UNIVERSITAT POLITÈCNICA DE CATALUNYA

Bachelor's Thesis

Enhancement of a Formula Student car perception system using a global 3D map

Oriol Gorriz Viudez

advised by
Josep R. Casas Pla
Javier Ruiz Hidalgo

supervised by
Joan Climent Vilaró

June 2022

In partial fulfilment of the requirements for the
Bachelor's degree in Computer Engineering: Computation

Acknowledgments

I would like to thank all the people and institutions who have contributed to this project:

- The Universitat Politècnica de Catalunya (UPC), specifically the Facultat d'Informàtica de Barcelona (FIB) for making me a competent professional. To all the professors I have met, specifically to my advisors Josep R. Casas Pla and Javier Ruiz Hidalgo for helping me throughout the project, their technical and personal guidance has helped me a lot. Also, to my supervisor Joan Climent Vilaró, who always had tips about the formal aspects.

- To the BCN eMotorsport team, for giving me the most powerful tool possible, passion. Everyone in this Formula Student team is passionate about what we do, spending days and nights to see the result of a job well done. To all my colleagues working on the car's autonomous system for being always helpful, ready to work and awesome teammates: Pau Biosca, Oriol Martinez, Marta Miranda, Oriol Pareras, Pol Ramon, Albert Rodas and Toni Salom. Also, to last season's alumni: Albert Gassol, Andreu Huguet, Raúl González and Ignasi Mallén.

- Lastly, to my parents. They support everything that I do and are happy to be part. Thank you for always being there.

Abstract

English

The implementation of an accurate localization and mapping system in our driverless Formula Student car has revolutionized this season's car perception pipeline. Now, a new system that takes advantage of this improved 3D map is needed. Cone positions are obtained in this map, and these are classified with the information extracted from camera images in order to compute the track limits. This thesis proposes a new system to classify and keep track of cones (CCAT) and another system (Urimits) to extend partial color dependant track limits using unclassified cones. Both systems have achieved an enhancement over last season's in range and accuracy. Now the possibility of detecting the whole track limits before the car completes the lap is possible.

Español

La implementación de un preciso sistema de localización y mapeo en nuestro coche de Formula Student sin conductor ha revolucionado el sistema de percepción del coche de esta temporada. Ahora, es necesario un nuevo método que aproveche este mapa 3D mejorado. En este mapa se obtienen las posiciones de los conos, que se clasifican con la información extraída de las imágenes de la cámara para calcular los límites de la pista. Esta tesis propone un nuevo sistema para clasificar y hacer un seguimiento de los conos (CCAT) y otro sistema (Urimits) para extender los límites de pista parciales dependientes del color utilizando conos no clasificados. Ambos sistemas han logrado una mejora con respecto a los de la temporada pasada en cuanto a alcance y precisión. Ahora es posible detectar los límites de la pista enteros cerrando la vuelta antes de que el coche físicamente complete la vuelta.

Català

La implementació d'un precís sistema de localització i mapeig al nostre cotxe de Formula Student sense conductor ha revolucionat el sistema de percepció del cotxe d'aquesta temporada. Ara, cal un nou mètode que aprofiti aquest mapa 3D millorat. En aquest mapa s'obtenen les posicions dels cons, que es classifiquen amb la informació extreta de les imatges de la càmera per calcular els límits de la pista. Aquesta tesi proposa un nou sistema per classificar i fer un seguiment dels cons (CCAT) i

un altre sistema (Urimits) per estendre els límits de pista parcials dependents del color fent servir cons no classificats. Tots dos sistemes han aconseguit una millora respecte als de la temporada passada pel que fa a abast i precisió. Ara és possible detectar els límits de la pista tot tancant la volta abans que el cotxe físicament completi la tornada.

Keywords

English

Formula Student, computer vision, computer science, driverless vehicles, LiDAR, mapping, real time, map-camera registration, algorithmics, statistical models.

Español

Formula Student, visión por computador, ciencias de la computación, vehículos sin conductor, LiDAR, mapeo, tiempo real, registro mapa-cámaras, algoritmia, modelos estadísticos.

Català

Formula Student, visió per computador, ciències informàtiques, vehicles sense conductor, LiDAR, mapatge, temps real, registre mapa-càmeres, algorísmia, models estadístics.

Contents

List of Figures

Acronyms

BB2L Bounding Box To LiDAR.

CAN Controller Area Network.

CCAT Cone Classifier And Tracker.

DOF Degree Of Freedom.

ECU Electronic Control Unit.

EKF Extended Kalman Filter.

FOV Field Of View.

FSG Formula Student Germany.

GEP Gestió de Projectes (project management).

IMU Inertial Measurement Unit.

INS Inertial Navigation System.

LiDAR Light Detection And Ranging.

LIO LiDAR-Inertial Odometry.

RGB Red, Green and Blue.

ROS Robot Operating System.

SLAM Simultaneous Localization And Mapping.

TLs Track Limits.

UPC Universitat Politècnica de Catalunya (technical university of Catalonia).

YOLO You Only Look Once.

Chapter 1

Introduction

1.1 Context

1.1.1 The competition

Formula Student is a student engineering competition held annually in multiple countries (being Germany the most famous). University students around the globe design, build, program, test and race a formula-style racing car, see Fig. 1.1. Likewise F1, Formula Student does also have a strict set of rules [1] that specifies the car's characteristics and what will be the events in which the single-seater will compete.

Figure 1.1: Group photo of FSG 2021.

In 2017, Formula Student Germany first introduced a driverless category. Since then, the importance of autonomous vehicles has been growing steadily.

The main event of Formula Student driverless is to lap a circuit as fast as possible. The score will be distributed according to the time a particular team has achieved compared to the best one.

The car is staged on a track delimited by cones (see DE 6.2 [2] and Fig. 1.2), having a yellow strip of cones on the right and a blue strip on the left; some special cones (orange and bigger) are used to indicate the starting line. In the autocross event, without any previous knowledge about the track layout, the car should com-

plete a single lap. In the trackdrive event, the car should complete 10 laps on the same track.

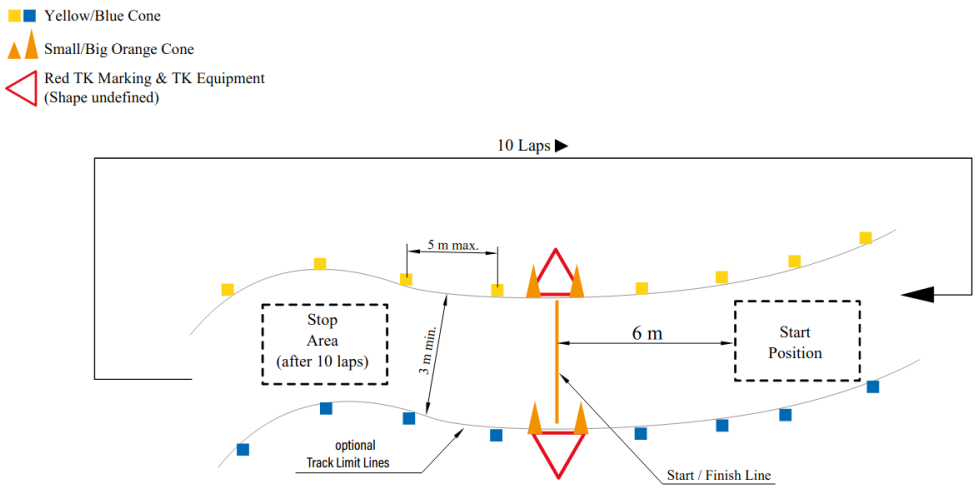

Figure 1.2: Track layout of autocross and trackdrive events.

The trackdrive is considered a localization and a path optimization problem, whereas the autocross is considered a perception challenge because the car should detect the cones and map the track.

1.1.2 The team

The UPC really values the competition and as a result multiple Formula Student teams (some among the top) have been founded.

In 2018, Driverless UPC was founded. It was the first Spanish team focused on driverless vehicles, and their aim was to convert an already built Formula Student car into a driverless car. Xaloc, was the result of that season (see Fig. 1.3).

Figure 1.3: Xaloc, first ever Spanish driverless car of its type.

In 2020, Driverless UPC and ETSEIB Motorsport joined forces, creating BCN eMotorsport.

In BCN eMotorsport, I am responsible for the perception of this season's car. The objective of this system is to recognise the limits of the track. To do that, the car has a LiDAR and cameras that allow sensing the cones as precisely and fast as possible.

Regarding the autonomous system of our car, we are organized in two different departments.

Perception department

Our duty is to compute the limits of the track without having any previous knowledge of the track. To do so, we have two RGB cameras and a rotating LiDAR sensor. From the cameras we obtain a vector of bounding boxes of every cone in the image, which includes a cone tag (color and size). From the LiDAR we obtain a point cloud of what is in front of the vehicle.

Driverless control department

They are responsible for the vehicle's correct performance of the actuators (steering, brakes and accelerator). They receive the limits of the tracks and compute the optimal local trajectory, and when they receive the full limits of the track (full lap) they optimize the global trajectory and velocities.

1.2 Motivation

Last year, when I joined the team, we focussed on a challenging problem: localizing the car in-track.

Localization is a metric that has to be precise and reliable during autonomous driving, otherwise, the vehicle will get out of track very easily. This year, during my journey at BCN eMotorsport, we have improved very significantly our car's localization system with the addition of LIO. The algorithm we have used is LIMO-Velo [3], built on top of Fast-LIO2 [4] and developed by one of our team alumni, Andreu Huguet. It supposes a huge leap for the team's localization system and solves the SLAM problem. With the implementation of this algorithm, we have encountered something unexpected for us, see Fig. 1.4.

The algorithm provides a detailed SLAM [5] map of what the LiDAR sees during its execution time, it is all the information that LiDAR has captured since the beginning of the run but accumulated; the cones in the track are amazingly detailed. This season's overall perception objective is to take advantage of the new data and detect the cones from this map in order to provide this season's car, the CAT14x, with the best perception system possible.

1.3 Problem statement

The implementation of LiDAR-Inertial odometry has revolutionized the entire perception pipeline; a new algorithm to detect cones in this 3D map has been developed,

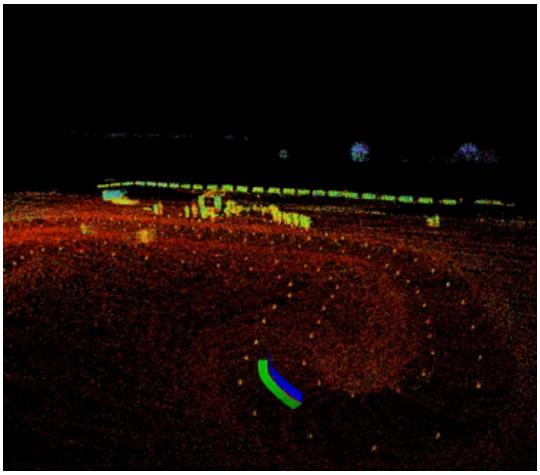

Figure 1.4: Detailed 3D map of the track (point cloud).

see Chapter 2 for further information. The perception stability and range has improved notably. With this change in the LiDAR cone detection phase, we noticed two things:

1. What **prevents** us from seeing further is now the **camera** FOV.
2. **Unmatched cones** (not seen by camera) may still be **correct**.

From these statements, it is clear that two new systems must be developed:

1. A cone classifier and tracker, in order to **classify** the cones (color and type) and improve **detection stability** (remove false positives).
2. A **color blind** track limits that will extend (where the color is lost) our partial-track color dependent track limits.

1.4 Objectives

These are the 4 main objectives that must be fulfilled throughout the project.

A - Classify each cone detected on the 3D map

Given:

- A **global map** of the environment.
- **Cone locations** in the map.
- **Cone bounding boxes** (detected by a neural network from real-time pictures of the track).
- Real-time **car state**.

Do:

A-1 **Synchronize** the car position with the cone bounding boxes.

A-2 **Register** the cone's bounding boxes to every cone found in the global 3D map and determine the class (color and size) of every possible cone.

A-3 Detect and **remove** the cones that were **miss detected** (false positives).

A-4 Stabilize and **keep track** as well as possible the cone observations.

B - All-track color blind track limits

Given:

- Classified **cone detections** $(x, y, z, type)$.
- Real-time **car state**.

Do:

B-1 **Extend** partial color dependent track limits with **unmatched** cones, i.e. cones classified as *unknown*.

B-2 When the entire track is seen, compute the **full track limits** (closing the loop).

C - Improvement

Achieve a real (measurable) improvement in cone detection precision and vision range compared to the current system. Quantify it.

D - Real time

The aim of this project is to run the proposed systems in a Formula Student single-seater during competitions, there the velocity of the car is key; system's latency is very important. In Formula Student, no matter how robust and precise an algorithm is if it takes too long executing.

Chapter 2

Fundamentals

2.1 Definitions

2.1.1 Point cloud

It is a set of points in space ($\mathbb{R}^3$). If it has been obtained from a LiDAR often these points also contain information about the intensity that the ray was reflected with, the beam number or timestamp. As images in ($\mathbb{R}^2$), point clouds are used to represent objects or environments digitally.

2.1.2 LiDAR

Light Detection and Ranging, it is a device which allows to measure the distance between a laser transmitter and a targeted object by measuring the time for the reflected light to return to the receiver. Usually they transmit multiple laser beams in a short period of time and so a point cloud is obtained. Current street car's LiDARs are rotatory, i.e. they produce a 360^{o} point cloud. While having a great range and resolution this type of LiDARs have a low output frequency $\approx$10 Hz. An example of a point cloud provided by our LiDAR can be seen in Fig. 2.1.

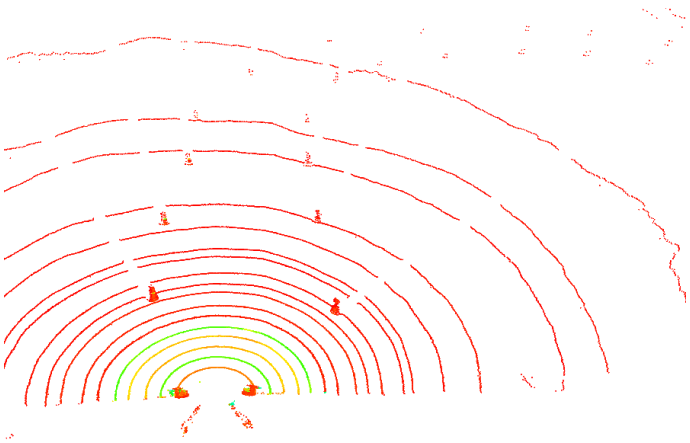

Figure 2.1: Point cloud obtained from our LiDAR.

2.1.3 YOLO

You Only Look Once, initially introduced by Joseph Redmon in 2016 [6], is a state-of-the-art neural network architecture for real-time object detection system in images. In our case, these detections have the form of bounding boxes and include the object's type t_{bb} and the detection confidence α_{bb}, that is: $\{x_{bb_0}, y_{bb_0}, x_{bb_1}, y_{bb_1}, t_{bb}, \alpha_{bb}\}$. See Fig. 2.2.

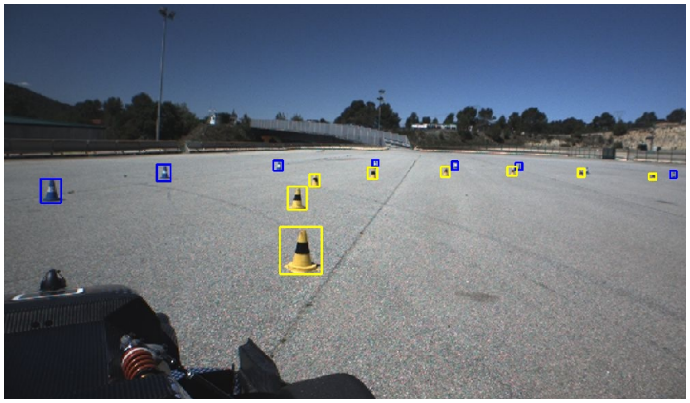

Figure 2.2: Our YOLO cone detection.

2.1.4 Track limits

The track limits (TLs) are the boundaries of the track, we define them as two cone traces (set of points, T), sorted by its order from a start point until no more track is seen or the loop is closed.

$$\text{TLs} = \{T_{left}, T_{right}\}$$
$$T = \{\, p_0, \ldots, p_{k-1} \mid i < j \iff \text{car will reach point } p_i \text{ before } p_j \,\}$$

Track limits are very important since they delimit the area that the car must drive in at all time. They are then passed to a control pipeline which compute the path that the car should follow inside these limits.

A track limits algorithm is a system which takes cones (positions) as an input and returns the computed track limits. We distinguish two types:

- **Class (color) dependent:** The track limits are computed from classified cones. According to FSG rules [1], the right trace will be formed by small yellow cones whereas the left will have small blue cones. It is much easier and robust to compute them using also cone color information.
- **Class invariant (color blind):** The track limits are computed from unclassified cones, i.e. geometrically. This is more risky since cone detections are not 100% believable, misdetections and false detections occur. The track limits should be robust to everything that can happen, the objective is to continue until the end of the test.

2.1.5 Global 3D map

We understand as global, everything that is referenced to the car starting point in the track $(0, 0)$ and does not have pre-established limits. Consequently, a global map is a map of the track environment with the coordinate center at the car's starting point. When talking about 3D, the map must also have information about depth and height of whatever is in it.

Our SLAM solution, LIMO-Velo, provides a point cloud of the track's environment (see Fig. 1.4), i.e. a global 3D map, where cones can be detected from.

2.2 Cone detection

Over the past of the years, we have seen that using only a particular sensor type to detect cones is not a good idea.

We have two separate cone detection techniques, one with cameras and another with the global 3D map obtained from LIO.

2.2.1 Global 3D map - Point cloud

The global 3D map is a detailed point cloud of the track. In this point cloud, cones are well-defined with a high point concentration. The objective is to detect the position of all cones, i.e. $\{(x, y, z)\}$. To reduce this problem, the map is divided into a 2D grid, with each cell storing its respective points. The detection system is divided into 3 phases:

1. **Ground removal:** The LiDAR is positioned on the car's front wing, which is very low to the ground. A lot of laser beams aim to the ground, as a result, there is a lot of noise on the map that is entirely ground. To remove the ground, its homogeneity in each cell is assumed and the points below a certain height percentile are removed.

2. **2D convolution:** A key feature that cones have is that there is nothing around, cone positions are detected by using an image inspired lax 2D top view convolution.

3. **Classification:** In this phase, it is very likely to have a large number of false positives, to get rid of them, a reconstruction of each possible cone is passed through a descriptor-based support vector machine that classifies them.

2.2.2 Cameras - RGB image

To detect cones in a RGB image, we use a state-of-the-art technique, a YOLO v4. This neural network takes images (1024 x 768 px) and outputs a bounding box on each cone, this bounding box does also contain a type, in this case the cone type of all used in a Formula Student competition (see DE 6.2.4 [2]).

The network is trained in-house with the FSOCO [7] dataset. We achieve human-like recognition in natural light conditions. The results are quite astonishing although the global position is not given (only position is image). See Fig. 2.2.

Chapter 3

Methodology

3.1 Baseline

The baseline for the cone classifier is the system that is currently in use on the car.

This system was introduced by Arnau Roche (team alumnus) and is called BB2L [8] in the team. It takes into account only one frame of LiDAR (a complete 360^{o} rotation) and a camera image with its bounding boxes extracted from a YOLO neural network. It projects each bounding box to the corresponding LiDAR 2D position and makes a match between this bounding box and a LiDAR cone detection, see Fig. 3.1.

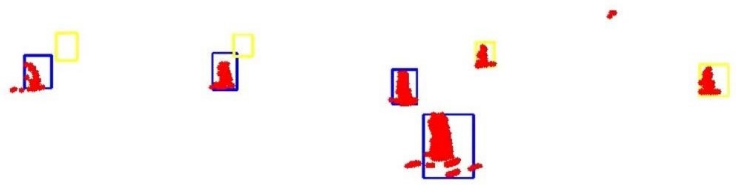

Figure 3.1: LiDAR point cloud to image projection (BB2L).

Then, to improve stability and accuracy, each match is tracked over time and hence the cone position and color tend to be the real ones using FastSLAM [9], see Fig. 3.2.

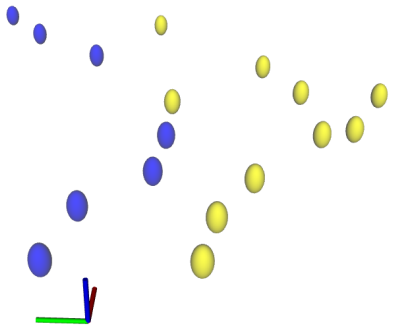

Figure 3.2: FastSLAM markers.

Advantages:

- Almost no false positives: it is a redundant system which combines the information of two sensors. "There will be a cone only if both sensors detect it".

Disadvantages:

- Range: since it uses only one LiDAR frame and one camera image at each time to detect cones, cones far away get few points in a LiDAR scan and it is impossible to distinguish them from noise.
- Stability: although there is a mechanism to improve it over time, it is not very stable.

3.2 Metrics

The objective of this project is to enhance the performance of the perception system of this season's car. There is, of course, a system running currently on our car. This is the system to beat. To quantify the improvement, these metrics can be considered:

- **Stability of a detection position:** whether or not the of a detected cone is constantly fluctuating between iterations. This can be quantified as the standard deviation.
- **Stability of a detection type:** whether or not the type (color and size) of a given cone changes between iterations. Quantified as the time it takes to finally stabilize into the true type. It may also be quantified as the number of times it changed from one type to another.
- **Precision of a detection:** the Euclidean distance between a real cone position in space (x_r, y_r, z_r) and the estimated position of the cone (x_e, y_e, z_e).
- **Latency:** when running a real-time program, latency is crucial. On top of that, if we want our car to go faster, we need to "see" the cones before it is too late. Latency will be the time from when the sensors give information until this system outputs the cones location estimation.
- **Range:** the distance beyond which it is impossible to localize the cones.

3.3 Project planning

3.3.1 Time planning

This project has a duration of roughly 4 months. The planning made considers the presentation date to be the 27th of June, so everything has to be ready by then.

I am the only person working on this specific project, so all time intervals are adjusted to this constraint. The car will compete around Europe this summer, specifically in July and August, so this is not a constraint.

Tasks

All the project's work is divided into 3 phases plus a continuous phase.

Phase 1: Planning and GEP

This phase contains all the tasks related to the project's consideration, tasks and objective planning. How do I end up in June with a worthy thesis?

- **Define the objectives and the scope [31-Jan to 13-Feb]:** The scope of the project as well as all the objectives are defined and clarified, so afterwards it is easier to divide them into sub-objectives. This part is crucial as all the following work is based on this.
- **Define the methodology and the tools that will be used [14-Feb to 20-Feb]:** Define how the objectives are accomplished. Which tools are used and why. Risks here are to choose the wrong tool and realize when it is too late.
- **1st Assignment GEP: Scope and context definition [21-Feb to 27-Feb]:** The scope and the context of the thesis are clarified as well as the objectives. GEP is an opportunity for non-experts in project management to learn the basics.
- **2nd Assignment GEP: Temporal planning [28-Feb to 6-Mar]:** Task timeline is provided in order to follow a pre-established plan and to have the thesis just in time. Risks are to make an unreal plan and create the need to modify it later on.
- **3rd Assignment GEP: Economic Management and Sustainability [7-Mar to 13-Mar]:** The financial aspects of the project are evaluated. Is this project profitable? Or is it be sustainable? A complete sustainability plan is proposed. Here, the main risk is not to meet these financial and sustainability objectives by the end of the project.
- **Final Assignment GEP [14-Mar to 20-Mar]:** A document regarding each and every aspect of the project structure is hand-in.

Phase 2: Implementation

The heavy work is carried out during this phase. At the end of it, the main objectives will be met.

- **Get to know the data and experiment with ROS [21-Feb to 6-Mar]:** The objective of this task is to gain confidence in the environment and the data types that are used. Point clouds and bounding boxes are data types that I need to master in order to make an efficient and useful algorithm to meet the project's objectives. ROS specifically is used throughout the work.
- **Synchronize the car position with the camera bounding boxes [7-Mar to 20-Mar]:** In order to meet the two big objectives, this has to be

achieved as well. Since the car is constantly moving around the track, having a system capable of synchronizing the car location with the camera bounding boxes is key. Here, the main problem that can occur is that the synchronization precision is not high enough to register the bounding boxes to the map when the car is moving very fast.

- **Register the cone's bounding boxes to every cone found in the global 3D map and determine the class (color and size) of every possible cone [21-Mar to 24-Apr]:** This is equivalent to completing the objective 1.4.

- **Compute the entire track limits using every detected cone, either classified or not [9-May to 5-Jun]:** Another big objective, to compute the track limits earlier taking into account not-classified cones, this is objective 1.4. The main risks are cone misdetections and false positives (both will alter track's shape).

Phase 3: Validation and further improvements

The correct functionality of the solution proposed is checked. The algorithms are validated in our Formula Student car.

- **Test the algorithms in-car [6-Jun to 19-Jun]:** A very important integration task, see if the algorithms proposed work in our new car. Algorithms can work with recorded data but when integrated on the car and combined with the control algorithms, unexpected effects can appear.

- **Quantify the improvement with respect to the methods used before [6-Jun to 19-Jun]:** No project is a success (in Formula Student) if the solution made is worse than what existed before, thus the solution has to improve the actual techniques. The risk here is not being able to develop better systems.

- **Identify further improvements to the algorithms proposed [6-Jun to 19-Jun]:** Defining a path to further improvements is key to the next projects that will be made in this area. Autonomous driving is (and will continue) booming, so by doing small steps we can help to create the car of the future.

Continuous work

All the secondary work that is carried out during the whole process of implementation.

- **Report up-to-date every 15 days (1st and 15th of every month) [14-Mar to 26-Jun]:** The written report must not be forgotten. Hence, every 1st and 15th of every month, an update to the report is carried out.

- **Presentation slides [20-Jun to 26-Jun]:** Presenting well-thought slides to the jury is a last step not everybody thinks about and is very important, since the jury must understand every aspect of the project.

Gantt chart

To better comprehend the project scope, it is useful to have a Gantt chart to see how good one is doing at every moment. The Gantt chart in Fig. 3.3 shows the tasks listed above with the corresponding dates.

MASTER PLAN	ACTIVITY	START	END
Phase 1	Planning and GEP		
	Define the objectives and the scope	31-Jan	13-Feb
	Define the methodology and the tools that will be used	14-Feb	20-Feb
	1st Assignment GEP: Scope and context definition	21-Feb	27-Feb
	2nd Assignment GEP: Temporal planification	28-Feb	6-Mar
	3rd Assignment GEP: Economic Management and Sustainability	7-Mar	13-Mar
	Final Assignment GEP	14-Mar	20-Mar
Phase 2	Implementation		
	Get to know the data and experiment with ROS	21-Feb	6-Mar
	Synchronize the car position with the camera bounding boxes	7-Mar	20-Mar
	Register the cone's bounding boxes to every cone found in the global 3D	21-Mar	24-Apr
	Detect the cones that were miss detected and remove them	25-Apr	8-May
	Compute the entire track limits using every detected cone, either	9-May	5-Jun
Phase 3	Validation and further improvements		
	Test the algorithms in-car	6-Jun	19-Jun
	Quantify the improvement with respect to the methods used before	6-Jun	19-Jun
	Identify further improvements to the algorithms porposed	6-Jun	19-Jun
Continuous	Final assignments		
	Report up-to-date every 15 days (1st and 15th of every month)	14-Mar	26-Jun
	Presentation slides	20-Jun	26-Jun

Figure 3.3: Project's Gantt chart.

3.3.2 Financial planning

Every professional project must have an estimated budget at the beginning, and this is no exception. The costs of the project are broken down into Staff Costs, Fixed Costs and Variable Costs. The Staff Costs are the time that a qualified person (me) will need to accomplish all the work. The person will be earning 10 €/h and will be doing the tasks specified in the planning.

It is important to emphasize that no material costs are taken into account because to make this project, all the sensors and tools needed will be provided by the company. Also, the energy costs are approximated.

The contingency level will be 15%, as it is an approximate value of the confidence I have in realizing the project on time and without any inconvenience. The obtained amount is 5494.08 €.

In Figure 3.5, there are the three risks I think are most likely to happen. As I will work with a LiDAR sensor (which is very expensive) there is a possibility that I drop it or I damage it accidentally, the same with cameras, but reducing the probability because they are normally static on the car.

Also, when coding to solve complex tasks, if for some reason, I realize that the computer we have is not powerful enough, we might buy a new one.

Taking into account all these factors, we end up with a total budget of 6204.08 €.

3.3.3 Management control

During the project's development multiple costs analysis will be carried out periodically in order to anticipate a possible budget dis-adjustment and to make sure that we do not exceed it. These steps [10] will be followed:

Activity	Nº of hours	€/hour	Amount	
Staff costs				
Get to know the data and experiment with ROS	40,00	10,00	400,00 €	
Synchronize the car position with the camera bounding boxes	40,00	10,00	400,00 €	
Register the cone's bounding boxes to every cone found in the global 3D map and determine the class (color and size) of every possible cone	100,00	10,00	1.000,00 €	
Detect the cones that were miss detected and remove them	40,00	10,00	400,00 €	
Compute the entire track limits using every detected cone, either classified or not	80,00	10,00	800,00 €	
Test the algorithms in-car	40,00	10,00	400,00 €	
Quantify the improvement with respect to the methods used before	40,00	10,00	400,00 €	
Identify further improvements to the algorithms porposed	40,00	10,00	400,00 €	
Fixed costs				
Material costs (computer amortisation)	420,00	-	168,00 €	(2000€ / 5 years, 4h / day, 250 working days)
Variable costs				
Electricity	420,00	0,08	33,60 €	(supposing 200 W, 0.4 €/Kwh)
Natural Gas	180,00	0,52	93,60 €	(supposing until May, spending 4 Kw and 0.13 €/Kwh)
Rental of the space	420,00	0,67	282,26 €	(supposing 500 €/month, 31 days/month)
		Subtotal	4.777,46 €	
Contingency level (15%)	-	-	716,62 €	
		Total	**5.494,08 €**	

Figure 3.4: Project's budget.

1. Establish actual position: Collect all the information possible about the budget and the money spent / available.

2. Compare actual with budget: The information gathered needs to be compared to the budgeted figures, which items were completed? at what price? which are left?

3. Calculating variances: A variance is the difference between the actual and budgeted income and expenditure. Variances can be calculated absolutely (€) or relatively (%).

4. Establish reasons for variances: The reasons for all variances need to be identified as quickly as possible to prevent an increase. These can be due to unplanned costs, delays in product acquisition, incorrect figures, etc.

5. Take action: Cut costs with some unnecessary items or get the items from other sources.

3.4 Sustainability and social commitment

3.4.1 Viability

This project intends to find an improvement for the perception system of a formula student driverless car. This can be extrapolated to street cars.

Risk	Cost	Probability	Amount
The LiDAR sensor breaks	10.000,00 €	5,00%	500,00 €
An RGB camera breaks (x2)	500,00 €	1,00%	10,00 €
The computer is not powerful enough (buy a new one)	2.000,00 €	10,00%	200,00 €
		Total	710,00 €
			Total Budget
			6.204,08 €

Figure 3.5: Risk analysis and total budget.

Our society is not far away from seeing autonomous cars on the roads, it will be very important to equip them with a robust and precise computer vision system. It cannot be argued that most automotive companies are still developing driverless cars, since current technologies are not good enough for street driving.

With the improvements that this project may bring, we could help **accelerating** the transition towards self-driving cars. Saving **time** and **money** to companies.

Autonomous cars will change transportation for ever. It will be **safer** (cars can broadcast its intentions to others) so a group decision can be taken.

Also, with driverless vehicles, transportation of people or goods could be more **economical** (no need to pay a driver).

3.4.2 Analysis of sustainability

Everybody has an image in their mind when thinking about autonomous cars, a car full of expensive sensors, see Figure 3.6. Most driverless cars use multiple LiDARs to estimate its position inside a pre-recorded map (or point cloud) and to estimate the position of close dangerous objects. This is not sustainable, in the face of the impossibility of a software capability to sense the environment with few inexpensive sensors, multiple complex sensors are used to externalize the computation. In addition, these are not cheap, the price of one of these cars can reach more than 100k €.

We will propose a combined solution (in a formula student car) that will involve only one LiDAR and two cameras to compute the limits of the track.

This project will be carried out on a full electric, driverless vehicle. This entails that the possible CO2 emissions will be minor and no consumables will be spent.

Driverless cars are way more sustainable than human-driven vehicles. A computer can process more data than humans do, as a result, the driving is more smooth and efficient, decreasing the brakings and the accelerations.

There is also the advantage that driverless cars can communicate among them and with the environment, so for example, if there is a traffic jam, the cars that are already stuck can precisely warn those that are not and they can adapt their velocities to the situation. Another example is intersections, in an intersection we need a traffic light (that results in brake and acceleration maneuvers), but with

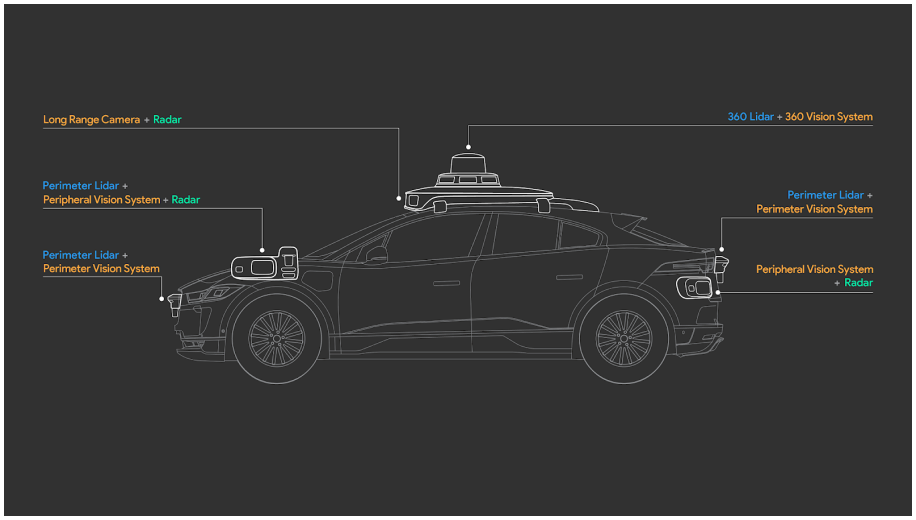

Figure 3.6: Waymo autonomous car sensor layout.

inter-communicated cars there can be no traffic lights and still no problem caused, as they will have a precise pre-computed trajectory to follow.

Personally I think that I will grow as a computer engineer, I will learn to work with powerful tools such as ROS and LiDARs, and that will make me more prepared for future work. Having the possibility to improve a specific aspect of the robot on-wheels of the future is highly rewarding. At the end of the day, this sector is constantly evolving and will soon have prepared street driverless cars.

3.5 Design considerations

When designing a system for a very specific use case like Formula Student, several design considerations must be taken into account.

3.5.1 General requirements

- **Easy to understand code:** The main objective of Formula Student is to learn, therefore code must be clean and easy to understand so that future generations can maintain and upgrade it. In this manner, time spent on understanding (and modifying) the code by third parties will be minimal.

- **Computational power:** The proposed systems will be running on the car's main computer, as this machine is shared with other systems such as the vehicle control's algorithms, we cannot build a system that requires all the resources. In addition, Objective D - Real time must be met.

3.5.2 Hardware architecture

Sensors

The car is equipped with multiple sensors that will allow the car to guess its actual and next states. Here, not only purely perception sensors will be used, but also IMUs.

- **1x Velodyne VLP-32C LiDAR**
  - 32 channels
  - Spinning at 10 Hz
  - 1.2M point/s
  - V. FOV: -25 to 15 deg
- **2x The Imaging Source DFK 33UX252 RGB USB 3.0 cameras**
  - 1024 x 768 px of resolution
  - 30 fps
  - 4.5 mm of focal length $\Rightarrow$ 80 deg FOV
- **2x SBG Ellipse-N INS/GPS**
  - 9-DOF IMU
  - 200 Hz
  - EKF to estimate attitude
- **1x Vectornav VN-300 INS/GPS**
  - 9-DOF IMU
  - 400 Hz
  - EKF to estimate attitude

Computer(s)

This project's proposed solution will run on the car's computer system. This involves two computers connected to each other through Ethernet.

- **Processing Unit**
  It is an in-house designed Intel computer. It is the main ECU on the car, responsible for maintaining the car's state (see rule T 14.10.2 [1]), running low-level control algorithms such as torque vectoring, and of course running the demanding perception and autonomous control algorithms. Technical specifications:
  - ASRock IMB-1222-WV motherboard
  - Intel i9 10900 10-core 64-bit CPU

- 16 GB SO-DIMM DDR4 2666 MHz RAM
- Ubuntu 20.04 LTS

The CPU stands out for its performance as well as for its number of cores. Allowing 20 threads running simultaneously, this project's system must take advantage of them through the use of parallelism.

- **Nvidia AI accelerator**
  This computer is an Nvidia Jetson Xavier AGX. On this computer, any task concerning a neural network is executed. Technical specifications:

  - 32 TeraOPS of AI performance
  - 512-core Nvidia Volta GPU with 64 Tensor Cores
  - 8-core Nvidia arm 64-bit CPU
  - 32 GB LPDDR4x
  - Ubuntu 20.04 LTS

  On the day of writing this report, it is only used to detect cones in the images given by two RGB cameras.

3.5.3 Software architecture

Programming language

Taking into account the real time requirement of the application, an efficient (compiled) programming language must be used. Another constraint should be considered when working with point clouds and images: community support and libraries. C++ is what fits best the demands. It is widely used in computer vision and it is one of the languages that the chosen framework accepts.

Modular programming

Modular programming is a method to separate the functionality into independent interchangeable modules, this way, if some feature needs to be modified, only the feature's module will result affected.

k-d tree

When working with cones (points in $\mathbb{R}^3$), it is a usual problem to find the closest cone to another. Therefore an optimized data structure to search in a queried position's neighborhood is needed.

k-dimensional tree [11], is a multidimensional binary tree widely used for organizing points in a k-dimensional space. If n is the number of elements, the tree is built in $O(n \log n)$ and each search is performed in $O(\log n)$.

Traditionally, the cost of finding the closest element is $O(n)$.

3.6 Tools

All the work is carried out during my journey in the Barcelona UPC Formula Student team (BCN eMotorsport), where I work alongside my department colleagues until I fulfill my objectives and we achieve a precise, robust and fast perception system.

My two directors guide me throughout the project with periodic weekly meetings.

ROS

All data will be synchronized and processed and using ROS, specifically version *Noetic Ninjemys*. ROS aims to be an operating system to manage asynchronous processes and data input. Its topic-message architecture, see Fig. 3.7, allows the computer to wake up programs only when they have the required data and not waste any CPU while data is unavailable.

ROS is what manages all car information in the main ECU, does all the heavy computation and commands the driverless car. These commands are then transmitted through the CAN bus to the actuators.

It is an essential tool in our car and will be used to accomplish the objectives described in this document.

- Free and open source.
- Multi-lingual programming (Python and C++).
- Data visualization tools (RViz).
- Compatible with computer vision libraries (OpenCV and PCL).
- Integrated tf library: dynamic reference frames transformation.
- Timestamped data recording and playback of topics: be able to simulate all sensor's data later on (rosbag). It is very frequent to work with rosbags, they are a way of testing our algorithms with real data while being off-the-track.

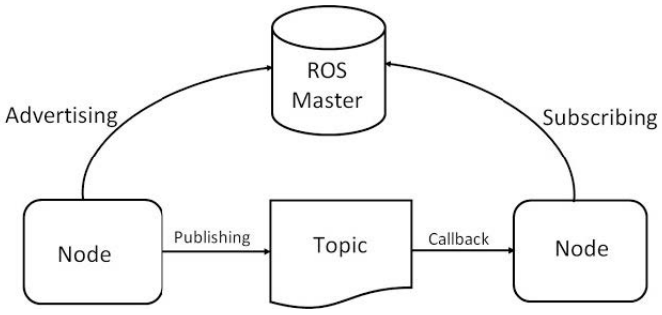

Figure 3.7: ROS topic-message architecture.

Git

Git is a version control system that enables the user to save and access different versions of a project. It offers additional features such as merge versions and to easily upload the changes to a git repository online so other users can access it as well. All the code is uploaded to the team's GitLab server so everybody has access and can download and modify the programs.

Chapter 4

CCAT: Cone Classifier And Tracker

4.1 Purpose

CCAT is a system built for the Formula Student team BCN eMotorsport in order to **classify** and **keep track** of the cones detected from a global 3D map.

To fulfill this purpose, CCAT implements a **sensor fusion** model. It **registers** the bounding boxes obtained from the neural network that processes the camera images to the cones detected on the 3D map that LiDAR odometry provides.

CCAT must obtain a vector of classified cones. Each cone should have a type t_{ccat} according to the class of cone they belong to. According to FSG DE 6.2.4 [2], possible types are (see Fig. 4.1) *big orange* (BO), *small orange* (SO), *small yellow* (Y) and *small blue* (B). In order to also represent the case where cone type is unknown (due to camera range or uncertainty), type *unknown* (UNK) will also be a possible type.

Big Orange *(BO)* | Small Orange *(SO)* | Small Yellow *(Y)* | Small Blue *(B)*

Figure 4.1: Possible cone types in FSG.

In addition, CCAT will make sure (as far as possible) that the cones it outputs agree with the ground truth. To do so, it will track every cone over time to robustly assign them a unique id_{ccat}.

We will set/change the decisions on the set of tracked cones only when a certain confidence is reached since, later, other algorithms will take these cones (and identify them through the id_{ccat}) and that will decide which path will the car take.

4.1.1 Input

- An **observation vector** from the global 3D map, this includes the cone position (centroid) p_{obs} ($\mathbb{R}^3$), a detection confidence α_{obs} and the cone's point cloud pcl_{obs}.
  This is: $\{(p_{obs}, \alpha_{obs}, pcl_{obs})\}$.
- Car **location** and **heading**, a pose 6D vector.
  This is: $(x, y, z, \phi, \theta, \psi)$.
- Left and right camera detected **bounding boxes** of cones, including the type of cone and confidence.
  This is: $\{(x_{bb_0}, y_{bb_0}, x_{bb_1}, y_{bb_1}, t_{bb}, \alpha_{bb})\}$.

Note: In order to synchronize the data afterwards, all input messages are timestamped.

4.1.2 Output

- **Classified** cone vector with class confidence and each cone will have a unique id (between iterations).
  This is: $\{\, (p_{ccat}, t_{ccat}, id_{ccat}, \alpha_{ccat}) \mid p_{ccat} \in \mathbb{R}^3, t_{ccat} \in \{Y, B, SO, BO, UNK\} \,\}$

4.2 Core ideas

CCAT is based on the following ideas in order to fulfill Objective **A**.

4.2.1 Latest data is more representative

When receiving 3D map detected cones, the more time it passes by, the more defined and larger the map will be. Consequently, the positions detected with this more refined map will be more representative (more accurate and farther range).

4.2.2 Pinhole camera model

The pinhole camera model helps registering cameras' bounding boxes (image space) to LiDAR seen cones (map space). It relates 3D world coordinates with image coordinates through the use of camera parameters (intrinsics and extrinsics), see Fig. 4.1. The extrinsic parameters, or camera pose, $E = \{x, y, z, \phi, \theta, \psi\}$ lead to a transformation in $\mathbb{R}^3$ from global (map) to camera coordinates, often expressed as a transformation matrix (r and t) directly. The intrinsic parameters $I = \{f_x, f_y, \sigma_x, \sigma_y\}$ relate camera coordinates to image coordinates, f_x and f_y are the pixel focal length (equal for squared pixels), σ_x and σ_y are the offsets of the principal point from the top-left corner of the image, approximately $\sigma_x = width/2$ and $\sigma_y = height/2$.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & \sigma_x \\ 0 & f_y & \sigma_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.1}$$

where:

X, Y, Z are coordinates in camera reference

u, v are coordinates in image reference

λ is a scaling factor

Cameras' intrinsics depend on the sensor and the lenses used. In our case, they were found by a team's ex-member Eloi Bové [12] using a checkerboard pattern.

4.2.3 Statistical model

The reason

There will be a matching process involving the cone centroid and the cone bounding box in an effort to obtain the type of cone and confirm its authenticity (true positive).

Due to sensors' synchronization and cameras' extrinsic parameters not being accurate enough, an offset (displacement in image space) usually appears between the detected cone bounding box (camera) and its centroid (LiDAR). This offset is accentuated linearly with velocity (specifically angular velocity ω) and quadratically with the cone to camera plane distance.

This means that **potentially wrong matchings** will be obtained when:

- The car is moving **fast**
- The observation involved is **far**

The proposed model will need to take these considerations into account and be robust to mismatches in order to converge to the correct type over time. This is a **statistical model**.

We still have another challenge for miss-detected cones, Objective A-3, but it is the same problem inversely. Detect statistically which cones are false positively detected, and consequently, which ones do converge to not have a matching.

A heuristic-ponderated polling system

Each cone will be matched at most one time per iteration. The system will save the n best matchings according to a heuristic h_m and then the statistical mode, namely the most voted type t will be output.

$$\{\, (h_{m_0}, t_0), \ldots, (h_{n-1}, t_{n-1}) \mid i < j \iff h_{m_i} < h_{m_j} \,\} \tag{4.2}$$

h_m is calculated as follows:

$$h_m = \frac{1}{d_m d_{cp}} \tag{4.3}$$

where:

$\boldsymbol{d_m}$ (px) is 2D Euclidean distance between cone bounding box and the projected cone centroid.

$\boldsymbol{d_{cp}}$ (m) is the 3D Euclidean distance between cone centroid and the camera plane.

Similarly, to **eliminate false positive** cone observations, the same system also keeps track of unmatched observations. The ratio matched/non-matched observations of each cone is calculated at every moment, the cone will be considered a false positive (and hence not valid) if this ratio is below a certain value and the distance from the cone to the camera plane is below a threshold (to guarantee that it does not exist).

4.2.4 3D greedy matching function

The matching function will be in charge of relating each map-observation with the corresponding bounding box.

In order for the statistical model to work as expected, the matchings given at every iteration must be as accurate as possible. We do not want to maximize overall matching accuracy in an iteration but to **maximize** each cone's **accuracy over time**.

To achieve this commitment, in every iteration each map-observation will be paired with the **closest** bounding box.

However, this search must be performed in $\mathbb{R}^3$. That is because a particular image coordinate ($\mathbb{R}^2$) gets transformed into a line (and not a point) in global $\mathbb{R}^3$ coordinates.

Think about having two LiDAR observed cones c_0 and c_1 whose centroid gets projected at the same image point (u_0, v_0), and two bounding boxes centered at the same point (u_0, v_0), one bigger than the other. Which bounding box belongs to which cone?

To solve this issue, the search is performed in 3D, being the two first coordinates the image point (u, v) and the third coordinate the height of the bounding box from camera, and for the global map observed cone, its estimation of bounding box height (obtained with pinhole model). In other words, if we have a bounding box centered at (u_{BB}, v_{BB}) with height H_{BB} and a global map detected cone with its projected centroid at (u_c, v_c) and an estimation of the height of the bounding box H_c, we will compare (u_{BB}, v_{BB}, H_{BB}) with (u_c, v_c, H_c).

4.3 Pipeline

In order to achieve the desired functionalities, the system pipeline is divided into modules to improve code readability and ensure maintainability. As shown in Fig. 4.2, there are 5 different modules, each one performs a very specific task and is designed to not have any dependency with other modules.

4.3.1 Manager

Description

The Manager module acts as a master module; its job is to manage all incoming data and call inner pipeline with the appropriate-synchronized data.

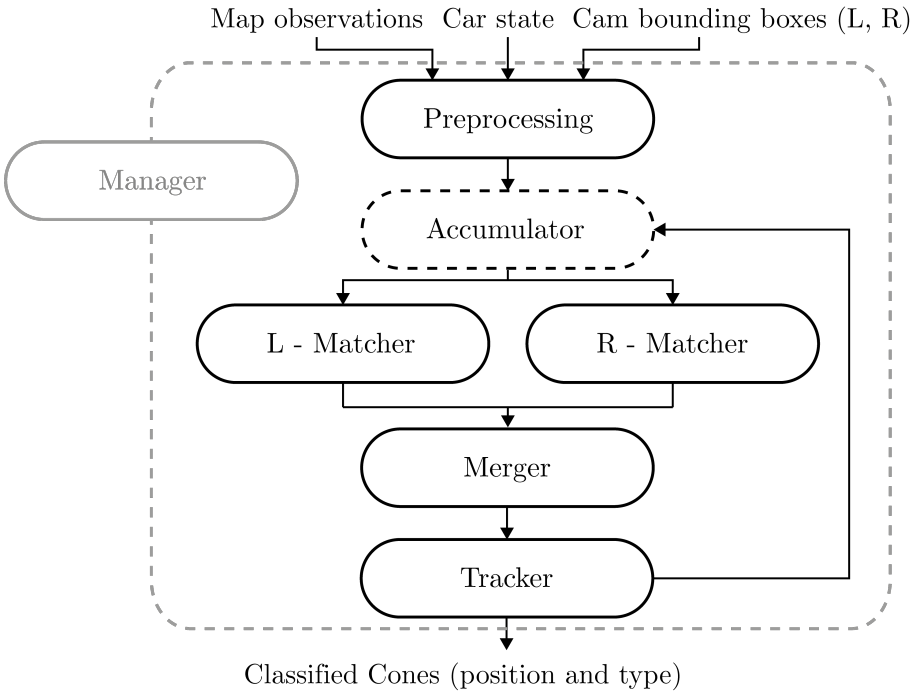

Figure 4.2: CCAT pipeline.

Tasks

1. **Synchronize** all input data.
2. **Call** inner pipeline modules with valid data **only** when necessary.
3. Runs all Matchers modules in **parallel** with OpenMP [13].
4. Allow real-time camera extrinsic parameters calibration on rosbag.
5. Allow **one camera only** operation, e.g. a camera gets disconnected.
6. Allow **LiDAR only** operation, i.e. LiDAR gets disconnected.

4.3.2 Preprocessing

Description

This module preprocesses all data that arrives in each iteration. The aim of this module is to filter and clean the data so all the other modules can use it directly.

Tasks

1. **Convert** car's pose to a transformation matrix to later transform all cones to car's local frame. Make the **inverse** of this matrix (transform global-local is wanted) and **invert** y coordinate.

2. **Cluster** in $O(n \log n)$ all incoming 3D map observations recursively using a threshold d_{clust} such that if the distance between two points is lower or equal to d_{clust}, they will get grouped into the same cluster, see Fig. 4.3. Cluster position will be the mean of all clustered cones position.

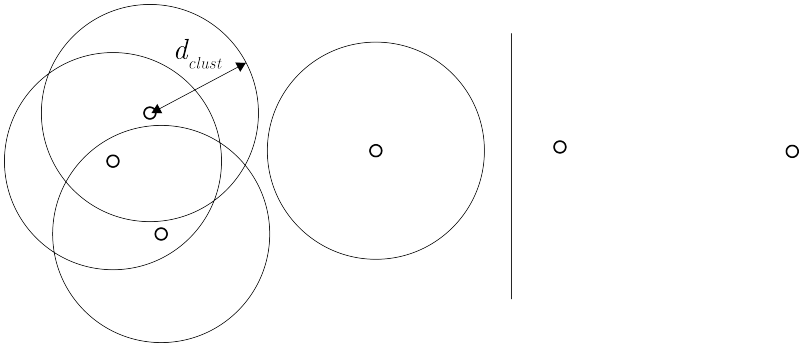

Figure 4.3: Clustering by distance.

4.3.3 Accumulator

Description

The Accumulator module is a fictitious module, created as part of the Tracker module. It is a necessary step between Preprocessing and Matcher(s) since we want to match all observations (historic and new). This module relates and adds current iteration's observations to the historical (accumulated since the beginning).

Tasks

1. **Accumulate** LiDAR **observations** with existing tracked cones such that for every new observation obs_i with position p_{obs_i}:
   - If there is an existing cone c_j with position p_{c_j} and $dist(p_{obs}, p_{c_j}) \leq d_{acc}$, it is assumed that obs_i is another observation of c_j. In this case, only the position is updated (latest data is more representative).
   - Otherwise it is assumed that obs_i is the first observation of a new cone. In this case, a new cone with position p_{obs_i} is created on the system.
2. Give every cone a **unique id**: Every time a new cone is observed, a new id is given (an increment of last id given).
3. **Transform** every cone' position from global into **local** (with the use of car's location and heading).

4.3.4 Matcher(s)

Description

The Matcher module is responsible for the **projections** and **matchings** between the camera bounding boxes and the system's cones. In addition, it also keeps **camera**

parameters (intrinsics and extrinsics) and allow real-time extrinsics calibration.

Tasks

1. **Transform** every cone's local position into **camera space** through the camera extrinsic parameters.
2. **Transform** every cone's camera position into **image space** through the camera intrinsic parameters (pinhole model).
3. **Remove** all cones whose position is not **in-frame** (behind the car or not caught on camera).
4. For each projected cone, **match** it with the closest camera bounding box within a maximum distance threshold $d_{match_{max}}$ using a k-d tree in $O(\log n)$.
5. Save a list of **cone updates**, that is a list containing all cone *id*s and:
   a) matching distance (d_{match}) and matched type (t_{match}) if the cone has a matching.
   b) global distance to closest matched cone (d_{cl_match}) otherwise.

4.3.5 Merger

Description

The Merger module is in charge of merging the output of the two Merger modules. This is necessary because the two Matchers can output the same cone due to the small zone of overlap between the two cameras; i.e. the same cone can be detected by the two cameras.

Tasks

1. **Merge** all Matchers' output into one single list such that there are no two cones with same *id*.
2. If multiple cones with same *id*s are found, the one with a smaller matching distance (d_{match}) will prevail.

4.3.6 Tracker

Description

The Tracker module is the only module in which cones are stored between iterations. The objective of this module is to keep track of all observations and decide which one should be output and the type of it.

Tasks

1. **Store all cones**. That is all historical observations.
2. For every cone, **decide** whether or not it is an existing cone and its **type**.
3. **Convert** all data into ROS message data (serializable and readable by another node).

4.4 Additional features

4.4.1 Camera extrinsic parameters calibration

Manual calibration

In order to correctly transform a position from 3D global frame to image coordinates, the 6 extrinsic parameters must be really accurate. It is very common that the cameras move a little, thus an easy way to calibrate is required.

ROS provides what they call *dynamic reconfigure*, it is a method through which program variables can be modified while running.

1. **Define** a ROS *cfg* **message** that will contain a variables list including their type and range.
2. A **callback function** is defined, it will take a this predefined ROS *cfg* as a parameter and will modify the program variables that are wanted to.
3. **Register** the callback with ROS *dynamic reconfigure*.
4. Using ROS' *rqt_reconfigure* interface one can dynamically adjust each parameter (see Fig. 4.4) according to its effect.

With ROS *dynamic reconfigure*, all the 6 extrinsic parameters can be modified individually. The effect is immediate.

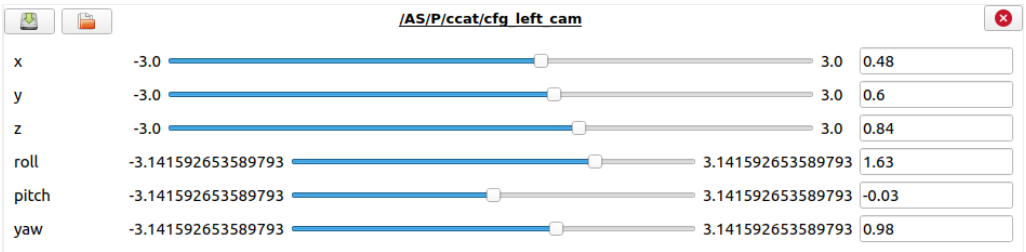

Figure 4.4: Manual extrinsics calibration with *dynamic reconfigure*.

Rosbag calibration

When working with rosbags, to validate or improve a specific algorithm, calibration is needed because camera extrinsics are often different depending on the day the rosbag was recorded (cameras move a little bit over time). Car has to be standing still to calibrate correctly, but in rosbag it moves.

- Cannot calibrate with rosbag playing, the car is moving.
- Cannot calibrate with rosbag stopped, no data is being received and the pipeline stops.

CCAT's solution is to loop in a single iteration while having the possibility to call ROS *dynamic reconfigure*.

Automatic calibration

Manually fine tuning extrinsic parameters is extremely complicated, i.e. it is very difficult to make far observations correspond to bounding boxes.

To solve it, the team developed an auto-calibration algorithm last season. It uses the matchings to find iteratively the 6 DOFs with a gradient descent of the distance between cone and bounding box centre.

When started, CCAT calls this automatic calibration with the matchings obtained from the default parameters and updates its parameters. It reruns this process until a certain mean squared error is reached. Only then, the Matcher(s) modules let information pass to next modules.

4.4.2 Operation in poor conditions

In Formula Student, teams have a limited amount of tries in a specific event, it is really important to have a robust system even when facing a sensor loss. In fact, the only data that is needed to maintain operation is the position and orientation of the car.

Camera loss

Cameras are connected via USB and are subject to vibrations all time. It is not rare that a camera gets disconnected abruptly. In this case, CCAT will bypass the affected camera's Matcher module.

LiDAR detections loss

CCAT will also maintain its normal operation when facing a detections loss. It will use the historic accumulated detections to continue running.

4.4.3 Data visualization

Visualizing all process steps is key when it comes to ensure working conditions or debugging. We can visualize the following through RViz [14]:

Cones projection

It continuously displays an image showing the bounding boxes and the projected cones.

- Shows the **range** of the perception detection for both YOLO and LiDAR map filtering.
- Shows the **type** of each camera detected cone (color of the bounding box).
- Shows if the **camera extrinsics** are well calibrated.
- Shows the type to which every cone is **matched** to (color of the cone point cloud).

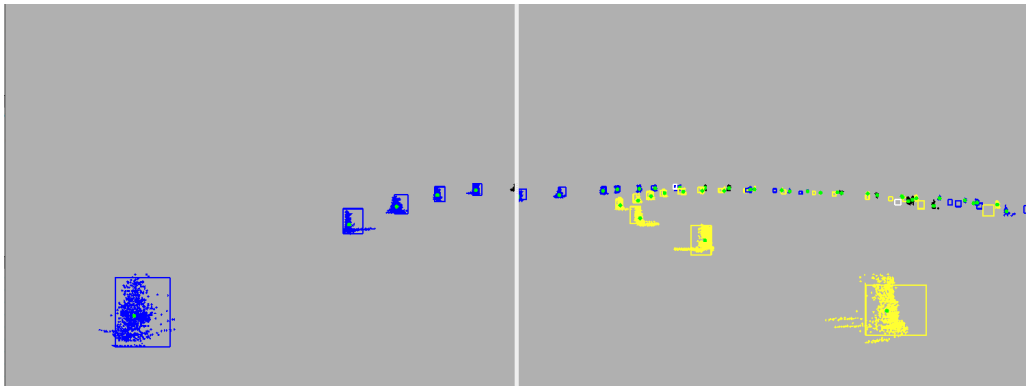

Figure 4.5: Cone projection (left and right cameras).

Iteration markers

To better visualize the quality of the matchings, markers are also provided. These markers are the result of matching accumulated cones with camera bounding boxes at a particular iteration.

Final cone markers

Finally, it is necessary to see what is the output that the system provides to measure the performance of the statistical model. In addition, each cone's id is also displayed on the cone position.

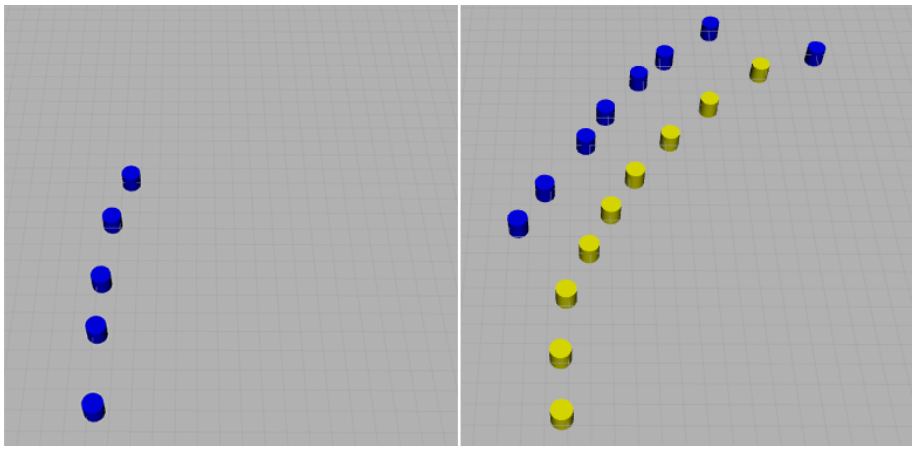

Figure 4.6: Iteration matchings markers (left and right cameras).

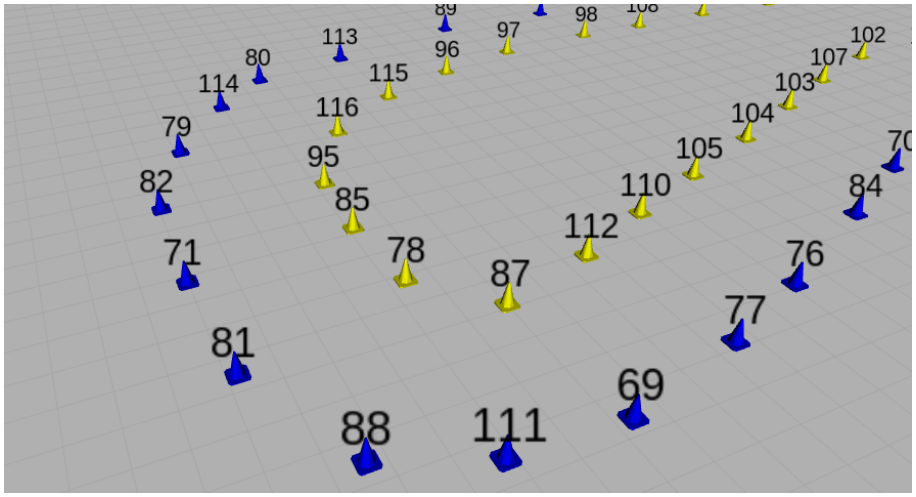

Figure 4.7: Final tracked markers.

Chapter 5

Urimits: All-track color blind track limits

5.1 Purpose

Urimits is a system built for the Formula Student team BCN eMotorsport that arises from the need to detect the track limits and **close the lap** as **earlier** as possible in order to optimize geometrically (taking into account the vehicle's dynamic model) the trajectory and velocity profile, this optimization can only be performed when the whole track is delimited.

The system carries out this task by using cones that CCAT has not been able to give a valid type yet, i.e. cones that are detected on the SLAM 3D map and not by camera. These cones have a high probability of being a misdetection (they are often far from the car).

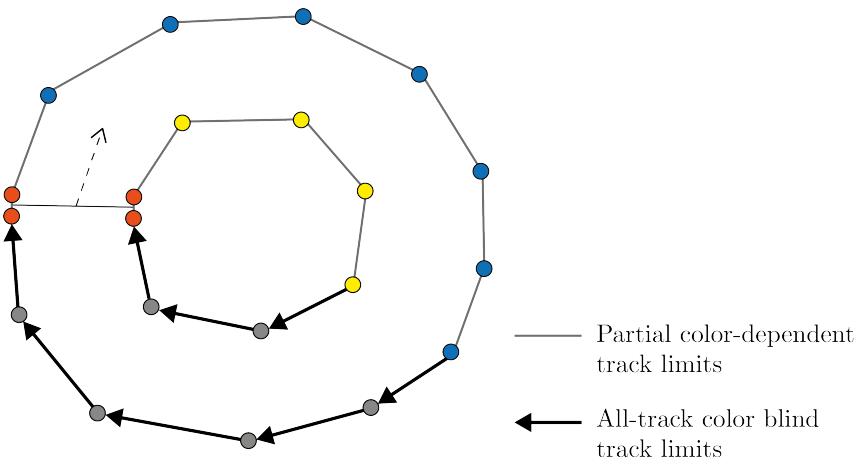

Figure 5.1: Urimits purpose.

As shown in Fig. 5.1, Urimits' starting point is partial track limit's ending point. This way, it will try to close the loop where color or shape is not stable enough for the polynomial comparison based track limits developed by a team colleague, Oriol Pareras.

5.1.1 Input

- **Classified** cone vector with confidence and unique id.
  This is: $\{ (p_{ccat}, t_{ccat}, id_{ccat}, \alpha_{ccat}) \mid p_{ccat} \in \mathbb{R}^3, t_{ccat} \in \{Y, B, SO, BO, UNK\} \}$
- **Partial** TLs computed with cones whose $t_{ccat} \neq UNK$ only.
- Car **location** and **heading**.
  This is: $(x, y, z, \phi, \theta, \psi)$.

5.1.2 Output

- A **valid TLs** (see 2.1.4) with the characteristic that the loop is closed, i.e. both traces have the same starting and ending point.

5.2 Core ideas

5.2.1 All-track

This track limits algorithm is all-track. This means that it either gives the limits of the whole track or it does not output anything.

This decision is based on probability. This color blind track limits algorithm delimits the track based on the **geometry** of the cone positions only. When not perceiving all the cones of the track, a partial solution using only the track geometry can lead to fake limits. If we can detect that the loop is closed, it has a **much higher** probability of being the genuine limits.

5.2.2 Greedy iterative approach

The core of this solution is an iterative process in which we start at partial track limit's trace final position and heading (x_0, y_0, θ_0). Here, all the process will be performed in $\mathbb{R}^2$, assuming that the track is flat.

Next cones will be found geometrically. Once they are found, we assume they are correct and we search (again) until there are no more cones to look for or the loop is closed.

See Alg. 1 to better understand how next cones are found in every iteration.

Algorithm 1 Urimits' iterative approach

```
s ← (x_0, y_0, θ_0)
t ← partial TLs' trace
cones ← possibleNextCones(s)
while t_0 ≠ t_{n-1} and ¬cones.empty() do
    t.append(best(cones))
    s ← newState(t)
    cones ← possibleNextCones(s)
end while
```

Note that this process is repeated for both traces (left and right).

5.2.3 Heuristic: angle and distance

When looking for the best next cone for a **particular trace** from position p_0 and having a previous position p_{-1}, which is the cone (with position p_c) that is most likely to go next? Urimits solves this problem by finding the cone with lowest heuristic h_{uri}. Assuming that the optimal track limits minimizes a functional that is a linear combination between angle and distance. This functional (heuristic) combines two sub-heuristics, distance h_{dist} and angle h_β, through the use of a weighting factor, this can be seen in Eq. 5.2.

$$\begin{aligned} h_{dist} &= \text{dist}(p_0,\, p_c) \\ h_\beta &= -\log\left(\max\left(0,\, \frac{\beta(p_{-1},\, p_0,\, p_c)}{\pi} - \mu\right)\right) \end{aligned} \tag{5.1}$$

$$h_{uri} = h_{dist} \cdot \gamma_{uri} + h_\beta(1 - \gamma_{uri}) \tag{5.2}$$

where:

$\boldsymbol{\mu}$ is a corrector to the logarithm, makes the logarithm shift to the left, not to penalize too much small angles, usually 0.2.

$\boldsymbol{\beta}$ is a function that calculates the angle between 3 points in $\mathbb{R}^2$, it always gives the smallest angle, being the range $[0, \pi)$.

$\boldsymbol{\gamma_{uri}}$ is the weighting factor. It allows to give more importance to the angle or to the distance.

What we obtain with a logarithm is no-linearity, we want the probability for a cone to be the next one to decrease logarithmically.

5.2.4 Angle-based correction

Once the first trace is computed using techniques described in 5.2.2 and 5.2.3, it is very likely that the calculated trace is miscalculated, as shown in Fig. 5.2, the inner trace could have taken cones belonging to the outer trace.

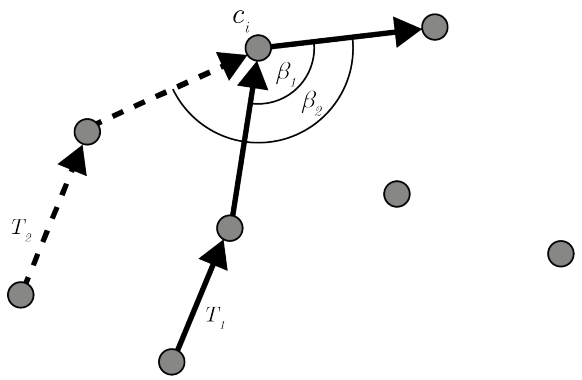

Figure 5.2: Traces collision.

When computing the outer trace, a conflict is reached given that both traces want to take cone c_i. To solve this problem, Urimits saves for each cone with a successor the minimum angle it forms. In this case, the angles created by both traces at this

point, β_1 and β_2, are compared. The trace with the greater angle will keep the cone, in this case, T_2. The incorrect trace T_1, will need to be recalculated since c_i belongs to T_2.

Each trace will have an exclusion set E, i.e. no cone in this set can be part of the trace. c_i is added to the exclusion set of T_1.

When performing a new search for T_1, the best cone not in E_1 will be appended to the trace.

On the other hand, c_i will be appended to T_2 and the iterative search process will restart for T_1. The effects can be seen in Fig. 5.3.

With this mechanism, Urimits makes sure that no trace takes other trace's cones.

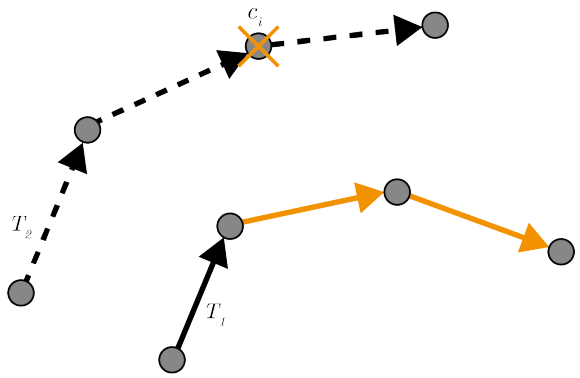

Figure 5.3: Traces state after correction.

5.2.5 Abort on intersection

When a valid-closed track limits is obtained, see 2.1.4, it might still be wrong, even though every single cone is not repeated in both traces, it might happen that the traces intersect with each other or with themselves, as shown in Fig. 5.4.

If that is the case, we assume that the track is not fully seen yet. The execution is aborted and no result is given; we wait until new data arrives.

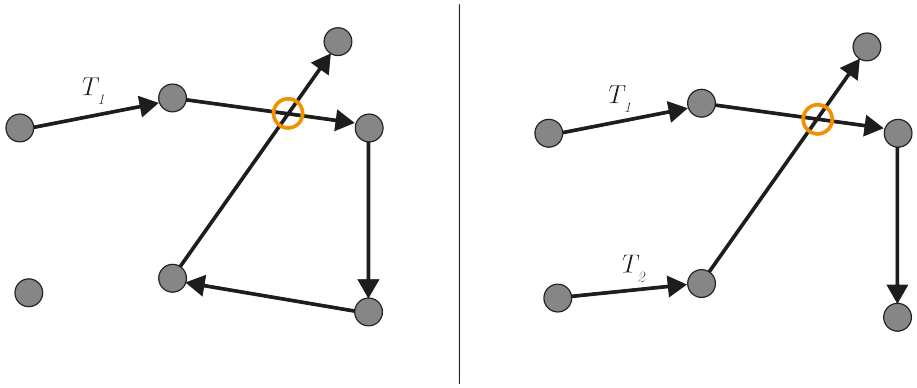

Figure 5.4: Traces intersection with themselves and with each other.

5.3 Pipeline

The pipeline is formed by multiple tasks executed sequentially, as shown in Fig. 5.5.

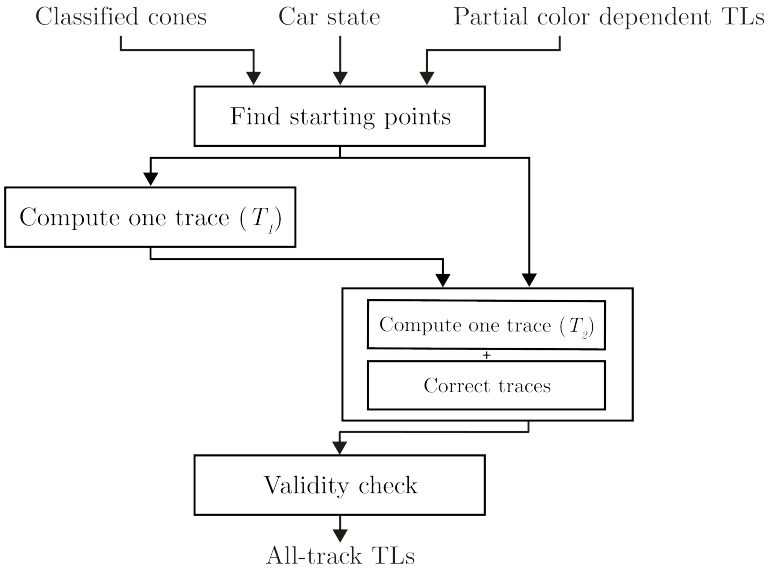

Figure 5.5: Urimits' pipeline.

5.3.1 Find starting points

In order to start this algorithm, the two first cones must be found. Two situations are distinguished here.

- **Partial color dependent TLs are empty:** In this situation, to find the starting points, we assume that the car is in-track and looking forward, i.e. car's heading is parallel to the traces.

  The space is divided using car's heading. Cones behind the car are discarded and the closest cone is chosen on each side. In Fig. 5.6, c_i will be chosen as left's first and c_j as right's first.

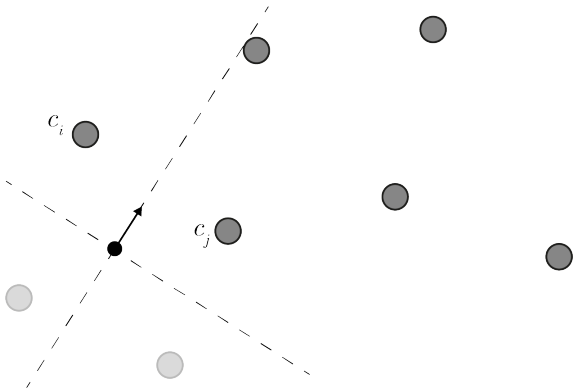

Figure 5.6: First points election.

- **Partial color dependent TLs has some cones:** In this case, correctness of traces is assumed. The starting points will be the last two cones of the traces.

5.3.2 Compute one trace

This task is responsible for computing as well as possible a single trace of track limits, it uses the methods explained in 5.2.2 and 5.2.3. When computing T_i, this process can be divided into:

1. Find the best cone c_{next} to append to T_i.
   - Find all cones in a radius d_{max}.
   - Remove all cones in T_i's exclusion set E_i.
   - Remove all cones whose angle with last cone of T_i is smaller than β_{min}.
   - Choose best cone according to h_{uri}.
2. Append c_{next} to T_i.
3. Repeat 1. and 2. until no more cones are found or the loop is closed.

5.3.3 Correct traces

This task is carried out in conjunction with Compute one trace for T_2. This task will only function when a conflict is encountered (see 5.2.4), i.e. c_{next} for T_2 is already taken by T_1.

Every time a conflict is found, it calls Compute one trace so it recalculates the wrong trace without the possibility of taking the conflicted cone.

5.3.4 Validity check

It checks whether or not the calculated track limits are valid. It does so by performing a series of checks. If one of the checks fail, the limits are considered invalids.

- Each trace **size** (cone number) must be greater or equal to 3.
- The traces must **not intersect** with themselves or with each other.
- A **minimum ratio** of cones must be taken by both traces in respect to the total number of cones.
- The **distance** between trace **centroids** can not exceed a threshold.

Chapter 6

Results

The two proposed systems have been integrated into the autonomous system pipeline of the car. These modules complete the perception system, which has the responsibility of computing and outputting a valid track limits. Fig. 6.1 provides a general view of the perception pipeline that will have the CAT14x during this season's competitions.

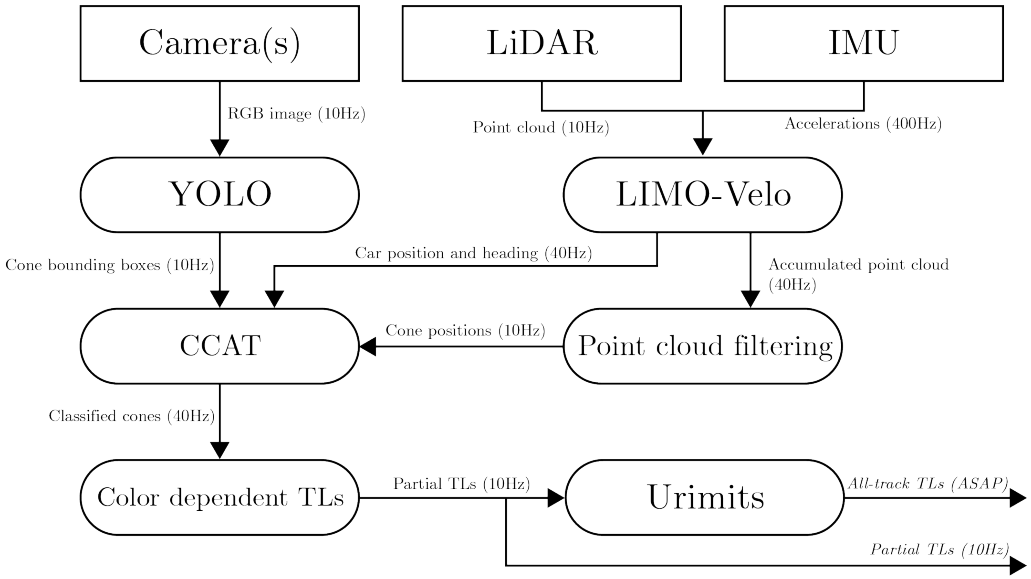

Figure 6.1: CAT14x's perception pipeline.

Due to unexpected mechanical problems, at the day of writing this report, the CAT14x is not ready for testing yet. Consequently, both solutions will be tested on rosbags, i.e. a playback of recorded data during a testing session with our single-seater. These recordings contain raw data (exactly the same that would be received in a real test).

The only problem with this testing mechanism is that car's feedback is not seen, in other words, we can not be 100% sure that these solutions will work when the control of the car depends on these. Nevertheless, a certain degree of confidence can be achieved.

Special emphasis will be placed in any "weird" or unexpected behaviour of the

proposed systems.

To ensure credibility, all experiments will be carried out on 3 different rosbags. As seen in Fig. 6.2, rosbag 1 consists of an easy (circle-like) track with almost no straights, rosbag 2 has longer straights, hairpin turns and includes orange cones, 2 at the beginning and some at at the left. Rosbag 3 has the particularity that all cone positions are given at the beginning, specifically at second 15 (assuming a perfect cone detection from the global map).

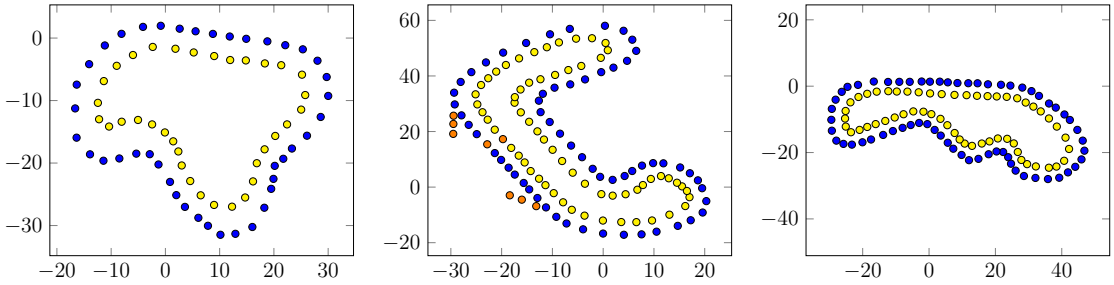

Figure 6.2: Track layouts 1, 2 and 3 (rosbag).

All experiments have been carried out in my personal computer using car's data, its specifications are:

- Intel i7 6700U 2-core (4 threads) 64-bit CPU
- 8 GB DDR3L SDRAM

This machine has a much worse performance compared to the car's processing unit, see 3.5.2.

6.1 CCAT

Here, all experiments regarding the proposed cone classification and tracker system will be effectuated.

6.1.1 Correct cone classification

Correct cone classification is key to this system.

Classification delay

For each cone, the time until a correct classification of each cone will be measured for each track layout.

As seen in Fig. 6.3, the time between the detection of a cone from the global map until its correct classification goes up t 28 seconds in some rare cases.

In rosbag 1, the distribution is more uniform whereas in rosbag 2, an accumulation of points can be seen in lower range. I assume that is due to the track's shape, it is easier to classify cones in a straight than in a curve (cones are seen for a longer period of time), and rosbag 2 is composed of long straights.

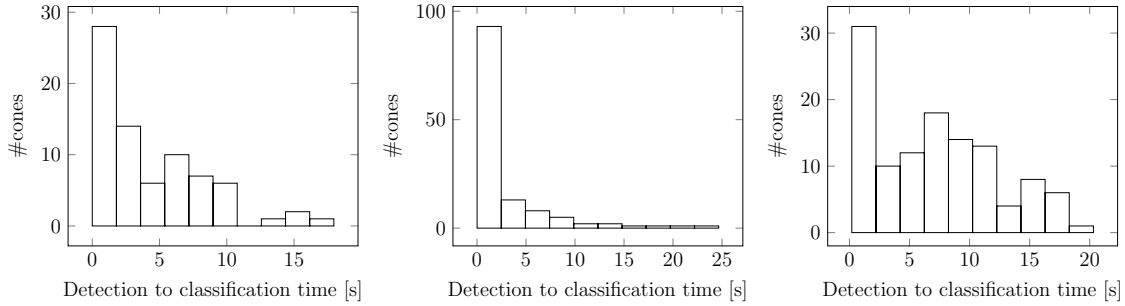

Figure 6.3: Time from cone detection to correct classification.

In rosbag 3 we can see that the times are very similar to rosbag 1. And, since the track's shape are very alike, we can conclude that the delay to observe a cone in the global 3D map does not cause a high impact in cone classification time.

Number of type changes in time

In order to quantify the stability of the classification system, we will compare the number of times the type of each cone changes during a hole lap for the three track layouts. See Fig. 6.4.

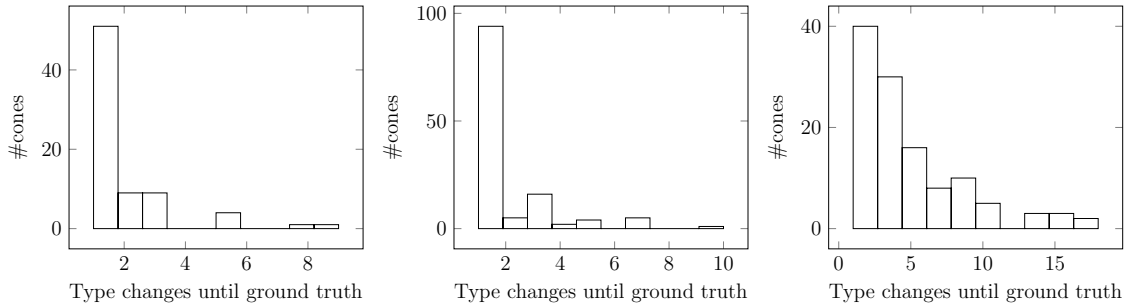

Figure 6.4: Number of cone type changes until the correct type is achieved.

We can see that most of the points are in the $[1, 3]$ range, which in conjunction with the classification time tells us that CCAT only changes the type of a cone when is sure.

Car distance

It is really necessary to have a large perception range in order to anticipate all possible events and to improve possibilities of seeing the whole track earlier, i.e. we can optimize trajectory and velocities sooner.

As shown in Fig. 6.5, the distance of correct classification of cones follows a uniform distribution.

From this figure, we can assure that the classification range is up to $\approx$40 m.

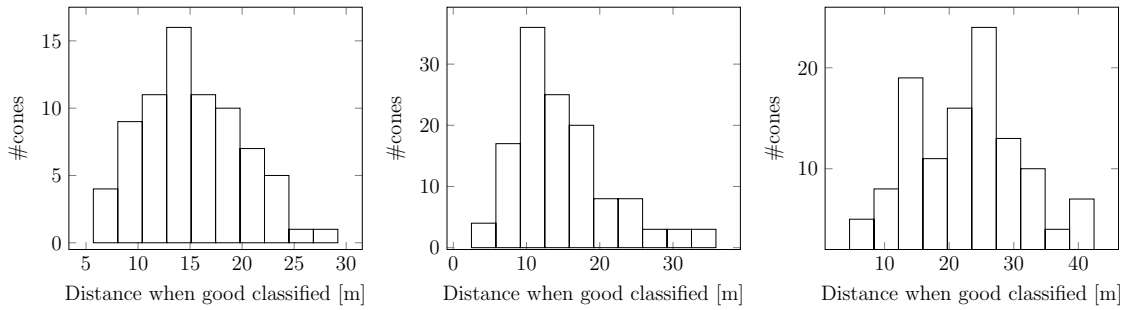

Figure 6.5: Cone distance with car when correct type is achieved (and not changed).

6.1.2 False positives removal

Since cone positions are detected from the global 3D map that LIO provides us, it is not rare to detect a cone where it is none (false positive). In our application, robustness is crucial, and as defined in Objective A-3, CCAT must be able to remove false positives from the track.

The only place where a false detection from the global 3D map could be an issue is near the track's cones. To see if CCAT can deal with it, we have introduced three false positives in rosbag 3, as shown in Fig. 6.6.

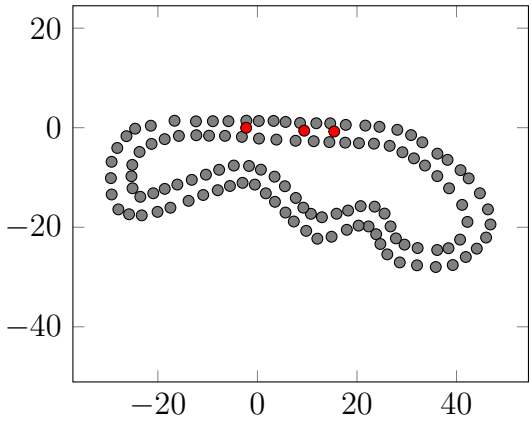

Figure 6.6: Layout 3 with false detections.

The result is quite amazing, CCAT uses a statistical model to invalidate cones that are near and do not get detected by the YOLO neural network.

The cones get invalidated when they are at ≈10 m in front of the car.

6.1.3 Delay

As stated in Objective D - Real time, real time must be ensured. The objective of this test will be to quantify the execution time of the algorithm over time.

When running the rosbags in my computer, the times stated in Fig. 6.7 were obtained.

We can see that the mean execution time is ≈35 ms, i.e. $\frac{1}{35\text{ms}} = 28.6\text{Hz}$. It is below the 40 Hz that data comes in at, but it is still acceptable for this application.

Note that in the car's processing unit, this system will probably run much faster (presumably at the desired 40 Hz).

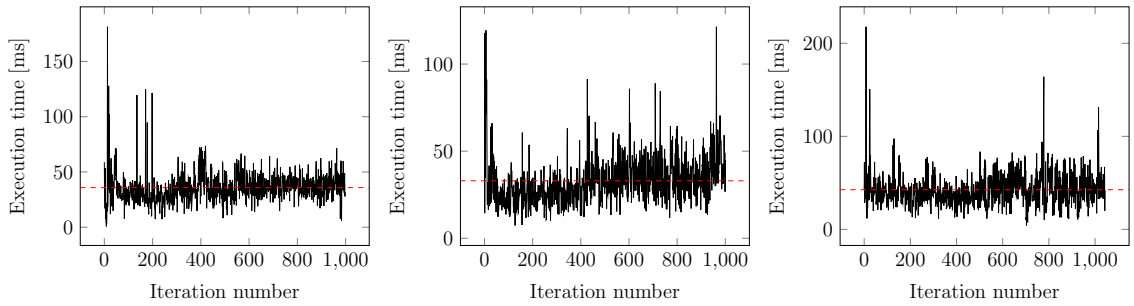

Figure 6.7: CCAT's pipeline execution time.

6.1.4 Comparison with BB2L+FastSLAM

Last season's system consisted of a cone detection phase on a single LiDAR frame and a second phase of keeping track of the cone color using a simple classification system.

FastSLAM uses EKFs in order to estimate cone positions, as a result, all the cones experience a large position change from first detection to the ground truth position over time. This change is in average ≈0.8 m, whereas with the global 3D map cone detections + CCAT, this difference goes down to only ≈0.06 m.

This offset can cause a lot of problems later on in the pipeline, to compute the track limits, an accurate position is needed, otherwise we encounter cones with a lot of noise in position and the track's shape is not well defined.

Taking this into account we can affirm that the proposed model is better.

6.2 Urimits

In this section the proposed track limits color blind and all-track will be validated.

Moreover, Urimits is the first algorithm of its type in the team. Taking this into account, no comparison to other algorithms is possible.

6.2.1 Correctness and distance

Being all-track implies that it should output nothing while the whole track is not detected.

The algorithm will be run on the 3 track layout and since the algorithm should compute the full TLs before the car completes the lap, we will measure car's distance remaining to the beginning of the lap.

Track layout 1

In the case of rosbag 1, as seen in Fig. 6.8, the track limits closes the lap as soon as all the cones are detected in the global 3D map. That happens 18 metres before

the car crosses the finish line. In a small track like this, this supposes closing the lap after completing only 77%.

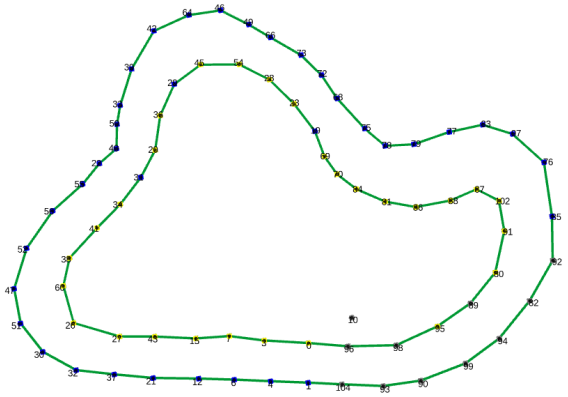

Figure 6.8: CCAT+Urimits execution on rosbag 1.

Track layout 2

In layout 2, Urimits is not capable of closing the loop as expected. This is Urimits' biggest problem, the whole model is based on a greedy approach. If we have a misdetection on a certain place, the entire algorithm fails. Here, the algorithm finds a cone that does not belong to the track itself and lenghtens the trace from there.

This rosbag's recording day, some cones were placed (orange) to separate people from the track, this cannot happen in a official competition but is still a big issue.

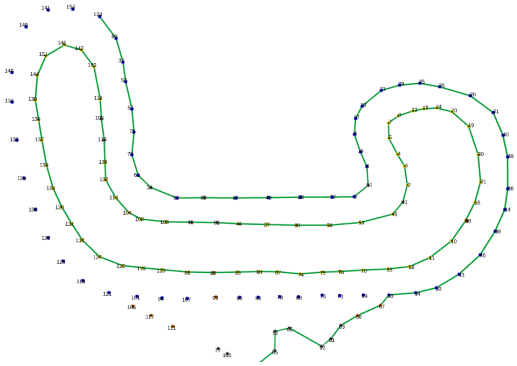

Figure 6.9: CCAT+Urimits execution on rosbag 2.

Track layout 3

In track 3, the loop is closed as soon as all cones are detected at second 15. Note the color of the cone marker in Fig. 6.10, most of them are gray, which means that they have not been classified yet.

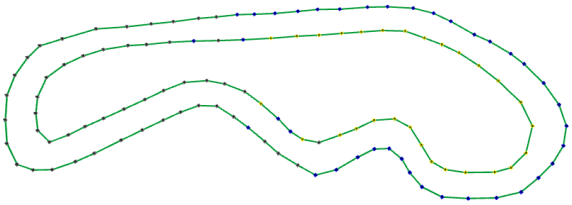

Figure 6.10: CCAT+Urimits execution on rosbag 3.

6.2.2 Delay

It is true that this algorithm's output does not have a defined rate (it should output as soon as the whole track is seen.

However, the computation time is still adding a delay to the pipeline, in this section, the delay will be quantified throughout the lap.

The execution time is expected to rise quadratically with the number of cones the car sees. This means that the closer the car is to the end of the track (more cones are mapped), the longer it should take to compute the limits of the track.

As seen in Fig. 6.11, the delay clearly increases over time. However, this is not a big deal since this amount of time is assumable.

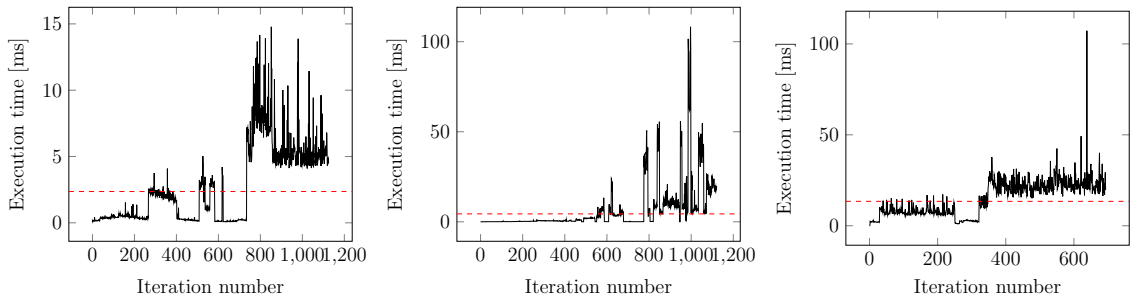

Figure 6.11: Urimits's pipeline execution time.

It is really interesting to see that in the third rosbag (the one with perfect cone detections), the average execution time rises up to 13.4 ms, whereas the other's average is below 5 ms. This really makes evident that when the track is long and with a plentiful of cones, the cost will be high. In addition, this also proves that Objective D - Real time is accomplished, since this average execution time would have a frequency of $\frac{1}{13.4\text{ms}} = 78\text{Hz}$.

In addition, we see that the algorithm makes some spikes, i.e. outliers which have a huge execution time, this is due to the recalculation of traces (see 5.2.4) which if the starting point (car position) coincides with being in a curve, the amount of recalculations is much higher.

Chapter 7

Conclusions

7.1 Key ideas

- **Statistical models** work best in autonomous driving given that operation conditions are not constrained enough.
- When working on a real-time application, finding an optimal solution to the problem in a constrained time is impossible. Shortcuts must be taken to find a **sub-optimal solution**.

7.2 Achieved objectives

The two proposed systems have accomplished all the objectives defined in section 1.4.

A - Classify each cone detected on the 3D map

CCAT has successfully achieved this objective, including all sub-objectives.

A-1 It successfully synchronizes data and even provides support if data is missing in some occasion.

A-2 All bounding boxes are registered to the closest cone (3D matchings).

A-3 Statistically, false positives in LiDAR observations are removed as soon as possible.

A-4 All detected cones are uniquely identified, and cone clustering helps improve overall system stability.

B - All-track color blind track limits

Urimits meets successfully this objective, alongside both sub-objectives.

B-1 When the partial color dependent track limits is not empty, the proposed system, takes last cones as the starting point, hence the track limits are extended.

B-2 This system iteratively adds cones until it detects that the loop is closed (plus it performs a validity check).

C - Improvement

As seen in the results chapter, these systems have **enhanced** significantly CAT14x's perception system over last season's car, Xaloc. Cones are detected earlier and more accurately. In addition, now there is the possibility of detecting the whole track limits before the car physically crosses the finish line.

D - Real time

The two proposed systems run on-car and in real time. In fact, they will be running during the competitions this summer.

7.3 To summarize

In order to solve last season's problems regarding the perception of our Formula Student driverless car (Xaloc), we have developed two new systems that will run in this season's car (CAT14x) during competitions in summer.

CCAT solves the sensor fusion problem using a statistical model that classifies cones. Since this classification in statistically we can not only distinguish between types but also get the confidence of each cone. The model does also remove global map misdetections in order to maintain robustness.

Urimits uses CCAT's data and takes advantage of classified cones marked as unknown. These are the cones that get detected in the global map but not by the cameras, i.e. color cannot be obtained. The model uses only the geometry of the track in order to compute the all-track track limits.

Chapter 8

Future work

8.1 CCAT's synchronization issue

When registering the camera detected bounding boxes into the cone centroids, as stated in 4.2.3, due to a synchronization issue between cameras and car's state (position and heading), bounding boxes do not fit with projected centroid in all situations as expected. Two possible solutions are stated.

8.1.1 Car position and heading interpolation

We get a new car state (position and heading) every $\frac{1}{40\text{Hz}} = 0.025\text{s}$, pictures are taken every $\frac{1}{30\text{Hz}} = 0.0\overline{3}\text{s}$.

If we have car's state at time intervals t_0 and t_1 and a new set of bounding box from time t_k is received ($0<k<1$), a possible solution is to linearly interpolate the car states in order to obtain a car state at time t_k. See Fig. 8.1 for better understanding.

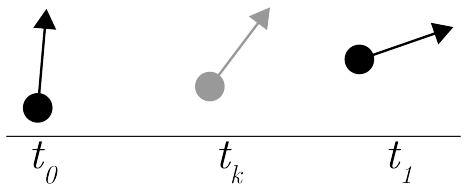

Figure 8.1: Car state interpolation.

8.1.2 Hardware synchronization

The ideal solution to the synchronization problem is to temporal synchronize electronically the localization algorithm (LIO) with the cameras.

This way, the cameras will take pictures at the same time that a new car state (position and heading) is computed. Later on in CCAT, the synchronization between the data would be perfect and no offset would be found.

8.2 Larger statistical classification model

Currently, as specified in 4.2.3, only the cone distance to the camera plane (d_{cp}) and its matching distance (d_m) is utilized in order to qualify a matching of a particular cone. Additional variables such as the 3D Euclidean distance between new cone centroid and old centroid (when the cone moves due to a better detection), the global map cone observation confidence or the cone class bounding box confidence.

These are variables that the system already has but does not use.

8.3 Improved camera system

Our setup has two RGB cameras with a wide angular lens. CCAT's architecture has been thought out to make it easier to change camera's setup. The number of Matcher modules can be incremented indefinitely, each one will have its extrinsic and intrinsic parameters.

The first thing i think will need to change is the focal length of the cameras. To better see farther, zoom cameras will suppose a huge boost to the classification range.

From my point of view, putting zoom cameras tilted to left and right does not make any sense; in a curve, improving the range does not give you the possibility to run faster. In a straight though, being able to classify cones farther can extend the prediction horizon and thus boost the velocity.

8.4 Track limits tree search

One of the down sides of Urimits is that it can fail if misdetections are the best choice at a particular moment. As shown in Fig. 8.2, T_1 and T_2 are the correct traces, but Urimits mistakes T_2' for T_2.

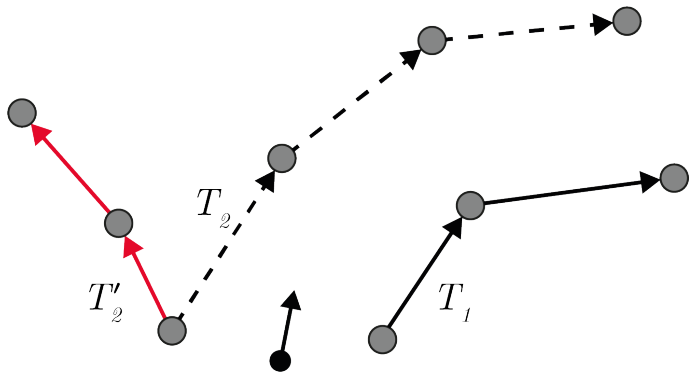

Figure 8.2: Urimits' best choice failure.

To solve this issue, multiple options should be considered and not only the best cone according to the angle and distance. A tree search would be suitable in this case. Saving all the the possible traces would result in an exponential growing problem. Pruning mechanisms should be used in order to reduce the complexity.

Bibliography

[1] Formula Student Germany. Rules 2022 v1.0, 2022. https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FS-Rules_2022_v1.0.pdf.

[2] Formula Student Germany. Competition handbook v1.1, 2022. https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FSG22_Competition_Handbook_v1.1.pdf.

[3] Andreu Huguet Segarra. LIMO-Velo: A real-time, robust, centimeter-accurate estimator for vehicle localization and mapping under racing velocities. Degree report, UPC Barcelona, February 2022. URL http://hdl.handle.net/2117/365241. Accepted: 2022-04-04T11:59:49Z Publisher: Universitat Politècnica de Catalunya.

[4] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, pages 1–21, 2022. doi: 10.1109/TRO.2022.3141876.

[5] Wikipedia contributors. Simultaneous localization and mapping — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Simultaneous_localization_and_mapping&oldid=1092202177, 2022.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[7] Niclas Vödisch, David Dodel, and Michael Schötz. Fsoco: The formula student objects in context dataset. *SAE International Journal of Connected and Automated Vehicles*, 5(12-05-01-0003), 2022.

[8] Arnau Roche López. LiDAR cone detection as part of a perception system in a Formula student car. Degree Thesis, UPC, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament de Teoria del Senyal i Comunicacions, July 2019. URL http://hdl.handle.net/2117/168711. Accepted: 2019-09-26T07:29:05Z Publisher: Universitat Politècnica de Catalunya.

[9] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference*

on Robotics and Automation (Cat. No.03CH37422), volume 2, pages 1985–1991 vol.2, 2003. doi: 10.1109/ROBOT.2003.1241885.

[10] Hbpublications. The 5 step budgetary control process, Oct 2021. URL `https://hbpublications.com/2020/06/17/the-5-step-budgetary-control-process/`.

[11] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975. ISSN 0001-0782. doi: 10.1145/361002.361007.

[12] Eloi Bové Canals. Position estimation using a stereo camera as part of the perception system in a formula student car. Degree Thesis, UPC, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament de Teoria del Senyal i Comunicacions, Jun 2019. URL `http://hdl.handle.net/2117/168738`.

[13] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1): 46–55, 1998. doi: 10.1109/99.660313.

[14] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: A toolkit for real domain data visualization. *Telecommun. Syst.*, 60(2):337–345, oct 2015. ISSN 1018-4864. doi: 10.1007/s11235-015-0034-5.