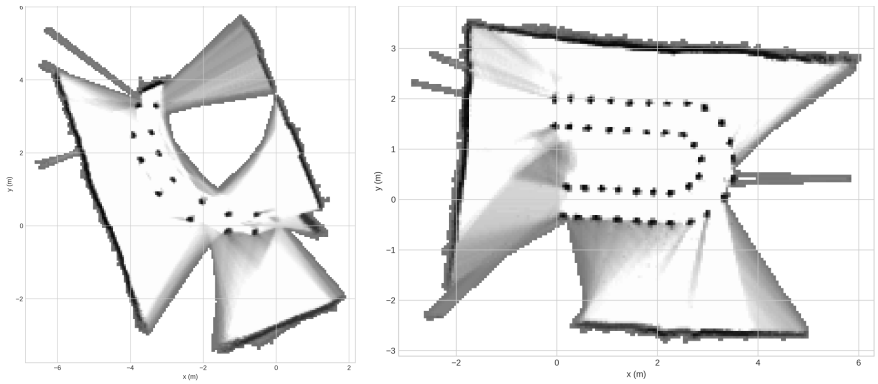

Figure 5.7: Small turn and hairpin mapped with Cartographer in ETRO-lab

5.2.2.2 Discussion

During the small turn trial, we observed that our robot successfully computed the correct waypoints based on the cones ahead of it. However, during the hairpin turn, we encountered some difficulties, particularly in the steering aspect, while the waypoint computation remained accurate. The robot exhibited understeering behavior, causing it to veer off the track and

collide with a cone. As a result of this poor positioning, the robot computed waypoints based on the incorrect cones, leading to continued deviation from the track.

To address these issues, we made an important observation and subsequently implemented an improvement in the pure pursuit controller. The modification involved adapting the steering velocity of the robot according to the degree of the turn. By adjusting the controller's behavior based on the specific characteristics of each turn, we enhanced the robot's steering performance and reduced instances of understeering.

This adjustment in the pure pursuit controller was crucial for improving the robot's ability to navigate through high-radius turns and hairpin turns accurately. On fig. 5.8 we can observe the smooth trajectories our improved pure-pursuit controller is producing on curved tracks.

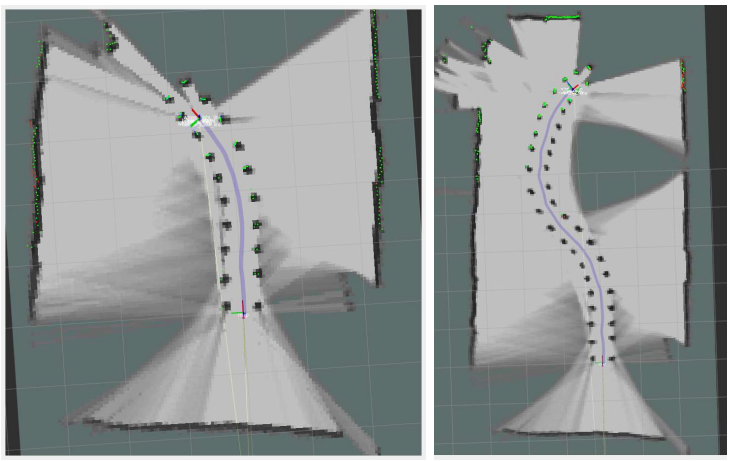

Figure 5.8: Pure pursuit controlled motion producing a smooth centered trajectory

5.3 Experiment C: navigation on a closed loop track

The purpose of this experiment is to assess the robot's ability to perform a discovery lap on an unknown closed loop track, resembling the trackdrive FSD event. The primary objective is for the robot to navigate the track safely while simultaneously mapping its surroundings.

To replicate the trackdrive scenario, we arranged 94 cones in the ETRO laboratory, creating a closed loop track as on fig. 5.9. The track features various segments, including a straight path, a small chicane, a high-radius turn with a radius of 3 meters, and a sharp turn with a radius of 2 meters.

The experiment involved the robot autonomously navigating through the track while mapping its environment during three trials at four different velocities. Throughout the experiment, we collected data on the robot's trajectory, perception, and mapping capabilities through rosbags. The sensor data, including LiDAR scans, and odometry readings, were recorded for further analysis.

Figure 5.9: Setup of the closed loop track

5.3.1 Speedtest

Our primary objective was to stress-test our navigation system at different speeds for the discovery lap. To achieve this, we conducted a series of experiments consisting of three trials at five distinct speeds. The robot was positioned at the same starting point on the track for each trial. By systematically varying the speeds, we aimed to observe and analyze the robot's performance at different velocities.

Through these trials, we meticulously examined the robot's behavior, carefully noting any instances of failure or limitations in the system. This approach provided us with valuable insights into the factors influencing the robot's performance at different speeds. By thoroughly analyzing the collected data, we gained a comprehensive understanding of how the robot's navigation capabilities were affected by various speeds.

5.3.1.1 Results

During this experiment, three metrics were analyzed: lap time, the number of times a cone was hit, and the accuracy of mapping during autonomous navigation. The experiment was conducted at different speeds, including 0.10 m/s, 0.15 m/s, 0.18 m/s, and 0.22 m/s.

At a speed of 0.10 m/s, the robot had sufficient time to generate a usable map and accurately extract cone positions. All cones were successfully mapped, although there were some instances

of false positives due to the basic cone filter used. The robot navigated along the correct path with minimal deviation from the planned trajectory, resulting in excellent mapping accuracy.

When the speed was increased to 0.15 m/s, the experiment presented slightly more challenges. However, the generated maps remained usable and accurately represented the occupancy cells of the track. There were a slightly higher number of cone collisions compared to the trials at 0.10 m/s, consistently occurring at the same location. Despite this, the pure pursuit controller performed well, keeping the robot close to the planned path with acceptable deviation.

Speed Test	Trial	Lap Time (s)	Cone Hits	Mapping Accuracy
0.10 m/s	1	219	1	excellent
	2	206	0	excellent
	3	211	0	excellent
	Avg	**212**	**0.3**	**excellent**
0.15 m/s	1	185	3	good
	2	162	0	very good
	3	169	1	very good
	Avg	**172**	**2**	**very good**
0.18 m/s	1	155	5	good
	2	150	4	good
	3	160	6	mixed
	Avg	**155**	**5**	**good**
0.22 m/s (max. speed)	1	not completed	6	mixed
	2	not completed	5	bad
	3	not completed	7	bad
	Avg	/	/	**bad**

Table 5.1: Experimental Results

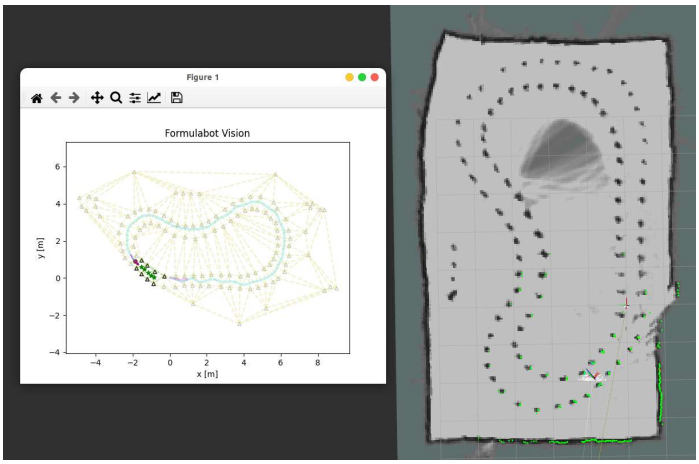

Figure 5.10: Robot vision and map of the closed loop track

5.3.1.2 Discussion

The results of the experiment indicate that the speed of the robot significantly affects the mapping accuracy and cone collision rate. At higher speeds, such as 0.18 m/s and 0.22 m/s, there was a noticeable decrease in mapping accuracy. This decrease can be attributed to the low refresh rate of 5 Hz for the map. The infrequent updates of the map led to inaccurately estimated cone positions and compromised robot localization.

Consequently, at a speed of 0.18 m/s, there was an increased number of cone collisions due to the inaccuracies in cone position estimation and robot localization. The map was still usable,

but the compromised accuracy impacted the overall performance. When the speed was further increased to 0.22 m/s, the map became completely unusable. This resulted in map shifting and incomplete laps as the robot struggled to accurately navigate the track.

It is evident that the refresh rate of the map plays a crucial role in maintaining mapping accuracy during high-speed navigation. Improving the refresh rate or utilizing more accurate sensing techniques, such as higher resolution LiDAR systems, could mitigate the challenges observed at higher speeds.

5.4 Faults inventory

During the experiment, we encountered a significant problem related to map-shifting caused by the cartographer as on fig. 5.11. This happened when the robot was operating at its highest linear speed of 0.22 m/s and angular velocity of 2.84 m/s, the generated map became misaligned with the actual track. As a result, this led to inaccurate localization and incomplete laps, affecting the overall performance of the system.

Additionally, we observed the occurrence of false positives during cone detection as on fig. 5.12. The basic cone filter, combined with the limitations of the LiDAR system's accuracy, resulted in objects other than cones being mistakenly identified as cones. When two cones are positioned too closely together, they may appear as a single object to the LiDAR sensor, leading to incorrect detections and resulting in false negative predictions. To address this issue, it is crucial to enhance the cone filter algorithm to improve its ability to differentiate between cones and other objects. Alternatively, considering the adoption of a more accurate sensor could also help mitigate the inacuracies.

Furthermore, we noticed a decrease in mapping accuracy as the speed increased, particularly at speeds of 0.18 m/s and 0.22 m/s. This resulted in inaccurately estimated cone positions and compromised robot localization. The low refresh rate of 5 Hz for map updates posed limitations on the system's ability to capture rapid changes in the environment. To overcome this challenge, it is recommended to explore options such as increasing the map refresh rate, utilizing higher-resolution sensors, or investigating alternative mapping techniques. These improvements would enhance mapping accuracy, especially at higher speeds, and contribute to more reliable robot localization.

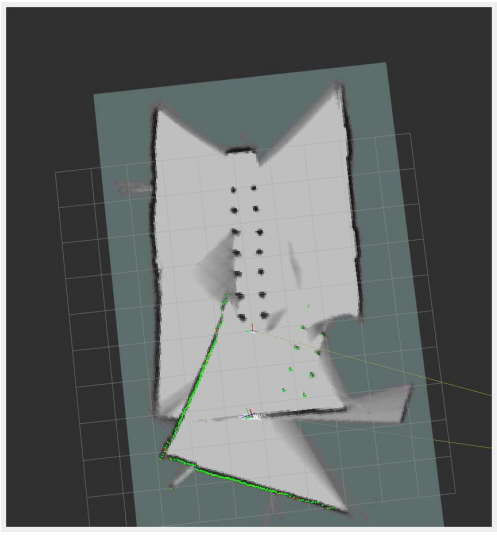

Figure 5.11: Map shifting occurring as a result of excessive linear or angular velocity

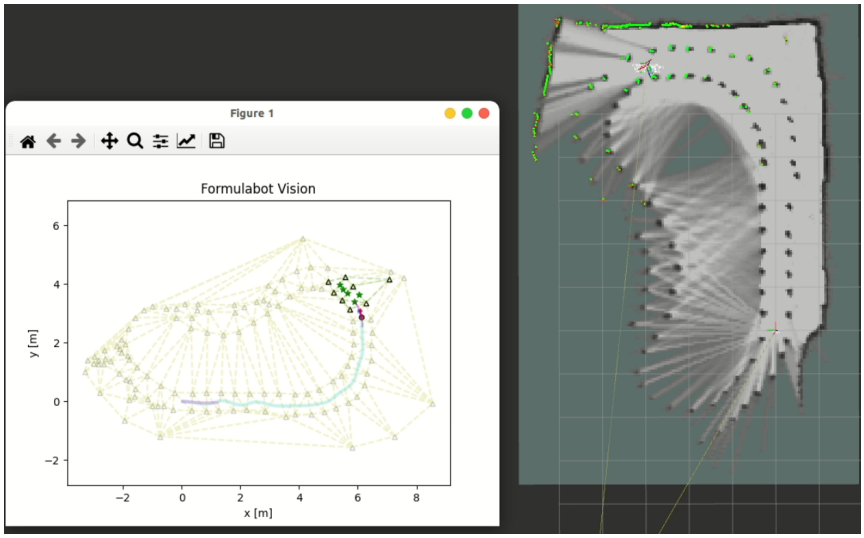

Figure 5.12: The occurrence of false positive cone detections resulted in the robot hitting a cone or deviating from the intended path.

Chapter 6

Conclusions

In conclusion, this study has provided us with invaluable insights into the performance of our autonomous navigation system. Through a series of experiments, we have gained a deeper understanding of how the system behaves at different speeds, uncovering its strengths and limitations along the way. We were pleased to observe that at lower speeds, the system exhibited exceptional mapping accuracy and demonstrated precise cone detection, which are crucial elements for successful navigation. However, as the speed increased, we encountered certain challenges that impacted the system's performance. We observed a decrease in mapping accuracy and an increase in false positive cone detections. These challenges can be attributed to the limitations of our LiDAR system's accuracy and the relatively low refresh rate of our map. To overcome these limitations, we recommend focusing on enhancing our cone filter algorithm and exploring the utilization of higher-resolution sensors.

Moreover, this thesis has allowed us to propose a comprehensive software architecture and racing strategy specifically designed to meet the unique demands of FSD races. We have developed a strategy that leverages the map generated by our SLAM package for localization, cone position extraction, and efficient path planning. To overcome the challenge posed by our system's color-blindness, we have introduced a novel concave hull method, enabling us to approximate the track boundary and compute relevant waypoints. These waypoints serve as the foundation for interpolating, extrapolating, and generating a safe and centered path between the cones, which our system navigates with the guidance of a reliable pure pursuit controller. We are confident that our contributions will significantly advance the research and development efforts of VUBR, providing practical solutions and valuable insights for the field of autonomous racing.

Chapter 7

Future work

We will now outline several potential directions for future research and development based on the findings and limitations identified in our current study. These areas of focus aim to further enhance the capabilities and performance of our driverless VUBR package.

- **Developing Cone Prediction:** Explore the implementation of cone prediction algorithms using Convolutional Neural Networks (CNN). This would enable the robot to anticipate cone positions ahead of time, improving trajectory planning and race performance.
- **Model Predictive Control (MPC):** Investigate the application of Model Predictive Control techniques to optimize the robot's navigation. By incorporating real-time sensor feedback and predictive models, MPC can generate optimal control inputs for precise and efficient trajectory planning.
- **ISAM2 for SLAM:** The implementation of Incremental Smoothing and Mapping 2 (ISAM2) appeared to significantly reduce the run-time for the AMZ FS team, achieving a three-fold decrease. ISAM2 provides an efficient and robust approach for map building and localization, thereby enhancing the overall efficiency and speed of our future system.
- **Optimal trajectory planner:** Developping algorithms to compute the best trajectory on a mapped race track. Utilizing the track map, consider the robot's dynamics & velocity profiles, constraints, and race objectives to generate optimized racing paths for improved performance.

By addressing these future research directions, we can further improve our driverless package in order to ultimately being able to participate to the FSD dynamic competitions. These areas present exciting opportunities for future studies.

Code

A.1 Source code

The code and data used for this project can be found in the GitHub repository at the following link: https://github.com/valigatotuS/LiDAR-FSAE

A.2 Most used commando's

Table A.1: ROS2 common commands per category

Category	Bash commando
Nodes	Create a new package: `ros2 pkg create <pkg_name>`
	List running nodes: `ros2 node list`
RQT	Launch the RQT Graph tool: `ros2 run rqt_graph rqt_graph`
Topics	List topics: `ros2 topic list`
	Show information about a topic: `ros2 topic info <topic_name>`
Services	List services: `ros2 service list`
	Call a service: `ros2 service call <service_name> <srv_type> <args>`
Parameters	Set a parameter: `ros2 param set <node_name> <param_name> <value>`
	Get a parameter: `ros2 param get <node_name> <param_name>`
Actions	List actions: `ros2 action list`
	Send a goal to an action server: `ros2 action send_goal <action_name> <action_type> <args>`

Figures

B.1 Race tracks

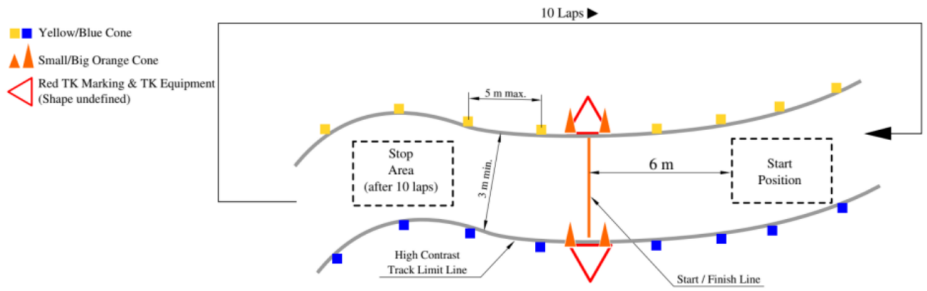

Figure B.1: Trackdrive track layout (FSG, 2018)

Bibliography

[1] AMZFormulaStudent. *AMZ Rollout 2023*. YouTube video. 2023. URL: %7Bhttps://www.youtube.com/watch?v=t0-4lk-hY98%7D.

[2] S. Bhaskaran, K. Zhou, A. Baab, and R. Calhoun. *Autonomous Vehicle Lidar: A Tutorial*. Jan. 2019. ISBN: 978-1-65327-791-9.

[3] H. Durrant-Whyte and T. Bailey. 'Simultaneous localization and mapping: part I'. In: *IEEE Robotics & Automation Magazine* 13.2 (June 2006). Conference Name: IEEE Robotics & Automation Magazine, pp. 99–110. ISSN: 1558-223X. DOI: 10.1109/MRA.2006.1638022.

[4] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, et al. 'AMZ driverless: The full autonomous racing system'. In: *Journal of Field Robotics* 37.7 (2020). Publisher: Wiley Online Library, pp. 1267–1294.

[5] S. Nekkah et al. *The Autonomous Racing Software Stack of the KIT19d*. arXiv:2010.02828 [cs]. Oct. 2020. URL: http://arxiv.org/abs/2010.02828 (visited on 05/05/2023).

[6] H. Reichert, L. Lang, K. Rösch, D. Bogdoll, K. Doll, B. Sick, H.-C. Rellss, C. Stiller, and J. M. Zöllner. 'Towards Sensor Data Abstraction of Autonomous Vehicle Perception Systems'. In: *2021 IEEE International Smart Cities Conference (ISC2)*. 2021, pp. 1–4. DOI: 10.1109/ISC253183.2021.9562912.

[7] 'What Is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink - MATLAB & Simulink'. In: (2023). URL: https://nl.mathworks.com/discovery/slam.html.