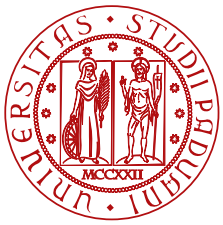


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

MASTER'S DEGREE IN COMPUTER ENGINEERING

A Stereo Vision SLAM Front-End for the Formula Student Driverless Competition

Supervisor
Prof. Pretto Alberto

Student
Minorello Fabio
ID number
2087924

ACADEMIC YEAR 2024-2025

Graduation date 27/03/2025

Acknowledgements

I dedicate this thesis to my grandmother, who passed away this year.
Her love, strength, and unwavering support have always been a guiding light in my life.

A special thanks goes to Professor Alberto Pretto for his guidance and expertise, which were fundamental in shaping this work, and for giving me the opportunity to explore a topic I am truly passionate about.

I would like to extend my appreciation to the Race UP Driverless team, with whom I had the privilege of collaborating. The dedication, teamwork, and passion of this team made this experience truly enriching and rewarding.

I am also profoundly thankful to my family for their constant support throughout this journey, providing me with the motivation and reassurance needed to overcome every challenge.

Finally, a heartfelt thank you to the friends with whom I have shared both the challenging moments and the joyous celebrations. Your unwavering presence and support have been invaluable in overcoming obstacles and making every accomplishment even more meaningful.

Abstract

Formula Student is an international engineering competition where student teams design, build, and race formula-style cars. The Driverless category challenges teams to develop autonomous vehicles capable of navigating and competing in dynamic and static events without a human driver. To achieve this, the car must construct a map of its environment and localize itself within it, essentially solving the well-known Simultaneous Localization and Mapping (SLAM) problem.

The SLAM process is typically divided in front-end, which involves sensor data collection and landmark extraction, and back-end, which focuses on optimizing the vehicle's estimated trajectory and the generated map.

This thesis, developed in collaboration with the Race UP Driverless team at the University of Padova, presents a Stereo Vision SLAM Front-End system for detecting and classifying the cones that delimit the track while estimating their positions along with a covariance matrix that quantifies the uncertainty of the estimation.

A new stereo camera setup using FLIR Blackfly S cameras is introduced, replacing the previous system and enhancing depth estimation. The thesis discusses the challenges of stereo vision calibration, synchronization, and integration into the existing software stack.

Additionally, a fine-tuned YOLO11-Pose model is presented, detailing the rationale behind its selection and the dataset creation process for training. This model improves classification accuracy, detects cones, and predicts peak point positions, all while being more computationally efficient than the previous setup.

The entire pipeline is implemented in ROS (Robotics Operating System) and evaluated against the previous system, demonstrating significant improvements in accuracy and computational efficiency.

Contents

Chapter 1

Introduction

Self-driving cars have the potential to revolutionize transportation, improving both safety and efficiency. While developing autonomous vehicles is a complex challenge, significant advancements have been made in recent years. Companies such as Waymo, Cruise, and Tesla have already deployed driverless cars in real-world urban environments, demonstrating the feasibility of this technology.

One of the key advantages of autonomous vehicles is increased road safety. According to ISTAT [1], in the first half of 2024 alone, there were 1,429 fatalities due to car accidents, an increase compared to the same period in 2023. Higher levels of automation can help mitigate dangerous driving behaviors, reducing the number of crashes and injuries.

Beyond safety, autonomous vehicles can also alleviate traffic congestion in major cities. Fewer accidents and smoother traffic flow mean reduced commuting times and lower stress levels for drivers. Additionally, by minimizing unnecessary idling, self-driving cars can contribute to lower greenhouse gas emissions, benefiting the environment.

Autonomous driving technology is also transforming industries such as agriculture and logistics. In farming, self-driving tractors and harvesters can operate with minimal human intervention, reducing labor costs while increasing efficiency. Similarly, autonomous trucks and delivery vehicles can lower transportation costs for food and materials, ensuring more efficient supply chains and reducing dependence on human drivers.

In areas with limited public transportation, fleets of driverless cars could provide mobility solutions for people without access to a personal vehicle. This could enhance independence for young people, the elderly, and individuals with disabilities, ensuring greater accessibility and inclusivity in transportation.

1.1 Formula Student

In this work, the problem of autonomous driving will be addressed in a controlled environment, simplified compared to the general-purpose autonomous driving described above. Specifically, our focus will be on the Formula SAE competition, a prestigious student design challenge organized by Society of Automotive Engineers (SAE) International.

Formula SAE tasks university undergraduate and graduate students with conceiving, designing, fabricating, developing, and competing with small, formula-style vehicles (Fig. 1.1). Teams are composed exclusively of university students from diverse academic backgrounds, fostering interdisciplinary collaboration across various fields worldwide. Success in the competition requires expertise in mechanical, electrical, and software engineering, along with economics, management, and marketing.

As stated on the official Formula SAE regulations [2]:

> *Teams will act as an engineering firm that will conceive, design, fabricate, test, develop and demonstrate a prototype vehicle.*
> *The vehicle should have high performance and be sufficiently durable to successfully complete all the events at the Formula SAE competitions.*
> *Additional design factors include: aesthetics, cost, ergonomics, maintainability, and manufacturability.*
> *Each design will be judged and evaluated against other competing designs in a series of Static and Dynamic events to determine the vehicle that best meets the design goals and may be profitably built and marketed.*

According to the official 2025 Rules of FS East, the competition is divided into three separate categories:

1. Internal Combustion Engine Vehicle (CV);
2. Electric Vehicle (EV);
3. Driverless Vehicle (DV).

Specific regulations concerning both the car and the track have been established to ensure the safety of all participants. According to the Formula Student Germany Competition Handbook 2025 [4], tracks will be marked with colored cones (see figure 1.2) :

- The left borders of the track are marked with small blue cones.
- The right borders of the track are marked with small yellow cones.

Figure 1.1: Example of the AMZ [3] Formula Student Driverless car, built by ETH Zurich students.

- Exit and entry lanes are marked with small orange cones.
- Big orange cones will be placed before and after start, finish and timekeeping lines.
- the maximum distance between two cones in driving direction is 5 m. In corners, the distance between the cones is smaller for a better indication.
- The start, finish and timekeeping lines as well as keep out zones around the timekeeping equipment are marked with red, orange or pink paint.
- Additionally for skid pad and trackdrive, track limit lines on either side of the track and entry/exit lanes may be marked with yellow, green or white paint.
- Timekeeping equipment may be surrounded by additional cones outside of the track boundary

Cones along the driving direction will be no more than 5 meters apart, and specific zones will be marked with colored paint.

Before the competition begins, each vehicle undergoes a Technical Inspection to ensure rule compliance. For the autonomous system, this includes providing data sheets for all perception sensors and documentation certifying their compliance with local legislation.
The Remote Emergency System, a remote-controlled module on the vehicles that activates emergency behaviors, is also verified. This system is activated if:

- The autonomous vehicle seems to be out of control;
- the vehicle gets visibly damaged (mechanically or electrically);

Figure 1.2: Official Driverless competition cones.

- The minimum average speed during the track drive event is not respected (2.5 m/s for the first three laps, and 3.5 m/s for the following ones);
- The presence on the track of something/someone not allowed to be there.

Other inspections include a tilt test, to check the wheels' contact with the ground and the presence of fluid leaks. A rain test is performed to ensure the safety of the electrical system in case of adverse weather conditions.

If the Technical Inspection has been successfully completed, the prototype is then judged according to two types of tests: static events and dynamic events.

- Static events
  - *Business Plan Presentation*: teams must deliver a comprehensive business model to convince potential investors or partners of the project's profitability.
  - *Cost and Manufacturing*: this event evaluates the team's ability to make cost-effective decisions during manufacturing, including make-or-buy choices. Teams must submit detailed Cost Report Documents for every material and component used in the vehicle.
  - *Engineering Design*: in this event teams explain their design choices and highlight features, concepts, or methods that add value to the vehicle
- Dynamic events
  - *Skidpad*: an eight-shaped track, figure 1.3, consisting of two pairs of concentric circles, has to be cleared twice. This means that the vehicle must perform two laps around each circle, as well as autonomously enter and exit from the test area.

- *Acceleration*: the track here is a simple straight line with specific length and width, figure 1.4, delimited by a starting and a finish line. The event consists of an acceleration test from a standing start. The primary metric that is used for evaluating this event is the taken time to complete the track.
- *Autocross*: the autocross track does not have a fixed layout, but it is designed adhering to predefined constraints on the length, width, and number of curves and straight sections. The goal is to complete two runs consisting of one lap each, in the minimum possible time. The evaluation is obtained according to the completion of the aforementioned laps and to the comparison between the elapsed time and the maximum allowed time.
- *Trackdrive*: this event is only for the driverless cup and consists of a closed loop circuit, shorter than the ones described above, designed according to some constraints. Teams have to complete ten laps, counted by the vehicle itself: this means that there will be no explicit signals or indications on the track, implying that the autonomous system must be able to keep track of the completed laps. Points received in this test depend on the number of completed laps, and on the individual elapsed time.

During dynamic events, penalties can be assessed in many different cases. The most relevant for the self-driving category are knocking over cones, going outside the track with all four wheels and not re-entering within a certain time, or stopping in an unsafe manner, meaning outside of the specified area, for example.

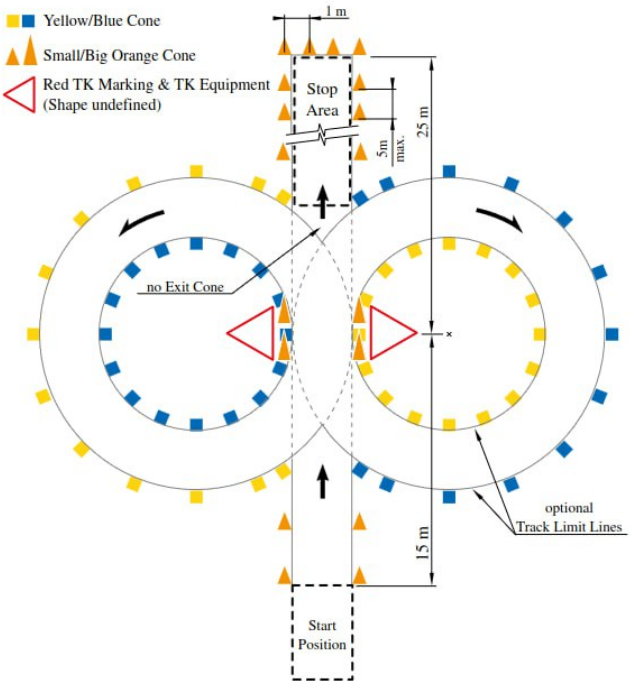


Figure 1.3: Skidpad track configuration.

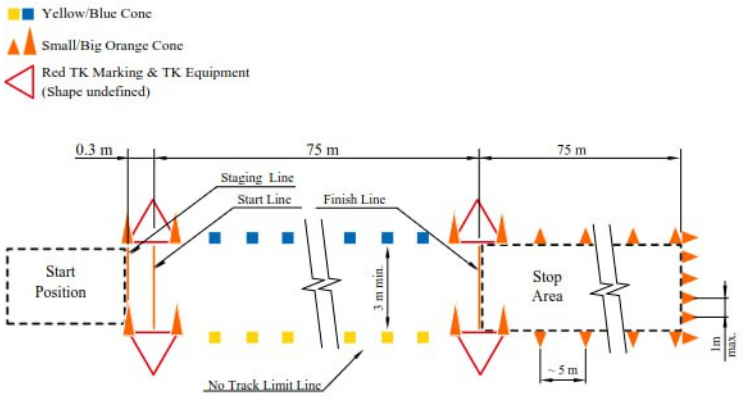


Figure 1.4: Acceleration track configuration.

Race UP [5] is the Formula SAE team of the University of Padua that every year designs, manufactures and develops two formula-style vehicles: the electric one “SG” and the internal combustion one “MG”. It was founded by the faculty advisor Professor Giovanni Meneghetti in 2006 and its history began with the design and development of the internal combustion vehicle “MG” (in honor of Professor Giovanni Meneghetti). In 2014, the team started conceiving also the electric vehicle which made its debut in 2016 with the so called “Origin-e”, then named “SGe” in honor of the technical supervisor Stefano Giacometti.

Figure 1.5: Race UP team logo.

1.2 Problem statement

Designing a self-driving car, whether for urban streets or a Formula Student track, requires careful consideration of both hardware and software components. Safety and reliability must be the top priorities when selecting sensors and defining the system architecture, ensuring robust performance in diverse and dynamic environments.

The perception module serves as the car’s sensory system, using sensors such as cameras, LiDAR, and radar to gather information about the environment. These sensors detect obstacles and map the road’s structure, much like human senses perceive the world. The collected data must then be processed and integrated with additional information from onboard sensors that monitor the car’s state, such as IMU and encoders. This enables the vehicle to construct a local map of its surroundings and accurately determine its position within it using Simultaneous Localization and Mapping (SLAM) algorithms [6]. A local map can also be combined with a broader global map, allowing the car to navigate efficiently and plan the optimal trajectory for safe movement.

Finally, precise and reliable control mechanisms are essential to ensure the vehicle follows the planned trajectory accurately.

In the Formula Student environment, the driving scenario is simplified compared to general road conditions, as there are almost no dynamic obstacles, and the track is defined solely by

colored cones, as specified in the regulations.

However, racing scenarios impose strict performance constraints, requiring a high-speed computing system that is both lightweight and energy-efficient. Additionally, sensor selection, placement, and mounting influence aerodynamics and weight distribution, making interdisciplinary collaboration essential. These factors must be carefully discussed with team members from various departments to ensure an optimal balance between perception accuracy and vehicle performance.

For the Race UP Driverless project, the goal is to compete in the Driverless competition for the first time in 2025, with the primary objective of successfully completing at least the Driverless Acceleration event. The team does not have a dedicated driverless car, but the sensors and actuation setup will be mounted on the electric formula-style vehicle.

Currently, the driverless setup has been tested on a sensor-equipped test car, which was able to autonomously navigate a few curves at low speed. This demonstrates the functionality of the control, odometry, and perception systems, an important milestone and a solid foundation for further improvements.

This achievement is the result of three years of research and development by the team. Initially, the system and algorithms were developed in simulation, and over the past two years, the work partially transitioned to a physical test vehicle. The project originally relied on stereo cameras for cone detection, but this year, the team switched to LiDAR-based perception due to its significantly higher depth accuracy, which was not achievable with the previous camera setup. However, this shift comes with certain drawbacks.

LiDAR point clouds require ground removal and segmentation to identify clusters, both of which are computationally expensive tasks that increase latency in cone perception. Additionally, when cones are far away, LiDAR provides only a few keypoints per object, making it difficult to reliably classify whether a given cluster corresponds to a cone. Cameras, on the other hand, enable easier cone detection and allow for the use of faster algorithms and deep learning models.

LiDAR also struggles with accurate color classification, relying solely on ray intensities, whereas cameras naturally capture full-color information, making it easier to distinguish between different cones. Furthermore, camera images contain significantly denser visual information compared to sparse LiDAR point clouds, leading to a more detailed understanding of the scene.

Beyond these technical considerations, cost is another key factor. High-resolution LiDAR systems are extremely expensive, whereas stereo cameras offer a cost-effective alternative without significantly compromising performance in structured environments.

Moreover, having two independent systems, one based on LiDAR and the other on stereo

vision, providing cone positions to the SLAM algorithm enhances reliability, a crucial aspect of a driverless car's perception system.

1.3 Thesis objectives and outline

This thesis aims to enhance the stereo camera SLAM front-end developed by the RaceUP team by improving both speed and accuracy.

To achieve this, a new stereo camera setup will replace the existing Bumblebee system, significantly improving both image quality and depth estimation. Additionally, the current YOLOv7 model for cone detection and classification, which struggles with accurately distinguishing cone size and color, will be replaced by a fine-tuned YOLO11 pose estimation model. This new model not only enhances detection and classification accuracy but also predicts the vertices of cones.

The detected cone peaks will then be used to triangulate the cone's position in three-dimensional space. The stereo triangulation error will be modeled with a covariance matrix, allowing the cone's 3D position to be associated with an uncertainty estimate. This approach addresses a limitation in the current implementation, where only cones near the camera are considered, and their positions are assumed to be accurate without considering uncertainty.

Chapter 2 reviews scientific literature and state-of-the-art solutions adopted by other Formula Student teams in autonomous racing scenarios. Particular attention is given to different sensor configurations and approaches to the SLAM front-end problem.

Chapter 3 introduces the SLAM problem, with a specific focus on the graph-based SLAM currently used by Race UP. The key requirements for an effective stereo vision front-end are outlined, along with the necessary inputs for the SLAM back-end. A theoretical background on stereo camera systems is also provided to clarify the challenges and solutions explored in this work.

Chapter 4 presents the transition from the previous sensor setup to the newly developed system, detailing the installation and configuration of the FLIR Blackfly S cameras in stereo mode.

Building on this, chapter 5 introduces the fine-tuned YOLO11-Pose model, explaining its selection, training process, and dataset creation. This new model not only improves speed and classification accuracy but also predicts the position of the peak of the cone.

Given that 3D cone position estimates are inherently affected by errors and noise, chapter 6 proposes a method for modeling these uncertainties using a covariance matrix, which is then associated with the estimated cone coordinates.

The implementation of the entire pipeline, from image acquisition to processing in ROS [7],

is detailed in chapter 7. Finally, the results of the proposed system are evaluated in chapter 8, where improvements in model accuracy and cone position estimation are analyzed. A comparative study between the new and old stereo camera setups highlights the enhancements achieved and justifies the advantages of the new approach.

Chapter 2

Related work

The competitive nature of Formula Student makes it challenging to find scientific literature specific to this application. Most teams are reluctant to publish their work as formal research or share their findings with competitors, resulting in limited formal publications in this field. However, there are alternative ways to gain valuable insights:

- Attending competitions and observing the cars in action on the track. Direct observation provides a firsthand look at the practical implementation of various technologies and strategies used by different teams.
- Reviewing university theses related to Formula Student driverless projects. While these works can offer useful insights into different approaches, they are not formally published or peer-reviewed, so their findings should be critically evaluated.
- Consulting competition judges can be particularly beneficial. With their broad experience in the field, they can provide valuable feedback on system architecture choices, suggest improvements, and highlight potential weaknesses.

Despite the difficulty of finding publications, a small number of papers on previous implementations from other teams are available. These resources are particularly valuable for analyzing different approaches, understanding their strengths and weaknesses, and comparing their results with the current Race UP implementation.

In [8], the AMZ Driverless team from ETH Zurich presents its fully autonomous racing platform. Their perception pipeline supports LiDAR-only, vision-only, or sensor fusion approaches. The primary sensor modalities include a 3D LiDAR sensor mounted on the front wing and a camera-based system consisting of three CMOS cameras with global shutters in a stereo and mono setup. The stereo configuration is optimized for detecting nearby cones, while the monocular camera focuses on identifying cones at greater distances. LiDAR data is processed with ground removal, followed by cone detection through clustering and classification

using intensity values. In the vision-based pipeline, YOLOv5 is employed for cone detection in images, while depth estimation is performed by matching SIFT features. For distant cones the monocular camera is used, a custom model was trained to detect keypoints on the cone and then PnP (Perspective-n-Point) is used to determine the pose since the cone shape and sizes are known.

In [9], ETH Zurich further refines its perception system, incorporating two 32-channel rotating LiDARs and two global shutter monocular cameras. The LiDAR-based pipeline remains unchanged, but camera-based depth estimation is improved by leveraging bounding box sizes and the known cone dimensions. When both LiDAR and camera data are available, a fusion strategy enhances accuracy: the LiDAR-based cone detection points are projected onto the image plane, and the corresponding RGB patches are processed by a customized CNN to predict cone colors.

[10] introduces RekTNet, a residual neural network that detects 7 keypoints on a cone's image, and utilizes the same PnP-based approach for depth estimation. But differently from [8] the monocular camera pipeline is used for short-range detection, leveraging the fact that the pixel-space location of a landmark on a globally flat surface is more sensitive to pose variations at close range. This improves localization accuracy. Additionally, stereo estimation in [10] is performed using Semi-Global Matching (SGM) to compute disparity by matching cone pixels in both images. The study also addresses camera acquisition synchronization challenges.

A different multi-modal approach is adopted by the Bombay Driverless team in [11], employing a three-tiered depth estimation strategy. For cones detected by LiDAR, a LiDAR-camera fusion pipeline is used, projecting detected LiDAR points onto the image for classification and outlier removal. If a cone is not detected by LiDAR, depth estimation relies solely on camera-based methods, using cone size as a depth cue. When cones are fully visible and upright, a monocular pipeline based on RekTNet [10] is used for keypoint-based PnP depth estimation. A stereo-vision pipeline is employed instead for occluded or misaligned cones.

In [12], the KIT Driverless team employs a redundant sensor suite consisting of three cameras and four LiDARs to ensure robust environment perception. This multi-sensor approach mitigates the weaknesses of individual sensors, improving overall system reliability. Their LiDAR-based pipeline estimates candidate landmarks, which are stored in a map. These landmarks are later validated using camera projections: each 2D landmark is mapped onto the image, where the system verifies whether they correspond to cones. If confirmed as valid, the cones are classified accordingly.

Moreover, there are other two theses related to the work of the Race UP Driverless team.

In [13], Tonin developed the first version of the Simultaneous Localization and Mapping (SLAM) module for the team, primarily working within the EUFS simulation platform [14].

The graph-based SLAM system, currently in use, was implemented using the g2o library. Two different correction approaches were tested: a global-only optimization method and a local, incremental approach. Additionally, she analyzed the nearest-neighbor data association algorithm to improve SLAM robustness against sensor noise and outliers. The SLAM front-end incorporated both camera and LiDAR inputs: the visual cone detection pipeline relied on a fine-tuned version of YOLOv7 to extract bounding boxes of the cones in stereo camera images, while the LiDAR-based detection pipeline utilized a classical segmentation approach combined with Euclidean clustering to identify cone shapes in the point cloud.

Building on this foundation, Khalili [15], a member of the Race UP driverless team, transitioned Tonin's work from simulation to real-world application. The first step involved acquiring a comprehensive real-world dataset using a properly sensorized prototype, with the sensor setup closely replicating the configuration used in simulation. Unlike the previous approach, which integrated both camera and LiDAR data, Khalili's work focused solely on visual perception, removing LiDAR from the SLAM front-end. The YOLOv7-based detection pipeline was adapted to process real-world data, ensuring reliable cone detection under real operating conditions. Additionally, the motion model was updated to incorporate an anti-Ackerman steering system.

Chapter 3

Theoretical background

3.1 SLAM

To navigate safely, confidently, and efficiently in an unknown environment, a self-driving car must solve a fundamental problem in robotics: SLAM (Simultaneous Localization and Mapping). SLAM is the problem of building a map of an unknown environment while at the same time navigating the environment using that map.

To achieve these goals, the robot relies on its sensors to observe the environment and extract meaningful information, such as landmarks, which are used both to construct the map and as reference points for localization. To track its position relative to a starting point, the robot employs odometry, a technique that estimates changes in position over time by analyzing data from sensors such as wheel encoders and IMUs (Inertial Measurement Unit). However, since all sensor measurements are inherently noisy, errors accumulate over time, leading to uncertainty in both the estimated trajectory and the generated map. To address this, the robot represents its trajectory and the environment in a probabilistic way.

To mathematically describe SLAM [16], we define:

- the robot pose at time t by x_t
- the path as $X_T = \{x_0, x_1, x_2, ..., x_T\}$ where T might also be infinite
- the robot's initial location by x_0 and x_0 is assumed to be known, while the other locations are not.
- an estimate of the robot motion between time $t-1$ and time t as u_t.
- the sequence of the robot relative motions $U_T = \{u_0, u_1, u_2, ..., u_T\}$
- the true map as $M = \{m_0, m_1, m_2, ..., m_{n-1}\}$ where m_i are vectors representing the positions of the landmarks

- the sequence of estimated landmark observations $Z_T = \{z_0, z_1, z_2, ..., z_T\}$ in the sensor reference frame attached to the robot. For example, in our case a vector representing the coordinates of a cone in the camera or LiDAR reference frame.

According to this terminology, we can now define SLAM as the problem of recovering a model of the map M and the robot path X_T from the odometry U_T and observations Z_T.

In the literature, there is a distinction between the full SLAM problem and the online SLAM problem.

- The full SLAM problem consists in estimating the joint posterior probability over X_T and M from the data, $p(X_T, M|Z_T, U_T)$

- The online SLAM problem, conversely, consists in estimating the joint posterior over x_t and M from the data, $p(x_t, M|Z_T, U_T)$

Therefore, the full SLAM problem tries to recover the entire robot path X_T, while the online SLAM problem tries to estimate only the current robot pose x_t.

In order to solve the SLAM problem, we need to know

- the probabilistic motion model, $p(x_t|x_{t-1},u_t)$ that represents the probability that the robot pose is x_t given the robot previous pose x_{t-1} and control input u_t

- the probabilistic measurement model, $p(z_t|x_t,M)$ that represents the probability of measuring z_t given the known map M and assuming that the robot takes the observation at location x_t.

To summarize, the vehicle state and the world map will be continuously updated with the motion: from the new pose just reached, new landmarks are perceived through sensors and every observation will be tested for association with already mapped ones.

Over the past two decades, three main paradigms have been developed to address the SLAM problem: EKF-SLAM, particle filter SLAM, and graph-based SLAM. The Race UP team has chosen the graph-based SLAM approach, which is briefly described in the following, using [17] as a reference.

A graph associated with the SLAM problem is created: the nodes of this graph are the robot poses in X_T and the landmarks in the map M. Each edge in the graph corresponds to an event:

- a motion event generates an edge between two robot poses,

- a measurement event creates a link between a pose and a feature in the map

Each edge in the graph corresponds to a nonlinear constraint. Adding such a constraint to the graph is trivial since it involves no significant computation. Figure 3.1 illustrates an example of a constructed graph, highlighting the difference between the true and estimated robot poses and landmarks.

First, consider a measurement z_t. This measurement provides information between the location of the landmark j and the robot pose x_t at time t. This information is mapped into a constraint between x_t and m_j. This edge can be considered as a (possibly degenerate) "spring" in a spring-mass model. The constraint is of the type:

$$(z_t - h(x_t, m_j))^T Q_t^{-1} (z_t - h(x_t, m_j)) \tag{3.1}$$

Here h is the measurement function, and Q_t is the covariance of the measurement noise.

Now consider robot motion. The control u_t provides information about the relative value of the robot pose at time $t-1$ and the pose at time t. Again, this information induces a constraint in the graph, which will be of the form:

$$(x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \tag{3.2}$$

Here g is the kinematic motion model of the robot, and R_t is the covariance of the motion noise.

The sum of all constraints results in a nonlinear least squares problem aimed at optimizing the path X_T and the map M.

In the domain of simultaneous localization and mapping (SLAM), two primary components are commonly distinguished: the front-end and the back-end.

- The front-end handles raw sensor data processing (cameras, LiDAR, IMU) and transforms it into an intermediate representation, for example constraints for an optimization problem.

- The back-end takes this intermediate representation from the front-end and does the task of solving the optimization problem or state estimation problem.

In this project, stereo cameras are used to perceive the environment and map the track. The front-end's role is to process incoming stereo images, detect and classify cones, and estimate their positions along with an associated covariance matrix that represents the uncertainty of the observations. It is a crucial component because it directly influences the quality and reliability of the features and measurements fed into the SLAM system.

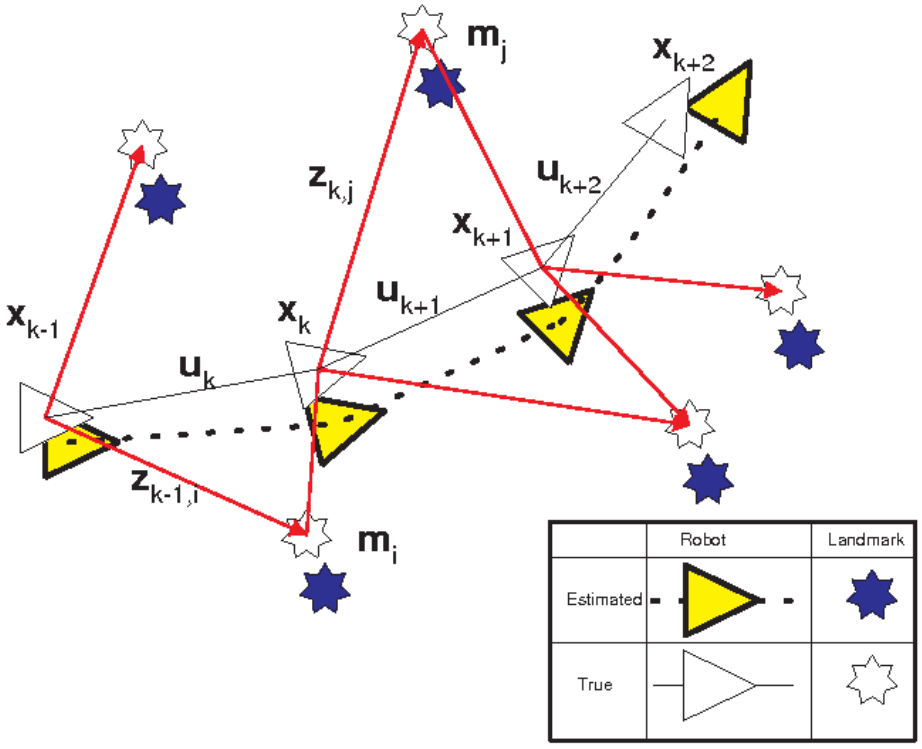


Figure 3.1: Graph-based SLAM example, showing the nodes and the constraints connecting them. The difference between true and measured poses shows why optimization is required. (Source [18])

3.2 Stereo camera

A stereo camera is a type of camera system that mimics human binocular vision by using two lenses to capture images from slightly different angles. This setup allows the camera to estimate depth information from the scene, which is crucial for applications like robotics, autonomous vehicles and 3D reconstruction.

Before describing how stereo cameras work, it is important to understand the mathematical relationship between the coordinates of a point in three-dimensional space and its projection onto the image plane. This relationship is well described by the pinhole camera model.

The geometry related to a pinhole camera is illustrated in the figure 3.2:

- the reference frame XYZ is the 3D camera reference frame, centered in the optical center O, where the camera aperture is located. Z is the optical axis (or principal axis)
- the reference frame xy is the 2D image reference frame, centered in o which is the principal point (or image center)
- a 3D point P is projected on the 2D image plane into the 2D point p
- f is the focal length, the distance between the camera's optical center O and the image plane

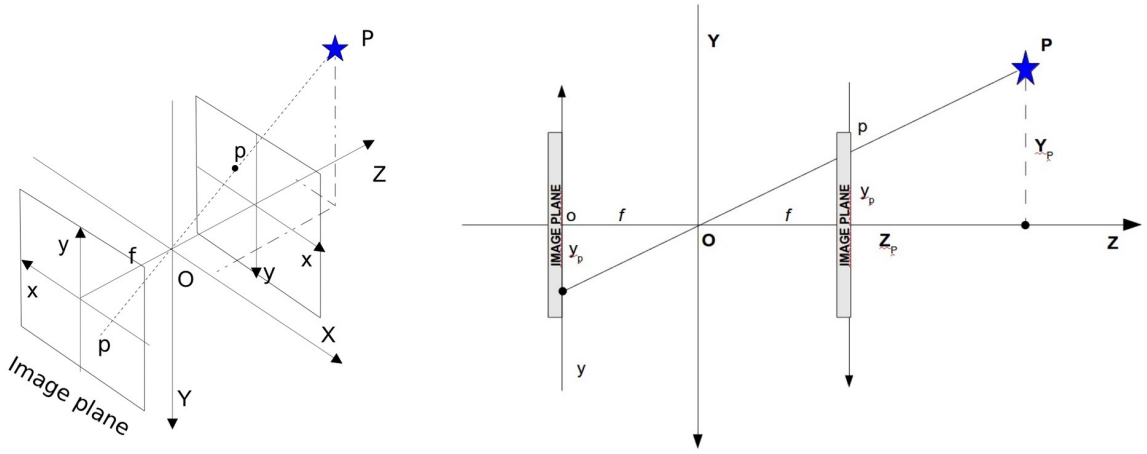


Figure 3.2: Pinhole camera model

If we define the focal length in pixels as $\alpha_u = \frac{f}{w_p}$ and $\alpha_v = \frac{f}{h_p}$, where w_p and h_p are the pixel width and height in meters, the projection $p = (x_p, y_p)$ of a 3D point $P = (X_P, Y_P, Z_P)$ on the image plane is given by:

$$x_p = u_c + \frac{X_P \alpha_u}{Z_P} \tag{3.3}$$

$$y_p = v_c + \frac{Y_P \alpha_v}{Z_P} \tag{3.4}$$

where u_c and v_c are the coordinates of the principal point in pixels.
u_c, v_c, α_u and α_v are the intrinsic parameters of the camera.

In general, the coordinates of a 3D point are not expressed in the current camera frame. Therefore, before projecting a point onto the image plane, we need to transform its coordinates from the world (or object) frame to the camera's coordinate system. To perform this transformation, the extrinsic parameters of the camera, which define its position and orientation relative to the world coordinate system, are required.

Both intrinsic and extrinsic parameters can be obtained through a process called camera calibration, which involves capturing images of a known calibration pattern, such as a checkerboard, from various angles and distances and processing them with specialized software. Additionally, this process estimates lens distortion parameters, which account for optical distortions introduced by the camera lens. To ensure an accurate image representation, the acquired images must be undistorted before further processing.

Two calibrated cameras mounted on a rigid support form a stereo rig or stereo pair. Before using the system to estimate the depth of points in a scene, stereo calibration is required to determine the precise rigid transformation between the two cameras.

Given a point in one image, the corresponding point in the second image must lie on a specific line called the epipolar line. In a general stereo setup, the two cameras may not be perfectly

aligned, causing epipolar lines to be slanted or curved. This misalignment makes it computationally expensive to search for corresponding points in the two images. Stereo rectification applies a transformation to a pair of stereo images so that corresponding points lie on the same row, reducing the search to a 1D horizontal scan instead of a full 2D search.

Once the two cameras are calibrated and the acquired images are rectified, the 3D coordinates of a point P can be computed using stereo triangulation. If (x_l, y) and (x_r, y) are the projections of P in the left and right images, respectively, P coordinates can be determined using the following formulas:

$$Z = \frac{b}{d} f \tag{3.5}$$

$$X_l = \frac{b\, x_l}{d} \quad X_r = \frac{b\, x_r}{d} \tag{3.6}$$

$$Y_l = Y_r = \frac{b\, y}{d} \tag{3.7}$$

where:

- b is the baseline, the distance between the two lenses (or optical centers) of the camera pair.
- $d = x_l - x_r$ is the disparity
- $P_l = (X_l, Y_l, Z_l)$ is the 3D point in the left camera frame and $P_r = (X_r, Y_r, Z_r)$ is the same point in the right camera frame.

In this work, OpenCV [19] is used for image processing, and its convention for stereo camera reference frames is adopted. The reference frames are illustrated in the figure 3.3

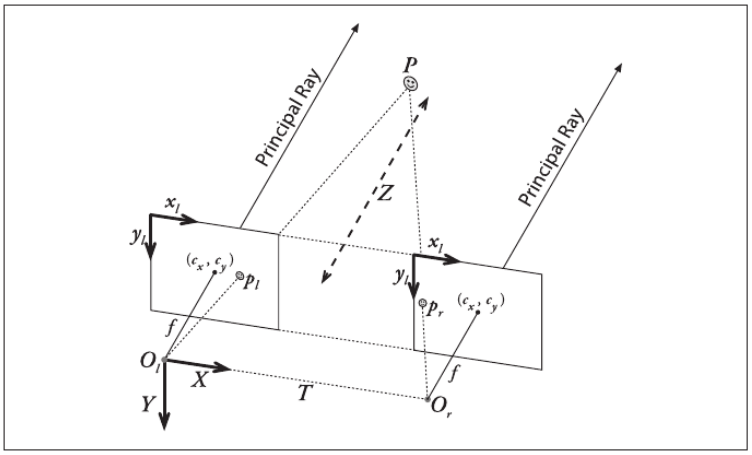


Figure 3.3: Stereo coordinate system used by OpenCV for undistorted rectified cameras, T is the baseline

Moreover, OpenCV [20] does not directly use the previously mentioned formulas to perform triangulation. Instead, it utilizes a disparity-to-depth Q matrix for each of the two cameras, for the left camera:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{(c_x - c'_x)}{T_x} \end{bmatrix}$$

where T_x is the baseline, c_x and c_y are the coordinates of the principal point of the left camera and c'_x is the x coordinate of the principal point of the right camera.

- If the principal rays intersect at infinity, then $c_x = c_x{}'$ and the term in the lower right corner of the Q matrix is 0.
- If the cameras are converging toward each other (i.e., slightly "cross-eyed"), zero disparity occurs at a finite distance and $c_x \neq c'_x$. Images are rectified relative to the principal points (c_x, c_y) and thus measurements must also be relative to these positions: $\tilde{x}^r = x^r - c_x^{right}$ and $\tilde{x}^l = x^l - c_x^{left}$. When computing the disparity, if $c_x = c'_x$, the additional term disappears. However, if $c_x \neq c'_x$, a correction factor must be applied to account for the shift in the optical centers.

Given a two-dimensional homogeneous point $p = (x,y,1)$ and its associated disparity d, we can triangulate the point into three dimensions using:

$$Q * \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

The 3D coordinates are then $(X/W, Y/W, Z/W)$.

Chapter 4

Car setup evolution

The following sections describe the evolution of the car's sensor setup. These setups have not yet been tested in a real competitive event but only during Race UP test days. Throughout the season, the electric vehicle team organizes full test days to evaluate the mechanical and aerodynamic performance of the new race car prototype. These occasions are extremely valuable for the driverless team, as they allow them to mount sensors on a real car and collect data from the moving vehicle to evaluate the developed algorithms. Even though the car is not driving autonomously, these tests serve as a chance to assess the accuracy of odometry or the precision of cone detection.

Between test sessions, the driverless team works on a test mule, a simplified version of the vehicle designed solely for development and testing. Unlike the competition car, the test mule does not need to comply with strict regulations, allowing for rapid prototyping and experimentation. It enables real-world validation of the car's systems and components, helping to identify and resolve potential issues early in the development process before they are integrated into the more complex and costly competition vehicle. Additionally, the test mule provides a valuable opportunity for the team to refine their testing and debugging procedures.

4.1 Previous Setup

In the initial iteration of the driverless setup, the selected sensors were:

- Velodyne VLP16 16-channels LIDAR
- Bumblebee2 RGB stereo camera
- XSens MTi Inertial Measurements Unit (IMU)

The optimal sensor placement (Fig. 4.1) was determined to be on the car's main hoop, above the driver's head, using a custom-designed support. The setup was arranged as follows:

- The plate holding the stereo camera and IMU was positioned parallel to the ground.
- The LiDAR was tilted 5° downward to maximize its field of view for cone detection.

This placement was chosen to ensure that all sensors remained within the car's frame, did not obstruct the driver's view, and were mounted on structurally robust components. Additionally, it minimized vibrations while maximizing the field of view for both the LiDAR and stereo camera.

Figure 4.1: Sensor placement for the first data acquisition

The Bumblebee2 stereo camera (Fig, 4.2) features two Sony 1/3" CCD ICX204 sensors with 4.65µm square pixels, a 12 cm baseline, and an output resolution of 1024 × 768 pixels. It supports two lens options: a 3.8 mm focal length with a 70° horizontal field of view (HFOV) or a 6 mm focal length with a 50° HFOV.

Figure 4.2: Bumblebee2 stereo camera

A dataset was collected using this setup during a test day, with the sensor system mounted on the SG-e 05 single-seater. This vehicle features four electric motors, an encoder on each wheel,

and a steering angle sensor integrated into the steering system. Sensor measurements were recorded while a driver completed four laps around the track shown in Fig 4.3. This dataset was used for developing this work.

Figure 4.3: Track layout for first data acquisition

4.2 Current setup

The team was not satisfied with the previous LiDAR due to its low resolution and found the stereo camera setup inadequate because of its low resolution and excessive sensor convergence. As a result, the LiDAR was upgraded to an Ouster OS1 64-channel model, and a custom stereo camera system was built using two FLIR Blackfly S cameras (Fig. 4.4).

Figure 4.4: FLIR Blackfly S

Each camera is equipped with a Sony IMX430 1/1.7” CMOS sensor, a global shutter, and a C-mount for interchangeable lenses. A 4 mm wide-angle lens with a maximum aperture of f/2.0 was selected. The output resolution of each camera is 1616 × 1240.

The cameras are screwed on a metallic support with a baseline of 20 cm and then positioned on the main hoop above the driver’s head, since previous tests confirmed that this location provided a good field of view. Meanwhile, the LiDAR is relocated to a dedicated support on the car’s nose to have a better view of the cones and capture as many data points as possible.

The two FLIR Blackfly S cameras are connected together using a GPIO cable with a 6-pin Hirose HR10 connector. This setup designates the left camera as the master, allowing it to trigger image acquisition on the right camera by sending a sync pulse to a specific pin. As a result, both cameras capture images simultaneously, ensuring synchronized stereo acquisition.

The FLIR cameras were stereo-calibrated following this procedure:

- The cameras were securely installed on the test mule’s hoop, positioned in front of two rows of cones.
- Camera and lens settings were set to maximize the visibility of the cones
- A dataset of images featuring a traditional calibration checkerboard was captured.
- The images were processed using MATLAB’s Computer Vision Toolbox stereo calibration app [21]. After multiple iterations, during which low-quality images were removed, the final stereo and intrinsic camera parameters were obtained.

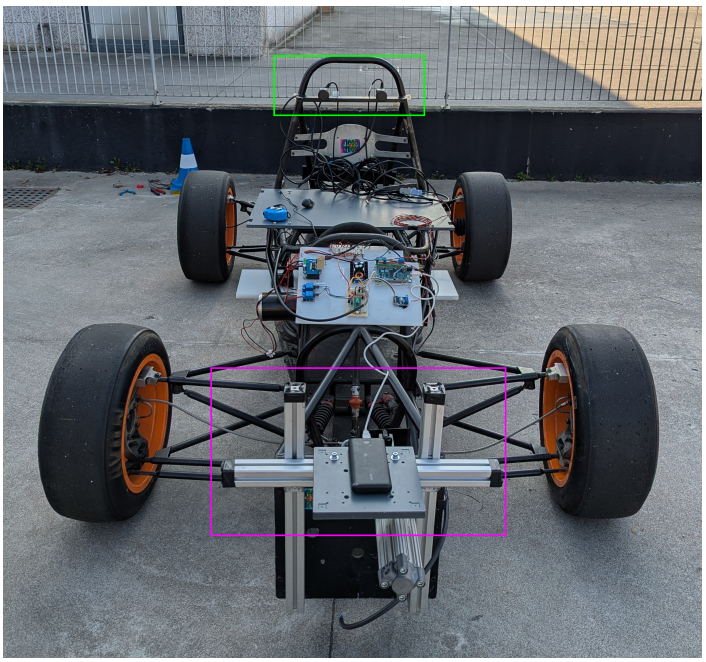
Since all the data processing is performed on the car, a powerful custom-built embedded computer has been developed to run all the software stack required by the car to move autonomously. The machine configuration is the following:

- CPU: 12 core AMD Ryzen 9
- GPU: GeForce RTX 4060
- RAM: 96GB DDR5
- Operating System: Linux Ubuntu 20.04

To ensure the system is waterproof, all components are enclosed in a sealed plastic box and water cooled to avoid the need of air intakes.

All the sensors have been mounted on a test mule, for the reasons explained above. Fig. 4.5 shows the sensors’ placement. The selected platform for the test mule is one of the first Race UP racing cars, MG-03, converted to an electric drivetrain, with a redesigned braking system to

incorporate an actuated braking and emergency brake system. Additionally, the steering system was upgraded by adding an actuator, a potentiometer to measure the steering angle, and an Arduino to control the actuator for achieving the desired angle. And finally, the back wheels are equipped with encoders for odometry estimation.



(a) In green the stereo cameras mount, in fuchsia the new LiDAR mount.

(b) FLIR Blackfly S cameras stereo setup

Figure 4.5: New sensors setup mounted on test mule

Chapter 5

Cone detection and keypoint extraction

As explained in Sec. 3.2, given a pair of matching image points in the two stereo images, the position of that point in 3D can be triangulated from their disparity. Therefore, a good pipeline to compute cone position is:

- Detect and classify cones in the two input images. The most effective approach for this task is using a deep learning model, as it offers greater robustness against variations in lighting conditions and image quality compared to traditional computer vision methods. Deep learning models can generalize better across different environments, ensuring more consistent and accurate cone detection.
- Every bounding box should be matched with the corresponding one on the other image.
- Detect and match keypoints or features within the cone bounding boxes in both images.
- Compute the depth of these keypoints to obtain the final 3D position of the cone.

Keypoint or feature extraction is a fundamental task in computer vision, with widely used techniques such as SIFT, SURF, ORB, and FAST. However, these methods have limitations, including sensitivity to noise, slow computation times, and strong dependence on image resolution. To address these challenges, in the following sections a deep learning-based keypoint extraction technique specifically designed for traffic cones is proposed.

5.1 Current stereo vision front-end implementation

As described in [15], the current Race UP stereo camera front-end utilizes the YOLOv7 [22] model fine-tuned on the FSOCO [23] dataset. The model was trained to detect and classify cones, but the cones class was never used due to its inaccuracy (Fig. 5.1). This represents a loss

of valuable information, which could be useful for data association or for further processing the detector's output.

Figure 5.1: Example of fine-tuned yolov7 inference on an image of the data acquisition described in 4.1. 3 big orange cones are classified as yellow and 4 blue cones are classified as small orange.

The team explored two different approaches for depth estimation.

The first approach treated the entire cone as a feature and used the center of the bounding box as the keypoint for depth estimation. At first glance, this method appeared easy to implement, robust against underexposure and overexposure, and computationally efficient. However, it had a fundamental flaw: the center of the bounding box in the left image does not necessarily correspond to the same physical point on the cone in the right image, leading to inaccurate disparity calculations.

To address this issue, the second approach considered the top point of each cone as the key feature to be matched between the two images. Cones within the bounding boxes were segmented by filtering pixels within specific color ranges, and the cone peak was identified as the highest central point in the segmentation mask. The depth was then computed using the disparity obtained from these cone peaks.

However, this solution also had drawbacks. Color thresholding is highly sensitive to lighting conditions, shadows, and reflections, which can significantly impact segmentation accuracy.

While the method may perform well in the dataset in consideration, its reliability under varying weather and lighting scenarios remains uncertain.

5.2 YOLO11 Segmentation

In this work, the first attempt to improve the Race UP team's previous approach focused on improving segmentation reliability by using a deep learning model capable of computing the panoptic segmentation of cones from a given input image.

After a thorough evaluation of the available models, YOLO11 Segmentation [24] was selected as it offered the best balance between accuracy and speed. An alternative option was Mask R-CNN [25], but the literature reports it as slower and less accurate. For instance, paper [26] highlights that even the older YOLOv8 outperforms Mask R-CNN in agricultural instance segmentation. While state-of-the-art models like SAM (Segment Anything) [27] provide superior accuracy, their computational demands make them impractical for real-time execution on the car's embedded system.

You Only Look Once (YOLO) [28] is a series of real-time object detection systems based on convolutional neural networks. First introduced by Joseph Redmon et al. in 2015, YOLO has undergone several iterations and improvements, becoming one of the most popular object detection frameworks.

The name "You Only Look Once" refers to the fact that the algorithm requires only one forward propagation pass through the neural network to make predictions, unlike previous region proposal-based techniques like R-CNN that require thousands for a single image. The network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

There are two parts to the YOLO series:

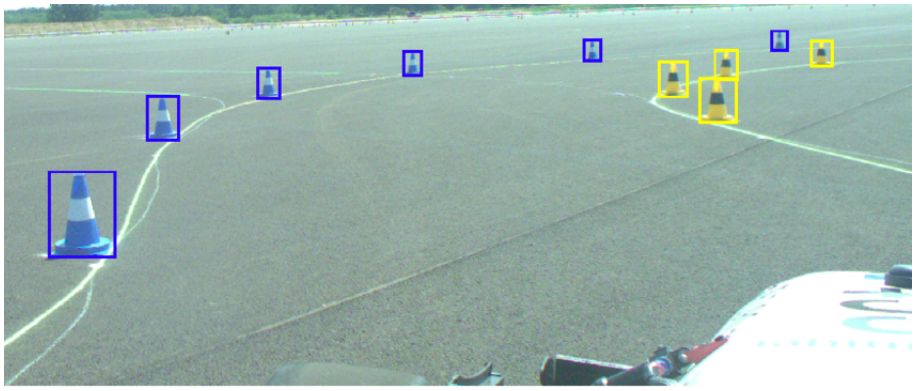
- The original part: YOLOv1, v2, and v3, developed by Joseph Redmon et al.
- Subsequent versions (v4 to 12) are not officially associated with the original YOLO authors but build upon their work. They have been developed by different researchers, further improving performance and introducing new features.

The latest available version of YOLO is YOLO12, but at the time this work began, YOLO11 was the most recent version. Even though YOLO was originally developed for object detection, the advancements by Ultralytics [29] have transformed YOLO11 into a highly versatile model, extending its capabilities across multiple computer vision tasks, including classification, instance segmentation, pose estimation, oriented object detection.

The model for each task is available in multiple sizes, including nano, small, medium, large, and extra-large, each with a different number of parameters. This allows for selecting the optimal trade-off between computational efficiency and model accuracy based on the specific application requirements.

The small `YOLO11-Seg` model was selected for fine-tuning on the FSOCO dataset [23], the same dataset used to fine-tune the YOLOv7 model adopted by the Race UP team. FSOCO is a public dataset of cone images annotated with both bounding boxes (Fig. 5.2a) and panoptic segmentation labels (Fig. 5.2b). However, not all images include both types of annotations. The larger subset, consisting of 11,572 images, contains only bounding box annotations with corresponding class labels, while a smaller subset of 1,517 images also includes mask annotations.

The dataset features images captured in various scenarios, from different cameras, and under diverse lighting conditions, ensuring a high degree of variance. It was created through the contributions of 40 Formula Student teams, who either provided images or helped annotate them.



(a) Cone bounding box labeling example

(b) Cone panoptic segmentation labeling example

Figure 5.2: Images from the FSOCO Dataset

`YOLO11-Seg` was fine-tuned for 100 epochs, starting from the default COCO weights. The training process included an early stopping mechanism with a patience of 20 epochs, meaning

training would stop if no improvement in loss was observed for 20 consecutive epochs. The 1517 images were split into train (85%), validation (10%) and test (5%) and the number of train images was increased with data augmentation. Unfortunately, the results obtained by applying the trained model to the test images collected during the data acquisition (described in 4.1) were not promising. Fig. 5.3 and 5.4 show the inference results on two images.

- If cones are small, the segmentation mask tends to cover the entire bounding box.
- If cones are big, the segmentation is qualitatively better but still requires post-processing to refine the results.

The unexpectedly bad segmentation is probably due to the size of the training dataset because even after changing training parameters and increasing the number of epochs, no improvements were obtained.

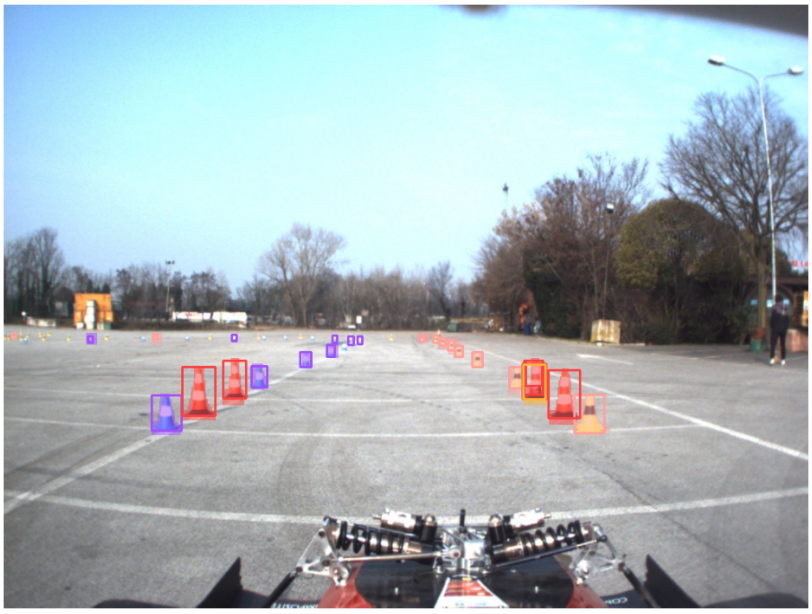


Figure 5.3: YOLO11 Segmentation result

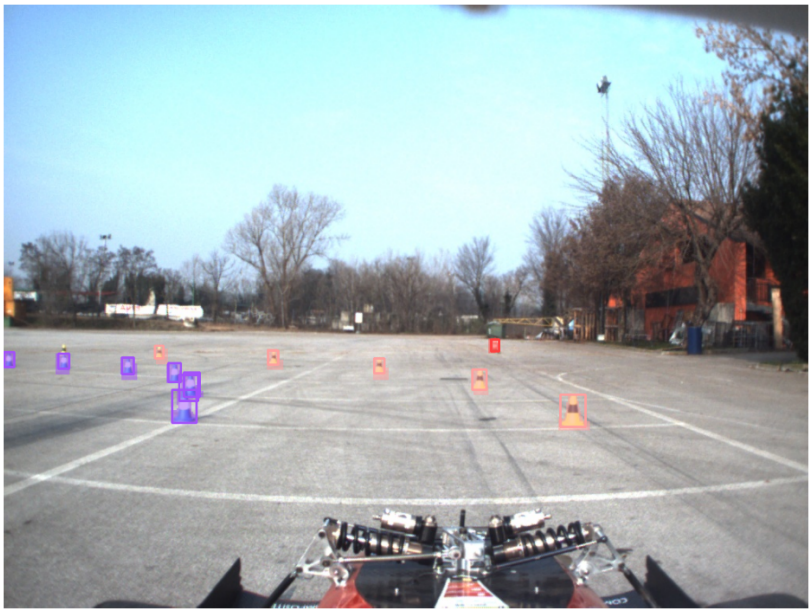


Figure 5.4: YOLO11 Segmentation result

5.3 YOLO11 Pose

With a more in-depth analysis, an increased number of images in the dataset and a model with a higher number of parameters, it would likely be possible to achieve better segmentation results.

However, after careful consideration, segmentation is used only to extract the cone's peak. An alternative approach could be to train a model capable of predicting not only the bounding box and class of the cone but also directly the keypoint corresponding to its peak. This idea was inspired by the `YOLO11-Pose` model, which is designed for pose estimation.

To train `YOLO11-Pose` for this task, it is essential to construct a high-quality dataset. The best approach would be to manually annotate the images, but this would have been extremely time-consuming. Moreover, without guarantees on the results, it could have turned into a waste of time. Therefore, an algorithm was developed to automatically generate a dataset by annotating the images with segmentation annotation in the FSOCO dataset.

The algorithm follows these steps for each image:

- First, the segmentation masks of the cones are extracted from the annotation file. To train a YOLO model, the dataset must follow a specific structure, where each image is associated with an annotation file. In the case of segmentation, this file must contain, for each line, the class ID of the cone followed by a list of normalized points (relative to the image dimensions) that define the polygon corresponding to the object's segmentation mask.

- Once the segmentation is obtained, the cone's principal axis is extracted using Principal

Component Analysis (PCA), selecting the axis with the highest variance. This step was necessary because not all images have a horizontal horizon (see Fig. 5.5).

- The highest point of the principal axis that belongs to the mask is selected as the cone's peak.
- The bounding box enclosing the cone is computed from the mask. It is saved in xywh format, x and y are the coordinates of the center of the bounding box, while w and h are the width and the height.
- Then, the coordinates of the mask points and the cone's peak are normalized based on the input image dimensions and saved in an annotation file. Each line of this file contains, in order: the class index (retrieved from the original annotation), the bounding box coordinates and the coordinates of the cone's peak.

The cone peaks obtained from this annotation algorithm are generally well-placed. While some may require minor manual adjustments, particularly for nearby cones (Fig. 5.6), these annotations provide a solid starting point.

Figure 5.5: Example of cone peak extraction with a not horizontal horizon. Cone segmentation is visible in green, in blue the principal axis and in red the final cone peak.

Once the dataset is obtained, training can proceed. The selected model is `YOLO11-Pose` in the small configuration (`YOLO11s-Pose`) to have a good trade-off between accuracy and inference time. A fine-tuning process of 100 epochs is performed with the recommended training parameters: batch size of 16 images, `AdamW` (Adaptive Moment Estimation with Weight Decay) optimizer with `learning rate` equal to 0.001111 and β_1 equal to 0.9. Mixed precision training was also recommended to speed up computation and reduce memory usage while maintaining

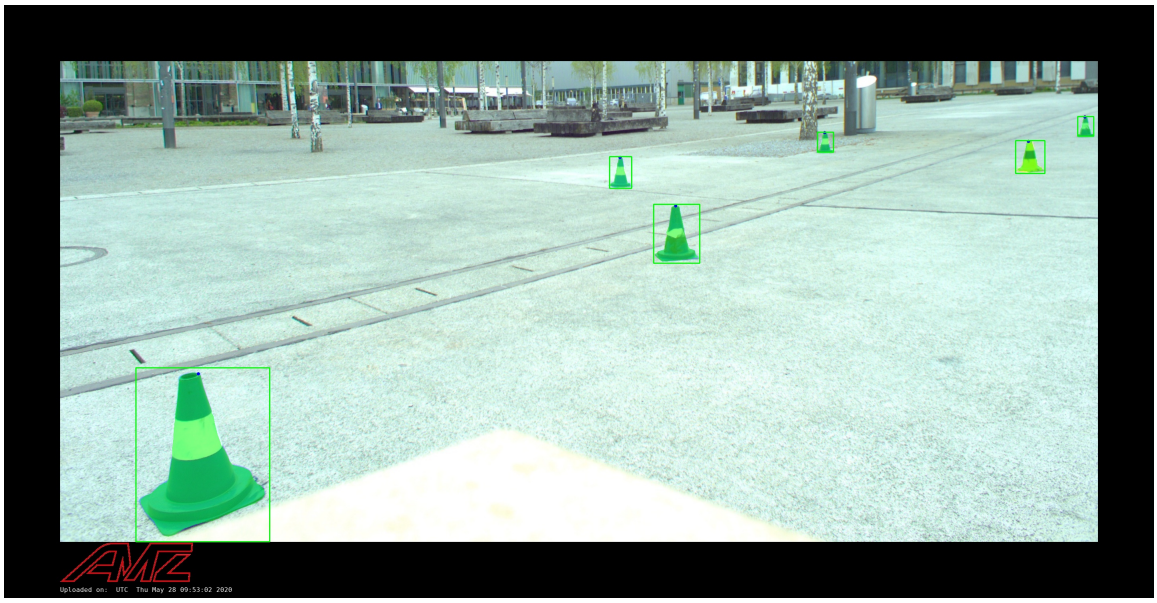


Figure 5.6: An example of cone peak extraction is shown, with the cone segmentation displayed in green and the final cone peak marked in red. Note that for the nearby cone, the peak point is not perfectly placed at the center of the peak.

model accuracy. It involves using different numerical precisions for different parts of the model and training process.

When evaluating the model on unseen images from the data acquisition, the results are consistently strong. As shown in Fig. 5.7, cone peaks are accurately detected, bounding boxes are correctly positioned and classified, and even small cones at a distance are successfully identified. Additionally, Fig. 5.8 highlights the model's robustness, demonstrating reliable performance even under challenging lighting conditions. In depth model evaluation can be found in Sec. 8.1.

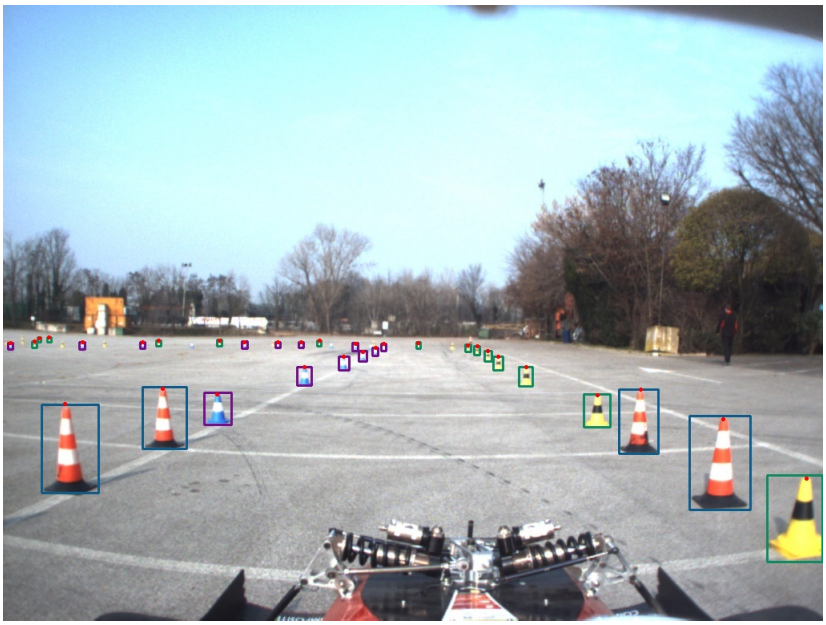


Figure 5.7: Result of the YOLO11-Pose fine-tuned on a data acquisition image.

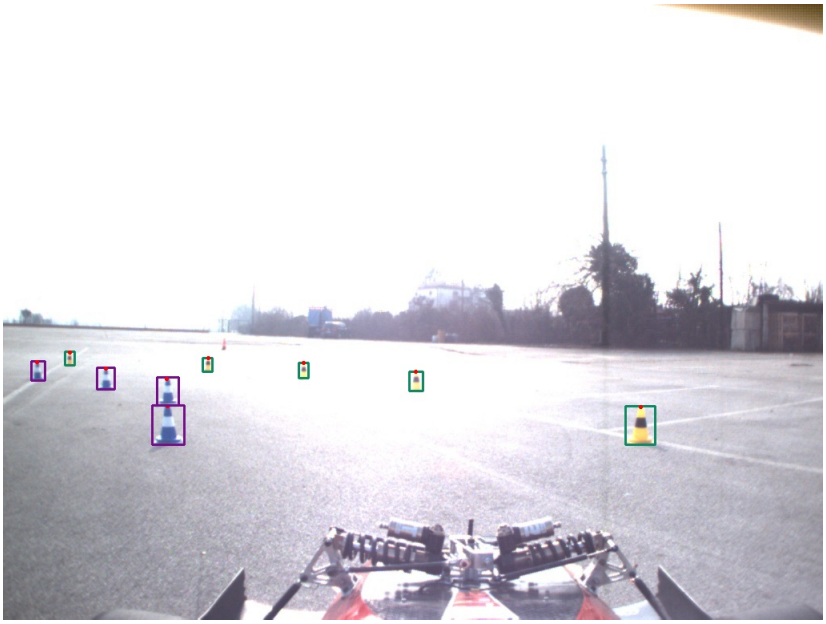


Figure 5.8: Result of the YOLO11-Pose fine-tuned on a data acquisition image, note that the model detects cone even in harsh lighting conditions.

Chapter 6

Modeling stereo triangulation error

As illustrated in 3.1, in graph-based SLAM when a new landmark is observed, a new edge must be created between the landmark and the pose from which it was detected. The constraint that the edge contains is of the type:

$$(z_t - h(x_t, m_j))^T Q_t^{-1} (z_t - h(x_t, m_j)) \tag{6.1}$$

where:

- z_t is the measurement of landmark m_j from car pose x_t,
- m_j is the landmark j,
- h is the measurement function,
- Q_t is the covariance of the measurement noise.

Therefore a stereo front-end algorithm must associate to each landmark's estimated pose the corresponding covariance matrix representing the error in the measurement.

The accuracy of the stereo vision system depends on the multiple factors listed below:

- Focal length and baseline, a longer focal length increases the disparity, which improves depth resolution, but it reduces the field of view. Similarly, a larger baseline increases the disparity, but it can cause occlusion issues, where objects close to one camera are not visible in the other.
- Resolution of the cameras. Since digital camera pairs map points from the continuous world to a discrete set of pixel pairs, their observations are subject to pixelation error. As a result, distinct world points become indistinguishable after projection and reconstruction by triangulation. Consider Fig. 6.1 that represents the geometry of stereo triangulation for

the case of 2D points projecting onto one-dimensional (1D) images. The tick marks on the image planes denote pixel boundaries, and the radiating lines extend these boundaries into space. Every point in the gray area is projected in one pixel of the left camera and one pixel of the right camera, leading to a significant loss of information. This effect can be mitigated by increasing the camera resolution.

- The precision of the intrinsic and extrinsic camera parameters, which are determined during the calibration process.

- The accuracy of the algorithm responsible for detecting the coordinates of corresponding pixels in the left and right images. An error in the estimation of x_l and x_r for matching points causes a wrong disparity value and consequently an error in the corresponding depth.

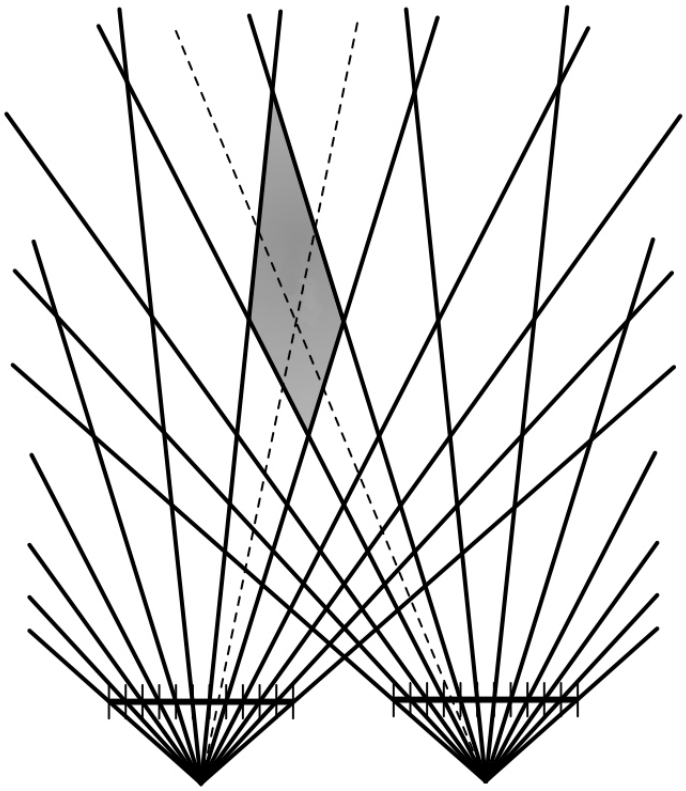


Figure 6.1: Pixelization error representation. (Source [30])

In this work, the focus will be on modeling the cone estimated position noise by propagating the error in detecting the cone peak in the two images. As suggested in [31], the approach is to assume 2D, normally distributed (Gaussian) error in the measured image coordinates and to derive 3D Gaussian distributions describing the error in the inferred 3D coordinates. The use

of Gaussian distributions to model image coordinate error is a common, convenient approximation that gives adequate performance. For the 3D coordinates, the true distribution will be non-Gaussian because triangulation is a nonlinear operation; it is approximated as Gaussian for simplicity and because it gives an adequate approximation when the distance to points is not extreme.

As explained in Sec. 3.2, the 3D coordinates of an image point in the left camera reference frame are computed using the disparity-to-depth matrix Q obtained after stereo rectification:

$$\begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{(c_x - c'_x)}{T_x} \end{bmatrix} * \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{6.2}$$

where T_x is the baseline, d is the disparity, c_x and c_y are the coordinates of the principal point of the left camera and c'_x is the x coordinate of the principal point of the right camera.

The relation 6.2 is equivalent to the function:

$$f : \mathbb{N}^3 \to \mathbb{R}^3, \quad f(x_l, y, x_r) = \begin{bmatrix} \frac{(x_l - c_x)T_x}{c_x - c'_x - x_l + x_r} \\ \frac{(y - c_y)T_x}{c_x - c'_x - x_l + x_r} \\ \frac{fT_x}{c_x - c'_x - x_l + x_r} \end{bmatrix} \tag{6.3}$$

where x_l, x_r and y are the coordinates of the pixel in the left and right camera respectively, y is the same for the two cameras because we are assuming rectified stereo images in input.

Assuming that these coordinates are corrupted by independent, uniform noise, which is approximated as Gaussian, then the covariance of the image observations is:

$$V \approx diag[\sigma^2{}_l \; \sigma_y^2 \; \sigma_r^2] \tag{6.4}$$

where $\sigma^2{}_l$, σ_y^2 and σ_r^2 denote the variance of the corresponding observation. If the image coordinates are assumed uncorrelated and only the pixelation error is considered, V is the identity matrix. Therefore, image coordinates are a random vector with mean $\mu = (x_l, y, x_r)$ and covariance V.

If f was linear, the 3D coordinates of the point would be normal with mean $\mu_P = f(\mu)$ and covariance $V_P = JVJ^T$ where J is the jacobian matrix of f:

$$J = \frac{T_x}{(-x_l + x_r + c_x - c'_x)^2} \begin{bmatrix} x_r - c'_x & 0 & c_x - x_l \\ y - c_y & (-x_l + x_r + c_x - c'_x) & c_y - y \\ f & 0 & -f \end{bmatrix} \tag{6.5}$$

Since f is nonlinear these expressions do not hold exactly, but they are used as satisfactory

approximations.

In the graph-based SLAM used by the Race UP team, landmark nodes are 2D points, therefore the Y coordinate of the 3D position of the cone P can be ignored. Consequently, the error associated with the y image coordinate has no impact on the covariance matrix of triangulated point P. A new f' can be defined:

$$f' : \mathbb{N}^2 \to \mathbb{R}^2, \quad f'(x_l, x_r) = \begin{bmatrix} \frac{(x_l - c_x)T_x}{c_x - c'_x - x_l + x_r} \\ \frac{fT_x}{c_x - c'_x - x_l + x_r} \end{bmatrix} \tag{6.6}$$

The covariance of the image observations becomes $V' \approx diag[\sigma^2_l \; \sigma^2_r]$ and the covariance of the triangulated point $V'_P = J' V J'^T$ where the jacobian J' is

$$J' = \frac{T_x}{(-x_L + x_R + c_x - c'_x)^2} \begin{bmatrix} x_r - c'_x & c_x - x_l \\ f & -f \end{bmatrix} \tag{6.7}$$

Geometrically, the covariance of the landmark measurement noise can be visualized as an ellipse centered at the estimated landmark position, with its principal axis oriented toward the origin of the left camera's reference frame. Figure 6.2 shows an example of ellipses with corresponding cone positions.

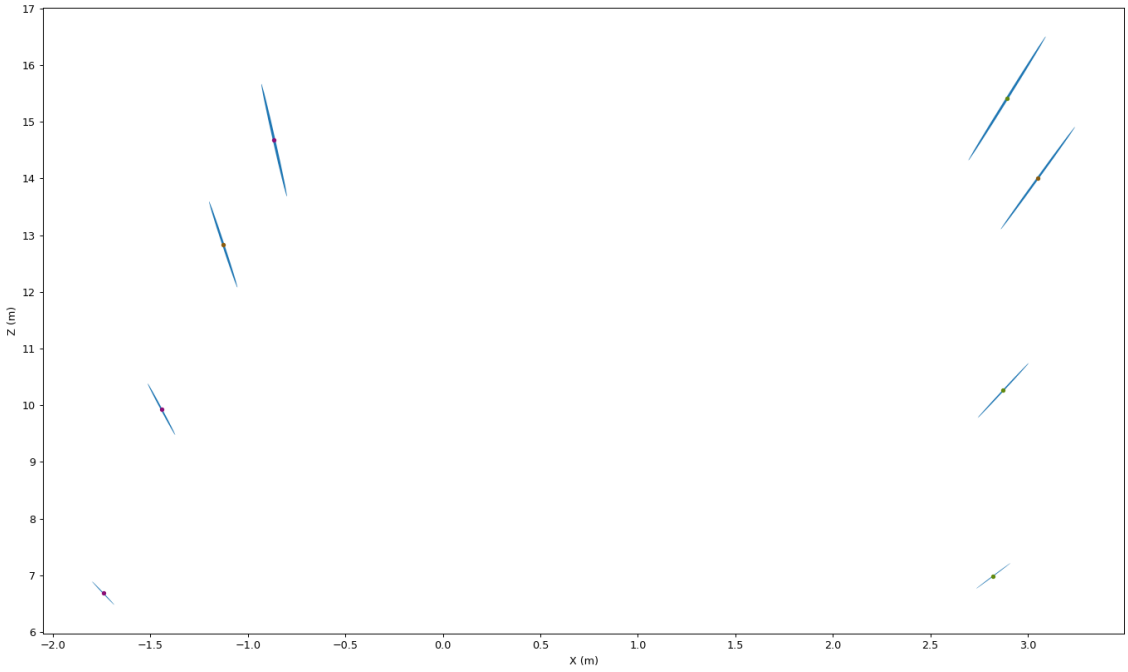


Figure 6.2: Example of covariance matrices visualization. Points are cone estimated position and in blue the covariance ellipses.

Chapter 7

Software Architecture and implementation

The main goal of the software design was to achieve real-time stereo images processing while maintaining accurate results. In order to guarantee reliability and good integration with the existing Race UP software stack, the SLAM front-end was developed using ROS 2 Foxy in C++. The stereo image processing is broken down into two distinct nodes that communicate between each other using message passing.

7.1 ROS

The Robot Operating System (ROS) [32] is an open-source software framework designed to facilitate the development of complex robotic applications. It is not an operating system in the traditional sense but rather a collection of tools, libraries, and conventions that simplify the process of creating robust and scalable robots. ROS provides a distributed architecture that enables communication between various components such as sensors, actuators, and control systems through a publish-subscribe messaging model.

ROS has evolved over time, with two major versions: ROS and ROS 2. The original ROS is widely used, while ROS 2 is a more recent iteration designed to enhance reliability, scalability, and security. Both versions offer a rich suite of developer tools, extensive community support, and a vast repository of software packages.

The ROS architecture consists of key components such as nodes, which are the basic building blocks of a ROS system, and the ROS Master, which acts as a centralized coordination mechanism facilitating communication between nodes. Topics and messages are used for data exchange between nodes, allowing for efficient and flexible system design.

ROS 2 nodes are independent processes that can publish and subscribe to topics, act as service clients or servers, and function as action clients or servers. Each node has configurable parameters that define its behavior. Communication between nodes is established dynamically

through a discovery process.

7.2 Stereo image acquisition node

To ensure flexibility in using different cameras and routing images to various processing algorithms, the image acquisition node should remain independent of the image processing node.

During the first data acquisition, as described in Sec. 4.1, the Bumblebee2 RGB stereo camera was used. Its acquisition node was part of the `bumblebee2_ros_driver` package, which wraps a basic, ROS-agnostic driver (`bumblebee2_driver`). However, since the Bumblebee2 cameras are no longer available to the Race UP team, this node was not used in this work. All stereo images collected during data acquisition were stored in a rosbag, allowing every node to interact with them by subscribing to the `/stereo_image` topic advertised by the rosbag. Each message contains both the left and right images acquired by the cameras.

A rosbag, often referred to as a ”bag,” is a file format used in ROS to store ROS message data. These files, identified by their .bag extension, allow users to record, store, and analyze data from various ROS topics. Bags are typically created using tools like rosbag, which subscribes to ROS topics and stores the serialized message data in a file as it is received. These bag files can be played back to the same topics they were recorded from or remapped to new topics, mimicking real-time data flow. This feature is particularly useful for testing, debugging, and simulating robotic systems without the need for actual hardware.

For the new FLIR Blackfly S cameras the acquisition software has a more complex structure. Unlike the Bumblebee2, which integrates both cameras into a single package, the FLIR Blackfly S cameras are independent units. To achieve hardware synchronization, they are connected via a GPIO cable with a 6-pin Hirose HR10 connector, where the left camera generates a synchronization pulse.

To control and manage the cameras, the Spinnaker ROS2 Synchronized Camera Driver [33] is used. This driver is specifically designed for hardware-synchronized cameras manufactured by Teledyne FLIR and operates with the Spinnaker SDK, which is the software development kit designed by Teledyne FLIR for controlling and managing their cameras. The key advantage of this driver over running multiple instances of an unsynchronized camera driver is that it assigns identical header timestamps to frames generated by the same synchronization pulse, ensuring precise temporal alignment.

The synchronized camera server operates by instantiating multiple individual camera servers, each implemented as a ROS 2 node. Every camera server is fully configurable, offering access to all parameters provided by the Spinnaker camera driver. The configuration is customizable, allowing users to specify which settings to load when launching the synchronized

camera server. All the camera nodes are run in a `ComposableNodeContainer` since it allows multiple nodes to be loaded into a single process, which can improve performance by reducing inter-process communication overhead.

Each camera node stores acquired images in a queue, and a separate thread publishes them to a ROS 2 topic (`/cam_sync/cam0/image_raw` or `/cam_sync/cam1/image_raw`). To ensure accurate timestamps, the system corrects the difference between the camera time and ROS time using either a Kalman filter or an Exponential Moving Average (EMA). Additionally, each camera node publishes a metadata topic containing exposure time, gain information, and other relevant parameters (`/cam_sync/cam0/meta` and `/cam_sync/cam1/meta`).

7.3 Running YOLO11 on C++

As described in 5.3, the `YOLO11-Pose` model was trained to detect cone peaks. The training was conducted using the Ultralytics [24] Python package, which is built on PyTorch and simplifies the usage of YOLO models. However, the model must run within a software stack written in C++, and while it is possible to implement a ROS node in Python, C++ ensures better performance.

There are several alternatives for running a PyTorch model in C++:

- Serialize the model as a TorchScript file and use LibTorch
- Convert the model to the ONNX format and use the ONNX Runtime for inference.
- Convert the model to TensorRT format for optimized inference on NVIDIA GPUs.
- Convert the model to an OpenVINO format for optimized inference on Intel CPUs and VPUs.

Since not all team members have access to an NVIDIA GPU or an Intel CPU, TensorRT and OpenVINO were excluded. Although ONNX Runtime offers high performance, it requires installing additional dependencies for its runtime. In the end, LibTorch was chosen because it is lightweight, requires no extra dependencies or dedicated GPU, and closely resembles PyTorch, making integration easier. Since the embedded computer in the car is equipped with an Nvidia GPU, the best performance can be achieved with TensorRT, so transitioning to that library is recommended. For now, ease of development and testing has been prioritized.

LibTorch [34] is the official C++ API for PyTorch, designed to provide a similar experience to its Python counterpart. Both PyTorch and LibTorch are built on ATen, a C++ tensor library that operates on both CPU and GPU, leveraging CUDA and cuDNN for accelerated computations when available. LibTorch maintains a nearly identical API to PyTorch, ensuring that most

functionalities available in PyTorch can be implemented in C++ as well. Since it is fully native to C++, it can offer performance improvements, particularly by reducing Python's overhead and enabling better integration within C++ based systems.

To use any YOLO model in LibTorch, the model must first be serialized into a .torchscript or .pt file using PyTorch or the Ultralytics Python package. This file contains a serialized PyTorch ScriptModule, which can be deserialized and loaded in C++ for execution without any dependency on Python.

Each stereo image consists of a batch of two images, which can be processed simultaneously by the model. However, before inference, each image must undergo a preprocessing pipeline consisting of the following steps:

1. The image is resized maintaining its aspect ratio and it is padded to reach the final size of 640 x 640

2. Convert to RGB color space

3. Normalize the image by scaling the pixel values to the range [0, 1] by dividing by 255.0

4. Convert the image to a Tensor

After inference, the output requires post-processing to extract the final bounding boxes, keypoints, and confidence scores. In this case, YOLO11-Pose returns a 2 x 11 x 8400 tensor, where:

- 2 is the batch size
- 11 corresponds to the number of values computed for each candidate bounding box:
  - Indices 0 to 3: Bounding box coordinates, represented in (x, y, w, h) format, where (x, y) is the center and (w, h) are the width and height of the bounding box.
  - Indices 4 to 8: Confidence scores for each of the detected classes.
  - Indices 9 and 10: The (x, y) coordinates of the detected peak point of the cone
- 8400 is the number of candidate bounding boxes that is fixed and depends on the input size

The post-processing pipeline consists of the following steps:

1. The maximum confidence score among the five classes is determined for each candidate bounding box. Bounding boxes with a maximum confidence below a predefined threshold are discarded.

2. The bounding box format is converted from (x,y,w,h) to (x,y,x,y) format (x and y coordinates of the top left and bottom right corner).

3. The class index corresponding to the class with the maximum confidence score is assigned to the bounding box.

4. Apply Non-Maximum Suppression (NMS) to remove overlapping bounding boxes.

5. The bounding boxes and keypoints are scaled to match the original image dimensions.

6. Store each bounding box and keypoint in a custom struct called `Detection`

The final result is a vector of Detections for each one of the two input images.

7.4 Stereo image processing node

The `YoloConePose` node is responsible for retrieving images from the stereo image acquisition nodes, detecting cones and peak points in both images using the fine-tuned `YOLO11-Pose` model, and computing the cone 3D position along with its associated error covariance matrix.

7.4.1 Initialization

In the initialization process, the node initializes various parameters and objects required for stereo image processing and cone detection.

- The node loads the camera calibration parameters from three YAML files: one for the left camera, one for the right camera, and a third file containing the extrinsic stereo parameters that define the transformation from the left to the right camera.

- The camera parameters are used to compute the joint undistortion and rectification transformation map for both cameras. This step also produces the disparity-to-depth matrix, essential for depth estimation.

- The node subscribes to one or more topics published by the stereo image acquisition nodes to receive real-time stereo images.

- It loads the fine-tuned `YOLO11-Pose` model for cone and peak point detection.

- The node publishes the `/cone_pose` topic where sending messages containing the cone positions and their associated covariance matrices for the SLAM node.

- If the `ENABLE_VISUALIZATION` flag is enabled, the node publishes an additional `/out_image` topic, which provides an image containing an annotated stereo image displaying the detected and matched cones. This feature is useful for debugging and verifying the correct detection of cones in front of the vehicle.

7.4.2 Image Callback

Whenever a new stereo image is received from the stereo camera, a callback function processes the images to detect cones and estimate their positions following these steps:

1. The images are loaded as OpenCV matrices, and debayering is applied if necessary (Fig. 7.1).

2. Images are rectified by applying the transformation map computed during node initialization (Fig. 7.2).

3. The rectified images are preprocessed into a batch tensor, passed through the YOLO model for inference, and the raw model outputs are post-processed to generate a list of detections (Fig. 7.3).

4. Cones detected at very long distances, characterized by small bounding boxes, are discarded, as their position estimation would be unreliable.

5. The detected cones in the left and right images are matched by comparing their bounding boxes. For each bounding box in the left image, the center is computed and paired with the bounding box of the same class in the right image that has the closest y-coordinate of the center. If the differences in x and y coordinates between the two centers fall within a reasonable threshold, the match is accepted, and the disparity is computed using the cone's peak coordinates (Fig. 7.4) . For each cone peak a point $(x_{peak}, y_{peak}, disparity)$ is generated for triangulation, where (x_{peak}, y_{peak}) are the coordinates of the cone peak in the left image. The matching bounding boxes are not searched strictly along the same horizontal epipolar line, as the rectification may not be perfect, or the predicted bounding boxes could be slightly shifted up or down by a few pixels between the left and right images.

6. The matched cone points are triangulated into 3D space using the `cv::perspectiveTransform` function of OpenCV and the disparity-to-depth matrix computed in the node initialization.

7. 3D points with negative depth values are discarded. This situation may arise if the cameras are excessively convergent, causing a significant x-axis shift in the principal points after

rectification. In such cases, depth estimation is only possible when the disparity is greater than the difference between the two principal point x-coordinates

8. The covariance matrix associated with the triangulated point is computed, as described in Ch. 6.

9. The computed 3D positions of the detected cones are published as `eufs_msgs::msg::ConeArrayWithCovariance` messages. Cones are classified by type (e.g., blue cones, yellow cones) and published with their respective 2×2 covariance matrices.

10. If visualization is enabled, a message is published on the `/out_image` topic containing the rectified stereo images displayed side by side, with lines drawn between matched cones.

Figure 7.1: Stereo image acquired from Bumblebee camera.

Figure 7.2: Rectified and undistorted image.

Figure 7.3: Result of YOLO11 Pose inference.

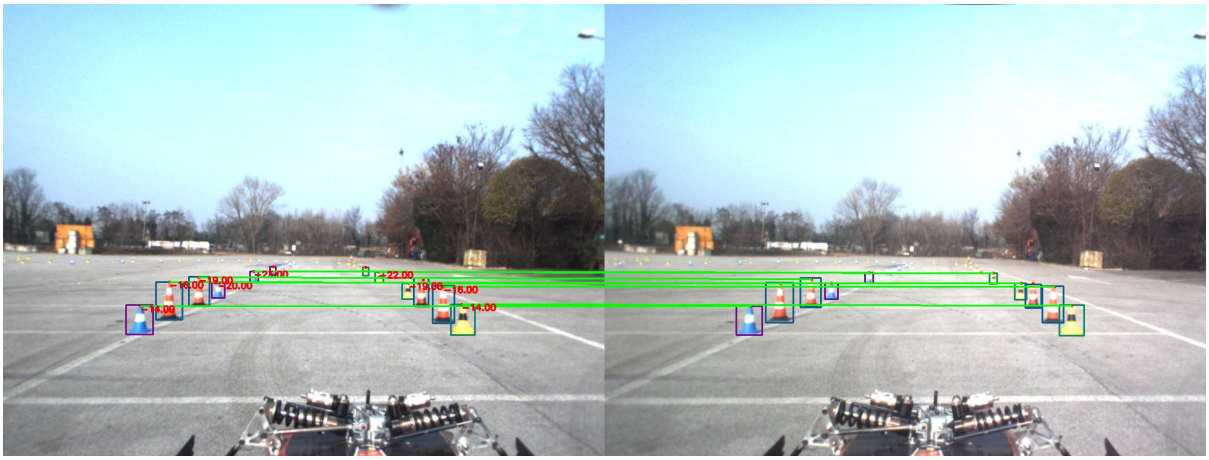


Figure 7.4: Result of the matching step with disparity visible in red. Note that the small cones in the back are removed.

Chapter 8

Experiments and Results

8.1 YOLO11-pose evaluation

In this section, the fine-tuned YOLO11-Pose model is evaluated for detection, classification, and peak point detection.

To assess the model's performance in detection and classification, the selected metrics include precision, recall, mAP50 and mAP50-95:

- Precision is the proportion of the model's positive classifications that are actually positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Recall is the proportion of actual positives that were classified correctly as positives.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- Intersection over Union (IoU) is a metric used to measure the overlap between the predicted bounding box and the ground truth bounding box. It is defined as the ratio of the area of their intersection to the area of their union:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

- Average Precision (AP) is a metric that evaluates the precision-recall curve for object detection models. It is computed as the area under the Precision-Recall (PR) curve, which plots precision against recall at different confidence thresholds:

$$\text{AP} = \int_0^1 P(R)\, dR$$

where $P(R)$ is precision as a function of recall. Higher AP values indicate better detection and classification performance.

- Mean Average Precision at IoU 0.5 (mAP50): The mean of the average precision (AP) scores calculated at a fixed Intersection over Union (IoU) threshold of 0.5. A detection is considered correct if the IoU between the predicted and ground truth bounding box is at least 0.5.

- mAP50-95: The mean of the AP scores computed across multiple IoU thresholds, ranging from 0.50 to 0.95 in increments of 0.05. This provides a more comprehensive evaluation of the model's performance across different levels of localization precision.

The results for bounding box detection and classification on the test split of the dataset used for training, annotated with cone peaks, are reported in table 8.1:

Class	Instances	Precision	Recall	mAP50	mAP50-95
blue_cone	888	0.926	0.680	0.854	0.843
large_orange_cone	59	0.803	0.831	0.866	0.864
yellow_cone	845	0.944	0.688	0.858	0.849
orange_cone	255	0.913	0.733	0.848	0.838
unknown_cone	27	0.247	0.243	0.226	0.223
all	2074	**0.767**	**0.635**	**0.730**	**0.723**

Table 8.1: Model Performance on Test Split

The model performs well on well-defined cone classes (blue, large orange, yellow, and orange cones) but there is a significantly lower performance for the unknown_cone class since there are few instances of it.

Looking at the confusion matrix (Fig. 8.1), there are no misclassifications among the well-defined cone classes. However, a significant number of cones are missed, as they are present in the scene but incorrectly classified as background. These missed detections mainly correspond to small cones located far from the camera. Given the inherent difficulty in estimating depth for such distant objects, this issue is negligible. Because of that, the recall metric is generally lower than precision, which remains high due to the very low number of false positives. This is a crucial aspect, failing to detect a few small cones is preferable to mistakenly sending non-existent ones to the SLAM back-end.

To assess the accuracy of the predicted peak point position relative to the true position, the following metrics are computed:

- Euclidean distance between the true and predicted peak point positions.

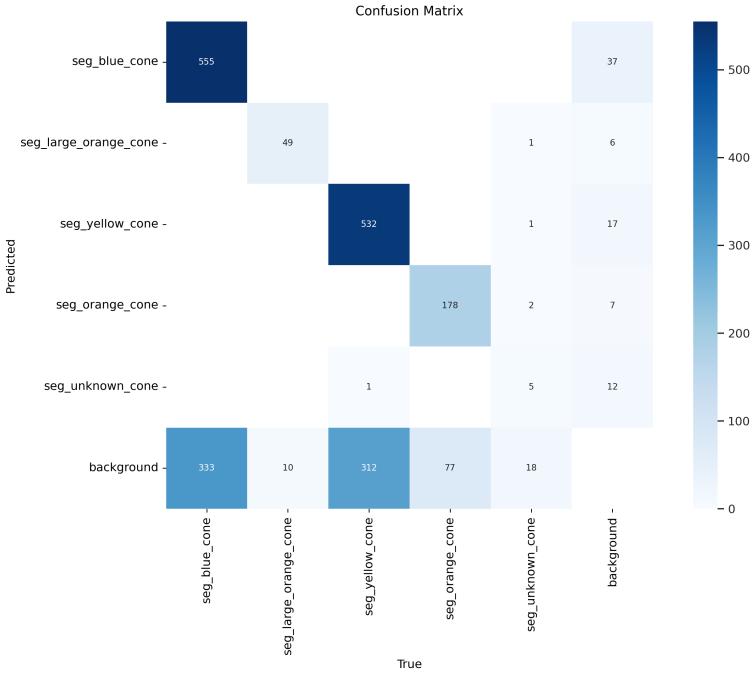


Figure 8.1: Confusion matrix for fine-tuned YOLO11-Pose model

- Absolute difference between the true and predicted x-coordinates of the peak point, capturing horizontal displacement.
- Absolute difference between the true and predicted y-coordinates of the peak point, capturing vertical displacement.
- Object Keypoint Similarity (OKS), a normalized score ranging from 0 to 1, evaluates the similarity of the keypoint's position while also considering the size of the bounding box in which the keypoint resides.

OKS is mathematically defined as:

$$OKS = \frac{\sum_i e^{-\frac{d_i^2}{2s^2k_i^2}} \delta(v_i > 0)}{\sum_i \delta(v_i > 0)}$$

where:

- d_i is the Euclidean distance between the predicted and ground truth keypoint i.
- s is the object scale (usually the area of the bounding box).
- k_i is a keypoint-specific constant that accounts for localization variance.

- v_i is the visibility flag (1 if visible, 0 if not). In our case, cone peaks are all set as visible, even if in some rare cases there are some occluded cone peaks, especially for nearby cones.
- $\delta(v_i > 0)$ ensures that only visible keypoints contribute to the score.

The results are summarized in Table 8.2, with a total of 827 keypoints detected across the images in the test set. When analyzing the distance error in pixels, it is observed that the error is larger in the y-coordinate than in the x-coordinate. This is encouraging, as the x-coordinate is primarily used for disparity computation. The high variance in the errors is likely due to the varying sizes of the bounding boxes. The Object Keypoint Similarity (OKS), which accounts for bounding box sizes, shows a high value, with values close to 1 indicating that the keypoints are nearly aligned with the ground truth positions. Additionally, the standard deviation of OKS suggests that the peak points are generally well-placed. The distributions of these metrics are shown in Fig. 8.2.

Metric	Mean	Std. dev.
Euclidean distance (px)	3.5487	2.8638
x distance (px)	1.9040	1.2773
y distance (px)	2.5784	2.4731
OKS (k=0.05)	0.9796	0.0454

Table 8.2: Keypoint positioning metrics.

8.2 Cone position estimation

8.2.1 New stereo camera setup

After the stereo camera calibration and image rectification, the camera parameters required for triangulation become available, enabling an initial analysis of the system's depth estimation capabilities. The distance between the principal points of the two cameras is $c_x - c'_x = 27$ pixels, which determines the maximum theoretical depth range of 330 meters. The relationship between disparity and depth is illustrated in Fig. 8.3.

To assess the accuracy of the new system in estimating cone position in the left camera reference frame, the following experiment was conducted.

The new stereo camera setup, consisting of two FLIR Blackfly S cameras, was mounted on the test mule to replicate the field of view that the cameras will have on the competition car. Cones were strategically placed in front of the car at various positions within the image frame, and images were actively acquired. Each cone's true position was precisely measured, allowing for an accurate evaluation of the error between the estimated and actual positions.

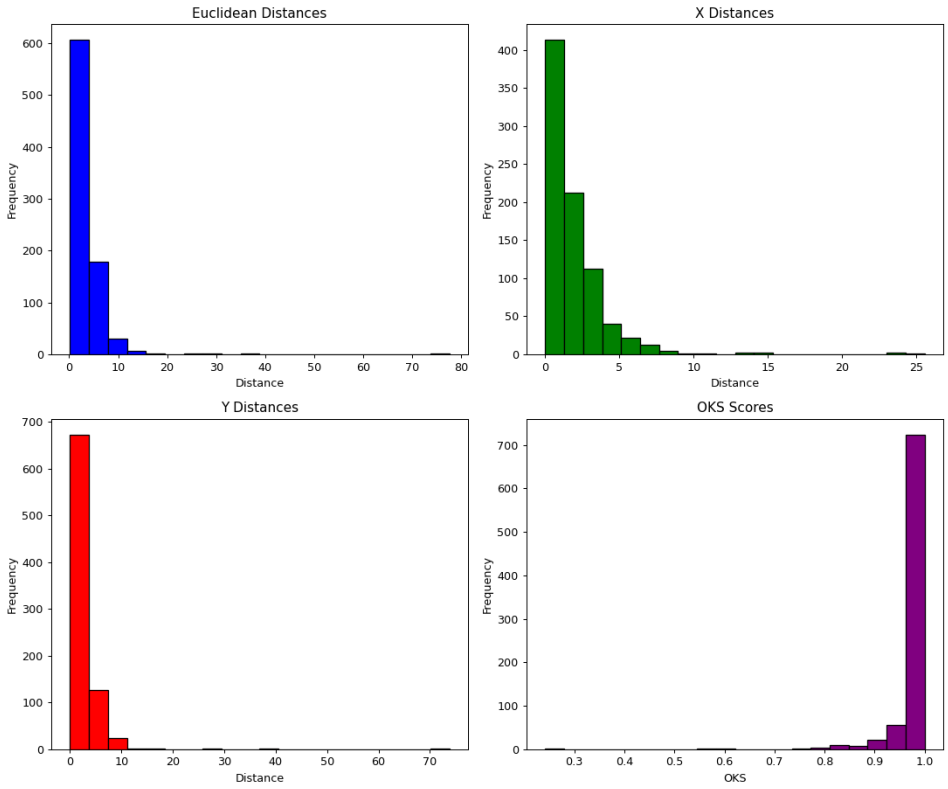


Figure 8.2: Keypoint metrics distribution.

These cones were placed at different locations to test the system across a range of depths. The maximum tested depth was 18 meters, while the nearest tested cone was placed at 2.80 meters, which is less than one meter from the nose of the car.

Since the y-coordinate (vertical displacement) is not relevant in this context, only the longitudinal error (z-axis) and lateral error (x-axis) are analyzed.

From Fig. 8.4 and Fig. 8.5, it is evident that for cones within 12 meters, the position estimation remains reasonably accurate, with errors consistently below 0.5 meters for both depth (z) and lateral (x) coordinates. However, the measurements exhibit some variability, likely due to uncertainties in peak cone positioning and minor inaccuracies in image rectification that can cause a slight change in the disparity value.

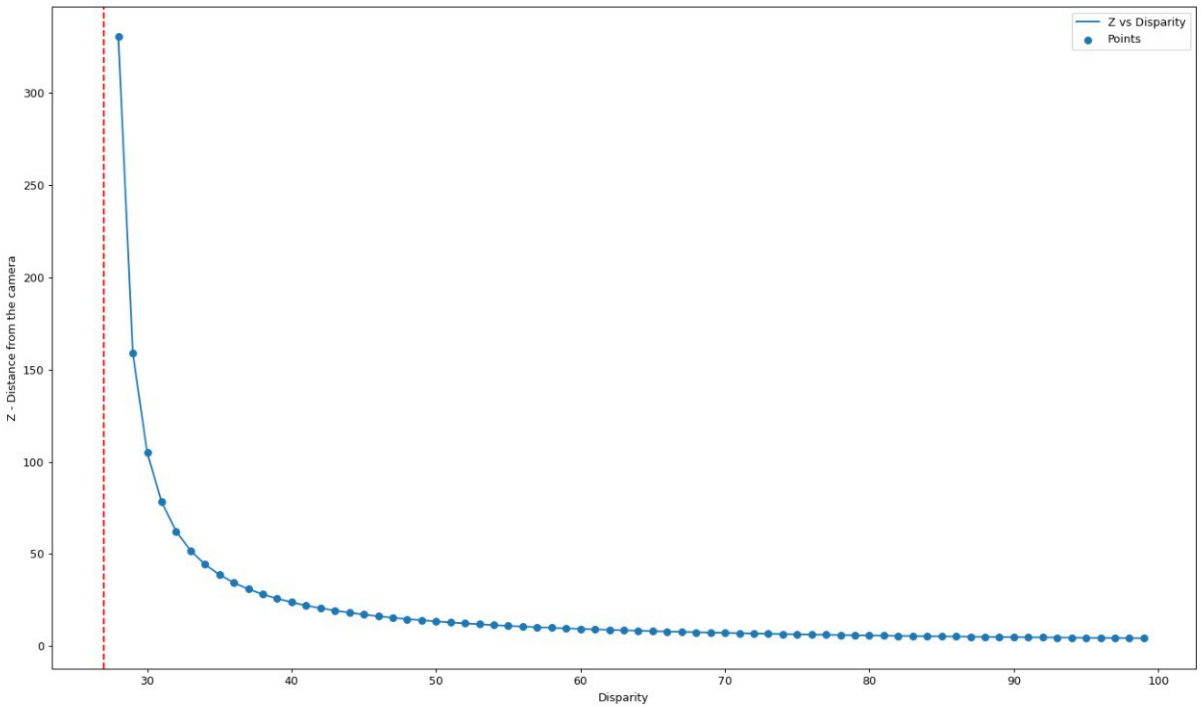


Figure 8.3: Disparity vs Depth (m) relation of FLIR Blackfly S stereo setup.

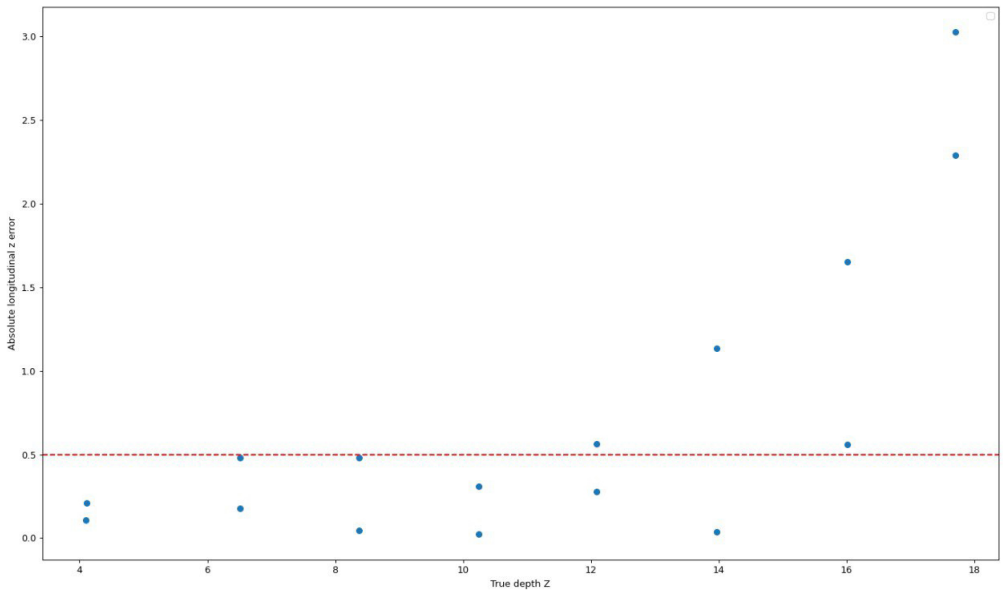


Figure 8.4: True depth Z (m) and absolute depth Z estimation error.

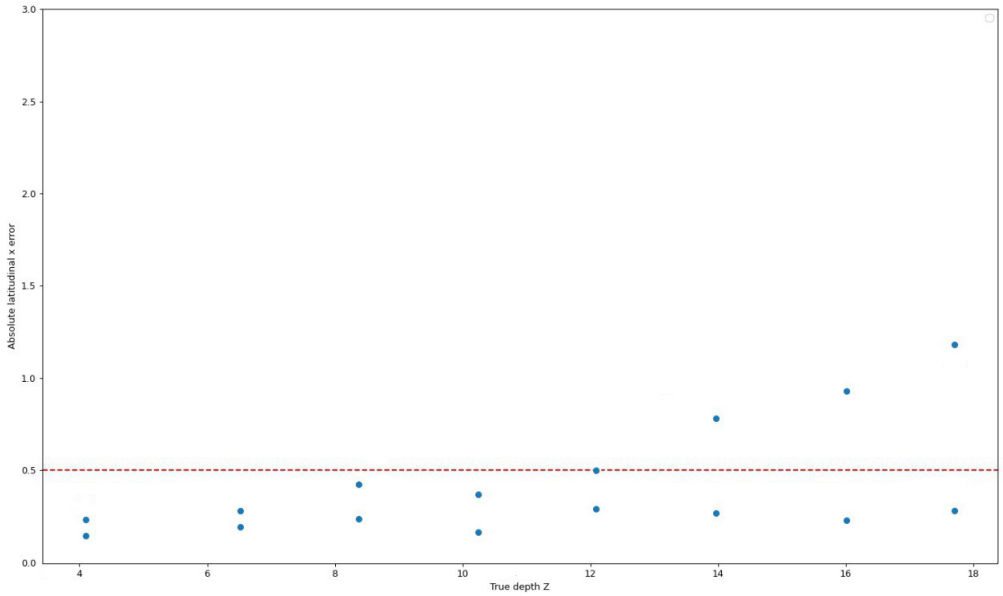


Figure 8.5: True depth Z (m) and absolute lateral X estimation error

8.2.2 Comparison with the older setup

The previous theses and team reports did not provide a detailed analysis of the Bumblebee2 RGB stereo camera's performance. While an accuracy below 0.5 m in cone positioning is claimed, the depth range at which this accuracy holds is unspecified.

A key limitation observed in the Bumblebee stereo camera is its coarse disparity resolution. Since disparity values are quantized, even a one-pixel displacement between the two cone images can lead to an error of nearly 0.5 meters in depth estimation.

Since the Bumblebee camera was unavailable for direct testing, a side-by-side comparison with the new FLIR Blackfly S cameras was not possible. However, theoretical analysis provides insight into their relative performance. The maximum measurable range of the Bumblebee camera is 112 meters since the distance between the two principal points is $c_x - c'_x = -26$ pixels after rectification.

Figure 8.6 illustrates the disparity-to-depth function for both stereo camera setups, aligned at their respective minimum disparities. This allows for a direct comparison of how small disparity changes impact depth estimation.

From the plot, key observations emerge:

- With the Bumblebee2 stereo camera, a 1-pixel disparity error results in a 0.5 m depth error for objects just 8 meters away.
- The FLIR camera, in contrast, exhibits this level of error only for objects beyond 12 meters, reducing its impact at typical operating ranges.
- In the FLIR Blackfly S setup, the disparity-to-depth curve grows less steeply, meaning depth errors increase more gradually than in the Bumblebee camera.

These findings suggest that, while the Bumblebee2 camera may be more suited for depth estimation of close objects, its accuracy degrades rapidly with distance. The FLIR Blackfly S stereo setup provides finer disparity values, leading to more reliable depth estimation at mid-range and long distances.

The superior accuracy of the FLIR Blackfly S system is further confirmed by analyzing the estimated covariance matrices associated with the position estimation. Fig. 8.7 illustrates how the largest eigenvalue of the error covariance matrix evolves with increasing depth. This eigenvalue corresponds to the variance in range estimation in polar coordinates. In this analysis, only the quantization error is considered, meaning that the variance of the x-coordinate of the cone peak in the left and right image is fixed at 1. The results show that the Bumblebee stereo camera exhibits significantly higher variance, with its error increasing more rapidly compared to the FLIR system.

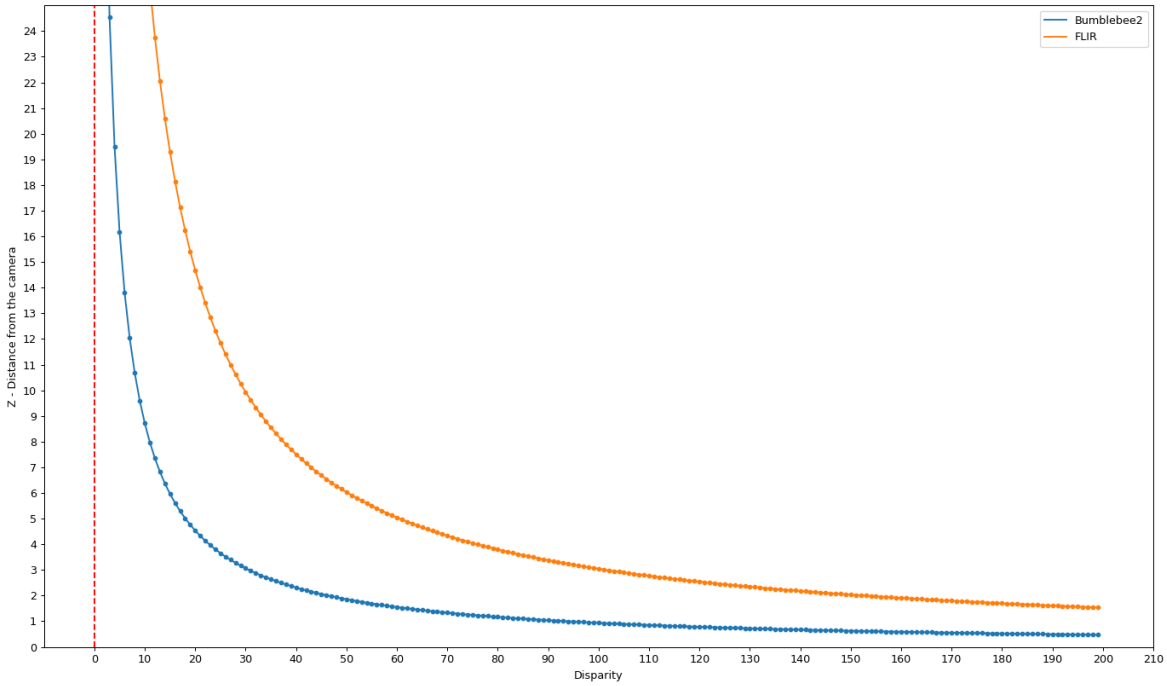


Figure 8.6: Disparity vs Depth (m) relation of FLIR Blackfly S and Bumblebee aligned to see the impact of one pixel change.

8.3 Pipeline latency

The performance of the full pipeline latency was tested on a computer different from the embedded system that will be placed in the car. The test machine is equipped with an AMD Ryzen 5 3600 processor (6 cores, 12 threads), 16 GB of RAM, and an NVIDIA GTX 1060 GPU with 3 GB of VRAM. This setup is significantly less powerful than the embedded system described in 4.2, so a performance improvement is expected when running the algorithm on the final hardware.

Latency was measured by processing all 9,946 stereo images contained in the rosbag from the first data acquisition. This approach ensured precise measurement while also evaluating the reliability of the ROS image processing node. After five test runs, the node consistently processed all images without any issues or slowdowns.

The execution time (in ms) for each step is reported in Table 8.3. On average, the left and right images are processed in 38 ms and in under 42 ms in the worst case. This represents a significant improvement over the previous implementation, which required 67 ms to elaborate a single image.

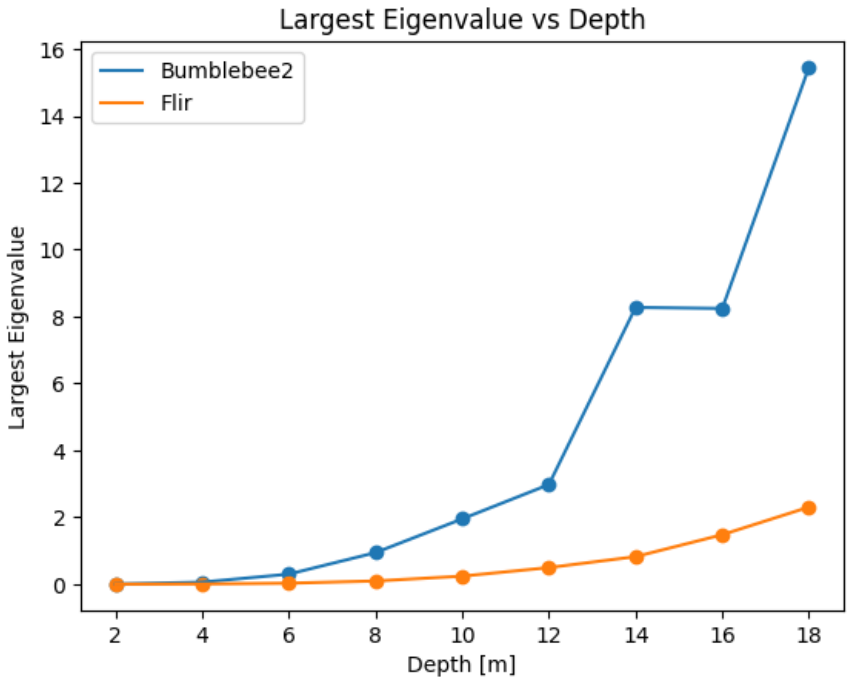


Figure 8.7: Relation between the depth Z of a cone position and the largest eigenvalue in the associated covariance matrix.

Step	Time (ms)
Rectification	2 ± 0
Preprocessing	3 ± 0
Inference	7 ± 1
Post-processing	26 ± 2

Table 8.3: Pipeline latency for processing a pair of images.

Chapter 9

Conclusion

This thesis, developed in collaboration with the Race UP Driverless team of the University of Padova, presents an approach for implementing a stereo vision SLAM front-end, addressing both hardware and software aspects.

After a brief introduction to the Formula Student Driverless competition, outlining the operational context and rules, a review of related works on similar topics is provided, presenting alternative approaches and benchmark results for comparison.

The SLAM problem is then introduced, with a specific focus on the graph-based SLAM currently used by Race UP. The key requirements for an effective stereo front-end are discussed, detailing the necessary inputs for the SLAM back-end. Additionally, a theoretical background on stereo camera systems is provided to facilitate understanding of the challenges and solutions encountered in this work.

The current sensor setup and the newly developed system are presented, explaining how the cameras are installed and configured to operate in stereo mode.

The fine-tuned YOLO11-Pose model is then introduced, along with the rationale behind its selection and the dataset creation process for training. The model improves classification accuracy, detects cones, and predicts peak point positions, all while being faster than the previous setup. The stereo rectified images serve as input to the model to obtain peak points, which are then used for disparity computation.

Since the estimated 3D cone positions are subject to errors and noise, this work also introduces a method to model these uncertainties using a covariance matrix, which is associated with the estimated 3D coordinates of the cone.

After an explanation of the pipeline implementation in ROS 2, the final results are presented, evaluating the model's accuracy and the cone position estimation performance. A comparison between the new and old stereo camera setups highlights the improvements and justifies the advantages of the proposed system.

In conclusion, the reported experimental results confirm the effectiveness of the proposed system, demonstrating its strengths, limitations, and practical applicability. This work lays a solid foundation for future improvements to the camera-based SLAM front-end, which will operate alongside the LiDAR cone detection algorithm currently in development by the team.

9.1 Future work

The new stereo camera setup and the improved cone detection model represent a significant step forward compared to the previous cameras and processing software. However, there are still areas for further enhancements.

The training dataset requires improvements that will be reflected in the final fine-tuned model:

- The dataset contains an insufficient number of unknown cones and large orange cones. Expanding it to include more examples of these cone types could enhance classification accuracy.

- Manually refining the peak annotations in the dataset, which were obtained using the automatic approach described in 5.3, could further improve accuracy. This is particularly relevant for cones close to the car, where the peak is sometimes detected slightly off-center. While this does not significantly impact overall system performance, since disparity inaccuracies in nearby cones lead to only minor errors in position estimation, it remains an area worth refining.

The new camera system was calibrated and tested in a controlled environment, but real-world data acquisition would provide a more comprehensive performance evaluation. Several camera parameters require fine-tuning, including lens focal length, aperture, gain, and exposure time. The optimal balance between these parameters must minimize motion blur, ensure proper cone exposure, and maintain sharp focus. However, it is challenging to determine the best settings using a static or slow-moving test mule. A real-world data acquisition would help refine these parameters, leading to a more well-tested camera setup. Once the final settings are determined, a new stereo camera calibration will be necessary.

Regarding stereo-based cone position estimation, results are highly accurate within 12 meters, but performance degrades significantly for more distant cones. This is a common challenge faced by other Formula Student driverless teams. Some teams have addressed this issue using PnP (Perspective-n-Point) with a keypoint detector model, leveraging only one calibrated camera and the known geometric structure of the cone. The world frame is placed at the base of the cone. Detected keypoints are used as 2D points on the image to make correspondence with

the respective points on the 3D model of a cone. Using the correspondences and the camera intrinsics, PnP is used to estimate the pose of every detected cone.

Currently, there are no publicly available keypoint detection models for cones. Teams such as AMZ and DUT/MIT have trained their custom models [8] [10]. However, no public dataset exists for this task. Given the strong performance of YOLO11-Pose, it could be used as a keypoint detector, increasing the number of keypoints in output for each bounding box from just the cone vertex to seven keypoints: two on the cone base, four on the central colored band and the cone vertex (Fig. 9.1).

The FSOCO dataset was annotated with keypoints using classical computer vision techniques to overcome the lack of a suitable dataset. For each cone, seven keypoints were extracted using color thresholding. However, this method required manual correction, and many cones in the dataset lacked the central band, making automatic keypoint extraction challenging. Due to time constraints, this work was not fully completed, but it remains a promising direction for improving cone position estimation.

Moreover, increasing the number of keypoints in the output from the fine-tuned YOLO11-Pose can also enhance the stereo pipeline, as it provides more disparity values for each cone, thereby improving the accuracy of cone position estimation.

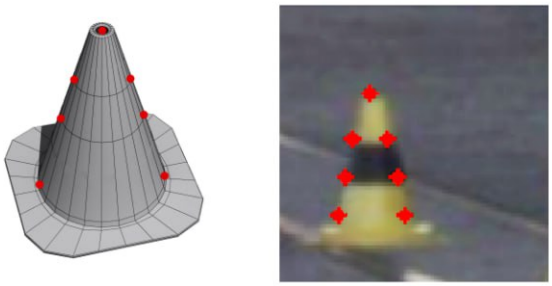


Figure 9.1: 7 possible keypoints to detect on the cone. [8]

Bibliography

[1] ISTAT. “Road accident statistics - first half of 2024.” Accessed: 2025-03-02. (2024), [Online]. Available: https://www.istat.it/wp-content/uploads/2024/11/REPORT_INCIDENTISTRADALI_PRIMOSEMESTRE2024.pdf.

[2] SAE International. “Formula sae rules 2025.” Accessed: 2025-03-02. (2024), [Online]. Available: https://www.fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=4bafd174-62da-48b6-b016-589385ca5ae6.

[3] AMZ Team. “Amz car.” (2025), [Online]. Available: https://www.amzracing.ch/en.

[4] Formula Student Germany. “Fsg competition handbook 2025.” Accessed: 2025-03-02. (2024), [Online]. Available: https://www.formulastudent.de/fileadmin/user_upload/all/2025/important_docs/FSG25_Competition_Handbook_v1.0.pdf.

[5] Raceup Team. “Raceup website.” (2025), [Online]. Available: https://raceup.it/.

[6] J. J. L. Cyrill Stachniss and S. Thrun, “Simultaneous localization and mapping (slam): Part i,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., Springer, 2016, ch. 46. [Online]. Available: http://handbookofrobotics.org/view-chapter/46.

[7] Andreasson, Henrik and Grisetti, Giorgio and Stoyanov, Todor and Pretto, Alberto, “Software architectures for mobile robots,” in *Encyclopedia of Robotics*, M. H. Ang, O. Khatib, and B. Siciliano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–11, isbn: 978-3-642-41610-1. doi: 10.1007/978-3-642-41610-1_160-1. [Online]. Available: https://doi.org/10.1007/978-3-642-41610-1_160-1.

[8] J. Kabzan, M. de la Iglesia Valls, V. Reijgwart, *et al.*, *Amz driverless: The full autonomous racing system*, 2019. arXiv: 1905.05150 [cs.RO]. [Online]. Available: https://arxiv.org/abs/1905.05150.

[9] L. Andresen, A. Brandemuehl, A. Honger, *et al.*, “Accurate mapping and planning for autonomous racing,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2020, pp. 4743–4749. doi: 10.1109/iros45743.2020.

9341702. [Online]. Available: http://dx.doi.org/10.1109/IROS45743.2020.9341702.

[10] K. Strobel, S. Zhu, R. Chang, and S. Koppula, "Accurate, low-latency visual perception for autonomous racing: Challenges, mechanisms, and practical solutions," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2020, pp. 1969–1975.

[11] Y. Rampuria, D. Boliya, S. Gupta, *et al.*, *Iit bombay racing driverless: Autonomous driving stack for formula student ai*, 2024. arXiv: 2408.06113 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2408.06113.

[12] S. Nekkah, J. Janus, M. Boxheimer, *et al.*, *The autonomous racing software stack of the kit19d*, 2020. arXiv: 2010.02828 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2010.02828.

[13] Alessandra Tonin, "Design of a perception system for the formula student driverless competition: From vehicle sensorization to slam," M.S. thesis, 2023.

[14] Edinburgh University Formula Student. "Eufs simulator." (2019), [Online]. Available: https://gitlab.com/eufs/eufs_sim.

[15] Navid Khalili, "A practical multi-sensor localization and mapping approach for the formula student driverless competition," M.S. thesis, 2024.

[16] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, *Introduction to Autonomous Mobile Robots*. The MIT Press, 2011.

[17] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *I. J. Robotic Res.*, vol. 25, pp. 403–429, May 2006. doi: 10.1177/0278364906065387.

[18] H. F. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:17915751.

[19] OpenCV team. "Opencv." (2025), [Online]. Available: https://opencv.org/.

[20] Gary Bradski, Adrian Kaehler, *Learning OpenCV*. O'Reilly, 2008.

[21] MathWorks. "Matlab stereo camera calibrator." (2025), [Online]. Available: https://it.mathworks.com/help/vision/ref/stereocameracalibrator-app.html.

[22] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

[23] N. Vödisch, D. Dodel, and M. Schötz, "Fsoco: The formula student objects in context dataset," *SAE International Journal of Connected and Automated Vehicles*, vol. 5, no. 12-05-01-0003, 2022.

[24] G. Jocher and J. Qiu, *Ultralytics yolo11*, version 11.0.0, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics.

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, 2018. arXiv: 1703.06870 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1703.06870.

[26] Ranjan Sapkota, Dawood Ahmed, Manoj Karkee, "Comparing yolov8 and mask r-cnn for instance segmentation in complex orchard environments," *KeAi*, 2024.

[27] A. Kirillov, E. Mintun, N. Ravi, *et al.*, *Segment anything*, 2023.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1506.02640.

[29] Ultralytics Inc. "Ultralytics." (2025), [Online]. Available: https://www.ultralytics.com/.

[30] C. Freundlich, M. Zavlanos, and P. Mordohai, "Exact bias correction and covariance estimation for stereo vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3296–3304.

[31] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 3, pp. 239–248, 1987.

[32] Open Robotics. "Robot operating system." (), [Online]. Available: https://www.ros.org/.

[33] Ros drivers community. "Ros teledyne flir camera drivers." (), [Online]. Available: https://github.com/ros-drivers/flir_camera_driver.

[34] PyTorch Foundation. "Libtorch." (), [Online]. Available: https://pytorch.org/cppdocs/.