Development of the Model Predictive Controller and Simultaneous-Localization-And-Mapping Algorithm of the Control System

Degree Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Albert Gassol Puigjaner

In partial fulfillment
of the requirements for the degree in
Telecommunications Technologies and Services Engineering

Advisors: Albert Aguasca Solé, Vicenç Puig Cayuela
Barcelona, June 2021

Contents

List of Figures 5

List of Tables 6

1 Introduction 14
- 1.1 Formula Student . . . 14
- 1.2 BCN eMotorsport . . . 15
- 1.3 Objectives . . . 16
- 1.4 Requirements . . . 16
  - 1.4.1 Localization and Mapping Requirements . . . 16
  - 1.4.2 Controller Requirements . . . 17
- 1.5 Specifications . . . 17
  - 1.5.1 Localization and Mapping Specifications . . . 17
  - 1.5.2 Controller Specifications . . . 17
- 1.6 Methods and procedures . . . 18
- 1.7 Work plan . . . 18
  - 1.7.1 Gantt Diagram . . . 19
  - 1.7.2 Milestones and deliverables . . . 19
  - 1.7.3 Deviations and incidences . . . 20

2 State of the art in Formula Student Estimation and Control 21

3 Model Predictive Control 22
- 3.1 Principles . . . 22
  - 3.1.1 System dynamics . . . 22
  - 3.1.2 Objective function . . . 23
  - 3.1.3 Constrains . . . 23
  - 3.1.4 General nonlinear MPC . . . 23
- 3.2 Dynamic Model . . . 24
  - 3.2.1 Dynamic Bicycle Model . . . 24
  - 3.2.2 Longitudinal Force . . . 25
  - 3.2.3 Lateral Forces . . . 25
- 3.3 Formulating the Nonlinear MPC . . . 26
  - 3.3.1 System dynamics . . . 26
  - 3.3.2 Objective Function . . . 27
  - 3.3.3 Constraints . . . 29
  - 3.3.4 Formulation . . . 30
- 3.4 NMPC Implementation . . . 31
  - 3.4.1 Algorithm . . . 32
- 3.5 LPV–MPC Formulation . . . 34
  - 3.5.1 LPV Dynamic Model . . . 34
  - 3.5.2 Objective function . . . 35
  - 3.5.3 LPV_MPC minimization problem . . . 36
- 3.6 LPV-MPC Implementation . . . 36

3.6.1 Algorithm 37

4 Graph-Based Simultaneous Localization and Mapping 40
4.1 Problem Statement 40
4.1.1 Odometry Measurements 41
4.1.2 Observations and Data Association 41
4.2 Graph-Based SLAM formulation 42
4.2.1 Dynamic Bayesian Network 43
4.2.2 Cost Function 44
4.3 Implementation 45
4.3.1 Front-End 45
4.3.2 Back-End 47
4.3.3 ROS Implementation 49

5 Testing and Results 50
5.1 Nonlinear Model Predictive Control 50
5.1.1 Simulation 50
5.1.2 Testing 52
5.2 LPV-MPC 54
5.3 MPC computational time 55
5.4 Graph-based SLAM 56
5.4.1 Testing 56
5.4.2 Computational time 57

6 Budget 58

7 Conclusions and Future Work 59
7.1 Conclusions 59
7.2 Future Work 60

A Appendix 62
A.1 Estimation and Control Appendix 62
A.1.1 Estimation 62
A.1.2 Control 64
A.1.3 Robot Operating System 65
A.2 Nonlinear MPC Appendix 67
A.2.1 Solving with FORCESPRO 67
A.2.2 Actuators Outputs 69
A.2.3 Parameters Tuning 69
A.3 Linear Parameter Varying MPC Appendix 71
A.3.1 Linearization of the Dynamic Bycicle Model 71
A.3.2 Solving with FORCESPRO 73
A.4 Graph-Based SLAM Appendix 74
A.4.1 Odometry measurement model and error 74
A.4.2 Landmark observation model and error 75
A.4.3 Gauss-Newton algorithm 75

References **79**

List of Figures

1 Formula Student competitions. 14
2 Trackdrive cones. 15
3 XALOC 2.0 16
4 SLAM Gantt diagram 19
5 MPC Gantt diagram 19
6 Representation of reference, current and predicted states and predicted controls. 22
7 Dynamic Bicycle Model. 25
8 Longitudinal and lateral errors. 28
9 MPC block diagram. 31
10 MPC ROS graph. 32
11 Data Association 42
12 Dynamic Bayesian Network 43
13 Dynamic Bayesian Network with constraints 44
14 Dynamic Bayesian Network with probabilities 44
15 Graph-Based SLAM Front-End 46
16 Graph-Based SLAM Back-End 48
17 Graph-Based ROS Graph 49
18 Longitudinal velocity: current (pink and red), solution (cyan) and target (blue). 51
19 Lateral velocity: current (red and cyan) and solution (blue). 51
20 Longitudinal velocity (FSG 2019 track). 51
21 Lateral velocity (FSG 2019 track). 51
22 Longitudinal velocity (FSI 2019 track). 51
23 Lateral velocity (FSI 2019 track). 51
24 Longitudinal velocity in real testing. 53
25 Lateral velocity in real testing. 53
26 Longitudinal velocity in real testing: current (red), solution (cyan) and target (blue). . . . 53
27 Lateral velocity in real testing: current (red and cyan) and solution (blue). 53
28 Yaw rate in real testing: current (pink), solution (blue). 53
29 Desired (red) and current (blue) steering actuator position in testing. 53
30 Longitudinal velocity in simulation: target (blue), LPV-MPC solution (cyan) and current (red). 55
31 Lateral velocity in simulation: LPV-MPC solution (blue) and current (cyan and red). . 55
32 Timing results of LPV-MPC and NMPC. 56
33 Graph-Based SLAM results 57
34 Global frame of coordinates. 63
35 SLAM problem. 63
36 TrackLimits algorithm. 64
37 Nodes, topics, publishers and subscribers. 66
38 Estimation and Control ROS architecture. 67
39 MPC cost function terms 70
40 Simplified Magic Formula Model. Fy vs. slip angle. 71

List of Tables

1 SLAM Milestones and deliverables. 20
2 MPC Milestones and deliverables. 20
3 NMPC parameters used in the simulations. 52
4 NMPC parameters for XALOC: first lap 54
5 NMPC parameters for XALOC: GRO laps 54
6 LPV-MPC parameters used in simulation. 55
7 Budget. 58

Abbreviations

AMZ Academic Motorsports Club Zurich

BCN Barcelona

CAN Controller Area Network

CoG Center of Gravity

DBN Dynamic Bayesian Network

ECU Electronic Control Unit

EKF Extended Kalman Filter

FSD Formula Student Driverless

FSSIM Formula Student Simulator

GNSS Global Navigation Satellite System

GPS Global Positioning System

GPU Graphic Processing Unit

GRO Global Race Optimizer

INS Inertial Navigation System

IP Internet Protocol

KA-RaceIng Karlsruher Institut für Technologie team

LiDAR Laser Imaging Detection and Ranging

LPV Linear Parameter Varying

LPV-MPC Linear Parameter Varying Model Predictive Control

MPC Model Predictive Control

NLP NonLinear Programming

NMPC Nonlinear Model Predictive Control

PID Proportional Integral Derivative

QCQP Convex multistage Quadratically Constrained Program

ROS Robot Operating System

SE(2) Two dimensional special Euclidean group

SLAM Simultaneous Localization and Mapping

TCP Transmission Control Protocol

UPC Universitat Politècnica de Catalunya

Abstract

This thesis aims to develop and implement two algorithms to control the first and fastest Spanish Formula Student Driverless (FSD) car and an algorithm to solve its localization and mapping problem. This project has been elaborated within the framework of the BCN eMotorsport team and seeks to improve the competitiveness of the team to compete against the best European teams.

An introduction to the necessary algorithms of the Estimation and Control architectures of a FSD vehicle is given at the beginning. Then, a detailed explanation of the control algorithms, altogether with their implementation and integration to Robot Operative System (ROS), is described. This is followed by the development and implementation of the localization and mapping algorithm. Finally, these algorithms are tested in simulation and with the BCN eMotorsport driverless car (XALOC).

Resum

Aquesta tesi de final de grau té com a principals objectius el desenvolupament i implementació de dos algorismes per controlar el primer, i més ràpid, monoplaça Formula Student autònom de l'estat, així com el disseny i implementació d'un algorisme que sigui capaç de resoldre el problema de la localització i mapejat d'aquest monoplaça. Aquest projecte es troba dins el marc de treball de l'equip BCN eMotorsport i cerca augmentar la competitivitat de l'equip per ser capaços de competir contra els millors equips a nivell europeu.

Per començar, es presenta una breu introducció als algorismes necessaris per desenvolupar les arquitectures dels sistemes d'Estimació i Control d'un monoplaça Formula Student Driverless (FSD). A continuació, s'exposa una explicació detallada sobre els algorismes de control d'aquesta tesis, juntament amb la seva implementació i integració amb Robot Operating System (ROS). Seguidament, es proposa l'algorisme de localització i mapejat i la seva implementació. Finalment, aquests algorismes són posats a prova mitjançant simulacions i també s'incorporen al vehicle autònom de l'equip BCN eMotorsport (XALOC).

Resumen

Esta tesis de final de grado tiene como objetivo principal el desarrollo e implementación de dos algoritmos para controlar el primero, y más rápido, monoplaza Formula Student autónomo del estado, así como el diseño e implementación de un algoritmo que sea capaz de resolver el problema de localización y mapeado de este monoplaza. Este proyecto se enmarca dentro del trabajo del equipo BCN eMotorsport y pretende aumentar la competitividad del equipo para ser capaces de competir contra los mejores equipos a nivel europeo.

Para empezar, se presenta una breve introducción a los algoritmos necesarios para desarrollar las arquitecturas de los sistemas de Estimación y Control de un monoplaza Formula Student Driverless (FSD). A continuación, se expone una explicación detallada sobre los algoritmos de control de esta tesis, juntamente con su implementación e integración con Robot Operating System (ROS). Seguidamente, se propone el algoritmo de localización y mapeado y su implementación. Finalmente, estos algoritmos se prueban mediante simulaciones y también se incorporan al vehículo autónomo del equipo BCN eMotorsport (XALOC).

Dedication

This thesis is dedicated to my family for their incredible support during the last two years as member of BCN eMotorsport.

This thesis is dedicated to all the members of BCN eMotorsport, with an special mention to the Autonomous Systems department members (Raúl González, Andreu Huguet, Ignasi Mallén, Toni Salom, Marta Miranda, Laura Pérez and Oriol Gorriz) for their support and sensational atmosphere created within the department.

Finally, this thesis is dedicated to the Driverless UPC Perception and Control members Arnau Roche, Adrià López and Fèlix Martí for their wise advises and support in times of stress and nervousness.

Acknowledgement

I would like to thank my advisor Vicenç Puig for his constant wise advises and recommendations throughout the development of this thesis, as well as for his willing to collaborate and constant attention.

I would like to thank my advisor Albert Aguasca for his attention and willing to collaborate with the project with all his available resources. Thank you Albert, the BCN eMotorsport project would not have existed without your incredible help and advise.

Finally, I would like to thank all the professors and researchers that help the team with their recommendations and resources.

Revision history and approval record

Revision	Date	Purpose
0	15/03/2021	Document creation
1	15/04/2021	First part revision
2	17/05/2021	Second part revision
3	30/05/2021	Structural revision
4	15/06/2021	Final revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Albert Gassol Puigjaner	
Vicenç Puig Cayuela	
Albert Aguasca Solé	

Written by:		Reviewed and approved by:	
Date	15/06/2021	Date	15/06/2021
Name	Albert Gassol Puigjaner	Name	Vicenç Puig Cayuela
Position	Project Author	Position	Project Supervisor

1 Introduction

The goal of this project is to design and implement localization, mapping, and controller algorithms as part of the estimation and control system of a Formula Student Driverless car. These algorithms aim to localize the car inside an unknown environment whilst recognizing its landmarks and apply the optimal controls to follow a predetermined trajectory. This project starts from scratch and is part of the BCN eMotorsport Formula Student Driverless project. An introduction to the Formula Student and the BCN eMotorsport team is presented below.

1.1 Formula Student

Formula Student is an international competition in which university teams around the world compete against each other to design and build the best race car. Teams represent universities and are exclusively composed of students. These competitions are held during the summer in several places worldwide such as Germany, Spain, Austria, United States of America and Czech Republic, among others. A map containing all the competitions around the world is shown in Figure 1. Three different classes of race cars compete in Formula Student competitions: Combustion Vehicles, Electric Vehicles and Driverless Vehicles. Each class has different technical inspections, challenges and scoring in the competitions.

Figure 1: Formula student competitions around the world.

Regarding Driverless Vehicles, these must pass a mechanical inspection to check the mechanical parts of the vehicle (such as the chassis and the vehicle dynamics), an electrical inspection to check all the electronics, wiring and Electronic Control Units (ECU), an accumulator inspection to check the status of the high voltage battery, an autonomous inspection to check the correct functioning of the autonomous actuators and a set of different technical inspections such as a tilt test, a brake test and a rain test. Once a team has passed the inspections, the team can compete in dynamic events.

The dynamic events evaluate the performance of race cars. For Driverless Vehicles, four different events are held: Autocross, Trackdrive, Acceleration and Skidpad. The Autocross event consists in completing one lap in an unknown circuit. The Trackdrive event consists in an unknown circuit, which usually coincides with the Autocross circuit, in which the car must complete 10 laps before stopping at the end-line and finishing the mission. The Acceleration event consists in a 75m straight lane that the car must complete as fast as possible. The Skidpad event consists in a circuit composed of two circles in which the car must complete two laps in each circle. All these dynamic events are timed and the team that achieves a shortest time wins the event.

Furthermore, static events are also held in Formula Student competitions. The static events aim to evaluate the engineering, communication, economical and management skills of the team members. In order to do so, three different events take place in the competitions: Business Plan, Cost Report and Engineering Design.

To ease the task of developing the autonomous architecture for a Formula Student Driverless car, the rules of the competition establish a predetermined set up of the circuits. Each circuit is composed of blue, yellow, small orange and big orange cones, as shown in Figure 2.

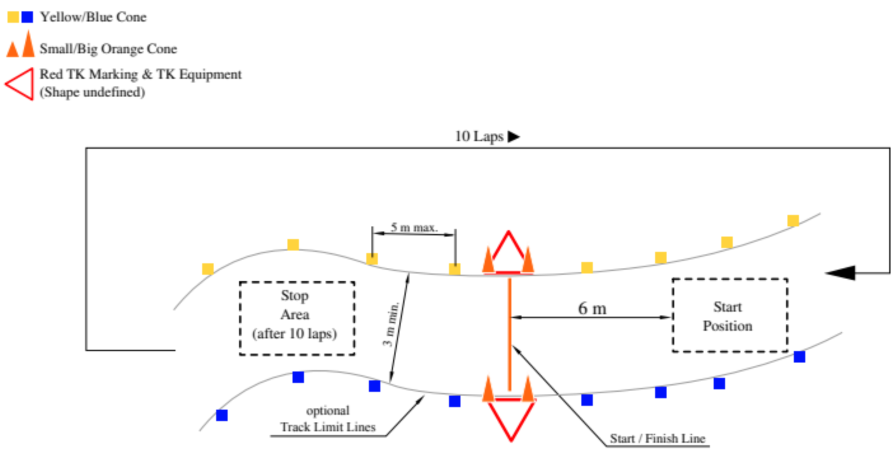

Figure 2: Cones present in a Formula Student Trackdrive circuit. Yellow cones are placed on the right side of the circuit whilst blue cones are placed on the left. Orange cones denote the start line of the circuit. Picture from the FSG Competition Handbook [1].

1.2 BCN eMotorsport

The BCN eMotorsport driverless project consists in the design and construction of a driverless car, the CAT13d (commonly known as XALOC 2.0) which is shown in Figure 3. The project started in 2018, with the main goal of designing and building the first

Spanish Formula Student Driverless car. This goal was successfully accomplished and the team finished all the dynamic events in Formula Student Spain. Two years later (with a year without many improvements because of the COVID-19), the team started the 2020-2021 season with the objective of optimizing its autonomous systems in order to achieve higher speeds and improve its competitiveness to become one of the top European teams. Therefore, the algorithms presented in this document were developed to achieve the 2020-2021 team objective.

Figure 3: XALOC 2.0. First and fastest Formula Student Driverless car in Spain.

Previously to detail the algorithms developed in this project, a brief description of all the XALOC 2.0 estimation and control algorithms developed by different members of the team is detailed in Section 2 The main algorithms developed in this project, localization and mapping, and controller algorithms, are detailed in Sections 3 and 4, respectively.

1.3 Objectives

The main purpose of this project is to increase the competitiveness of the XALOC 2.0 by increasing the speeds that the car can reach, reducing lap time and ensuring the reliability and stability of the vehicle. In order to fulfill this purpose, a set of requirements and specifications, which are detailed below, must be taken into account.

1.4 Requirements

The requirements of the Localization and Mapping, and Controller algorithms are detailed in the following subsections.

1.4.1 Localization and Mapping Requirements

The Localization and Mapping requirements are:

- The algorithm must be able to identify all the landmarks (cones) of a circuit using the information given by the perception system and the inertial sensors.

- The algorithm cannot make errors in data association. All observations must be associated to their corresponding landmarks.
- The algorithm must be able to remove the majority of outliers introduced by the perception system.
- The algorithm has to correct the position of the landmarks.
- The error between the mapped landmarks and the landmarks ground truth must remain below 50 cm
- The computational time must remain below 1000 ms.

1.4.2 Controller Requirements

The controller requirements are:

- The algorithm has to calculate the optimal controls that reduce the error between the planned and the current states.
- The algorithm must remain reliable when reaching high speeds (70-80 km/h).
- The average computational time must remain below 50 ms because of the real-time characteristic of the algorithm.

1.5 Specifications

The specifications of the Localization and Mapping, and Controller algorithms are detailed in the following subsections.

1.5.1 Localization and Mapping Specifications

The Localization and Mapping specifications are:

- A perception system is necessary to detect the 2D positions of the landmarks.
- The vehicle has to include inertial and position sensors.
- A previous localization system capable of obtaining an initial estimation of the car trajectory is necessary.

1.5.2 Controller Specifications

The controller specifications are:

- A planned trajectory, a velocity profile, the current position and velocities of the car, the current inertial measurements and the current states of the actuators are needed to feed the controller.

- A Controller Area Network (CAN) communication protocol is needed to communicate the algorithm with the ECUs of the car. Furthermore, CAN Decoder and Encoder systems are necessary to receive and send messages.
- The vehicle needs a steering actuator system, a service brake system and an electric motor to apply the controls.
- A global race optimizer is needed to compute an optimal global trajectory that ensures the stability of the vehicle at high speeds.

1.6 Methods and procedures

This project is part of the estimation and control architectures of the BCN eMotorsport team for the 2021-2021 season. This means that this project is developed during a whole one year period (September 2020 to August 2021). Furthermore, the goals of this project are constantly changing, as well as its specifications and requirements because of the new challenges that the team is encountering when developing and testing the algorithms. Developing this project in BCN eMotorsport requires a constant communication with the members of the team as well as performing other tasks that are not related to the algorithms described in this document.

The estimation and control team starts the season by identifying which algorithms are necessary to improve the competitiveness and reliability of the car. Then, after a previous research, the design and subsequent implementation of the algorithms is performed. Finally, the algorithms are tested in simulation and integrated into the vehicle. A description of the algorithms implemented for the estimation and control architectures can be found in Appendix A.1.

This thesis details the design, implementation and results of two algorithms: the localization and mapping algorithm, and the controller algorithm. Both algorithms were developed following the steps explained above.

1.7 Work plan

In order to plan the required tasks to develop this project, a Gantt diagram was developed, as well as milestones and deliverables. Furthermore, the incidences and deviations encountered while developing the project are exposed.

1.7.1 Gantt Diagram

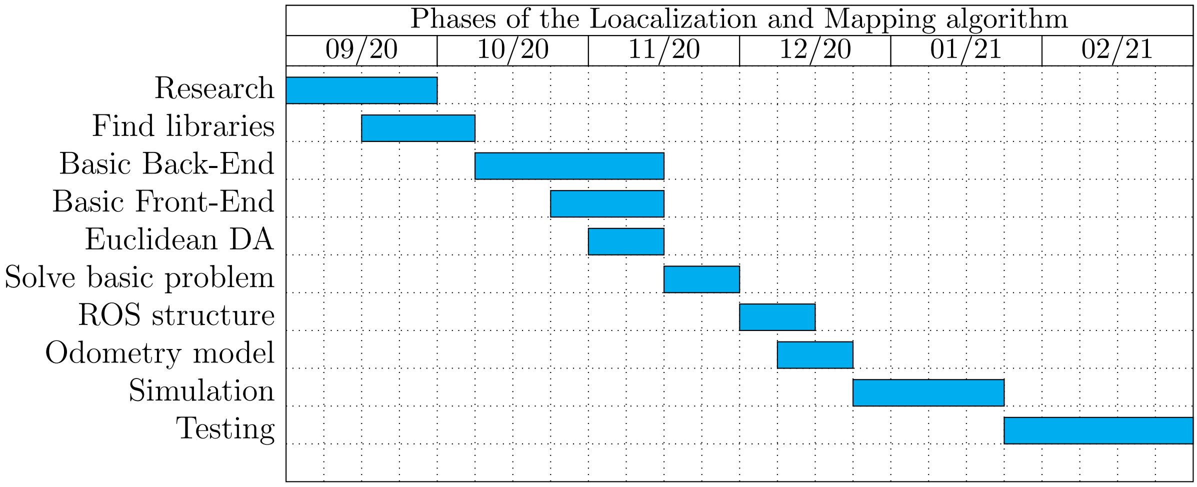

Figure 4: Gantt diagram of the Localization and Mapping.

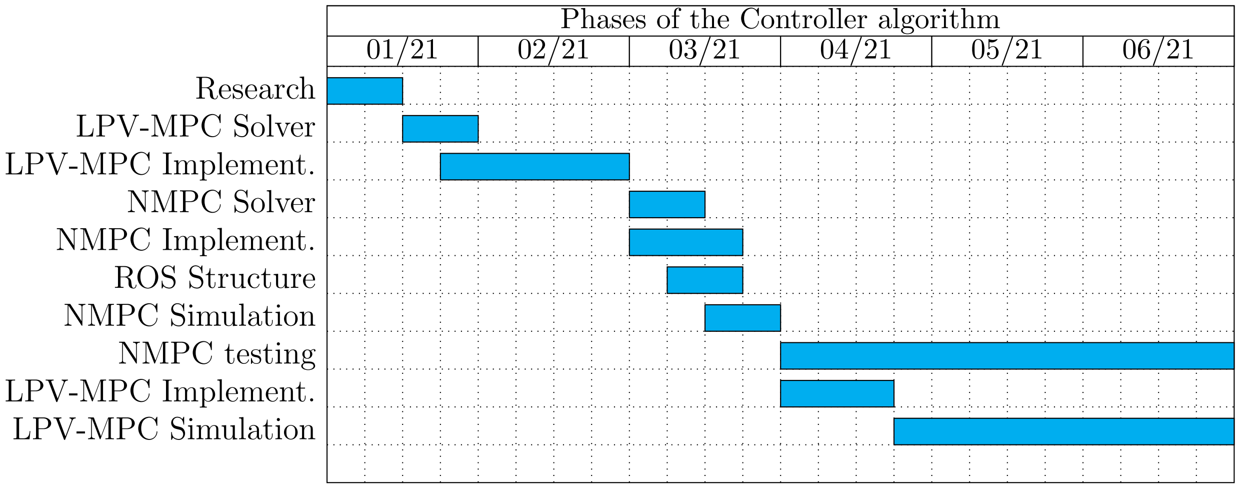

Figure 5: Gantt diagram of the Controller.

1.7.2 Milestones and deliverables

SLAM Milestones and deliverables

Table 1: SLAM Milestones and deliverables.

Short Title	Milestone / deliverable	Date (week)
Research	Understand basic algorithm and mathematical background	30/09
Find libraries	Find a library capable to implement the solver	7/10
Basic Back-End	Implement the Back-End of the algorithm using the solver	13/11
Basic Front-End	Implement the Front-End	13/11
Euclidean DA	Implement the Euclidean Data Association	13/11
Solve basic problem	Solve easy problem with given inputs	30/11
ROS structure	Add ROS callbacks, subscribers and publishers	14/12
Odometry model	Add vehicle odometry model to predict states	20/12
Simulation	Simulate the algorithm with the estimation and control pipeline	22/01
Testing	Test the algorithm with real data and integrate it into the car	27/02

MPC Milestones and deliverables

Table 2: MPC Milestones and deliverables.

Short Title	Milestone / deliverable	Date (week)
Research	Understand basic algorithm and mathematical background	20/01
LPV-MPC Solver	Implement the QCQP solver using FORCESPRO	30/01
LPV-MPC Implementation	Implement the algorithm in C++	24/04
NMPC Solver	Implement the NLP solver using FORCESPRO	15/03
NMPC Implementation	Implement the algorithm in C++	17/03
ROS Structure	Add ROS callbacks, rate, subscribers and publishers	17/03
NMPC Simulation	Simulate the NMPC with the rest of the control pipeline	30/03
NMPC testing	Integrate the NMPC in the vehicle and perform real testing	30/06
LPV-MPC Simulation	Simulate the LPV-MPC with the rest of the control pipeline	30/06

1.7.3 Deviations and incidences

Regarding the implementation of the controller algorithm (Model Predictive Control), the initial approach was to apply a linearization of the problem in order to decrease the computational time of the algorithm. However, as this approach did not initially yield the desired results, the controller was implemented without the linearization. Once the nonlinear controller was implemented and tested, the linear controller was developed and successfully implemented.

Furthermore, as we did not have unlimited testing time and testing sites, the period to test the algorithms integrated into our car took a long time. In addition, some hardware problems appeared and a considerable amount of time was needed to solve them.

As for the Localization and Mapping algorithm, its implementation went as planned and we did not encounter any incidence or deviation.

2 State of the art in Formula Student Estimation and Control

In 2017, Academic Motorsports Club Zurich (AMZ) impressed the Formula Student Driverless world with their Driverless Vehicle (see Autonomous Racing: AMZ Driverless with flüela). Two years later, in 2019, they published a paper describing their autonomous system [2], in which they detailed their perception, estimation and control algorithms. This publication was a breakthrough in the Formula Student world, as it had proven that AMZ autonomous system was able to reach high speeds and complete dynamic events.

Regarding Simultaneous Localization and Mapping (SLAM), AMZ proposed a FastSLAM technique to localize the vehicle and map the circuit. They also pointed out the possibility of improving their system by including a Graph-Based SLAM algorithm. As for their controller, AMZ used a Model Predictive Control (MPC) with a dynamic bicycle model to calculate the optimal controls. Furthermore, their controller took advantage of the four wheel drive (4WD) technology implemented in their car to achieve more accurate solutions.

In 2019, the Karlsruher Institut für Technologie team (KA-RaceIng) published a paper describing their full autonomous software stack [3]. Regarding the SLAM problem, KA-RaceIng developed and Extended Kalman Filter SLAM with a complex data association process. For their controller, Ka-RaceIng decided to implement an MPC with a dynamic bicycle model to control the lateral dynamics of their vehicle, whereas their longitudinal dynamics were controlled by a simple proportional, integral and derivative (PID) controller. Their approach proved to be successful as they won many dynamic events such as the FS Germany 2019 Autocross.

These two teams proved the necessity of using a predictive controller capable of taking into account the vehicle dynamics as well as the need of precise SLAM techniques to map the circuit as accurately as possible.

3 Model Predictive Control

3.1 Principles

Model Predictive Control (MPC) is a method of process control that identifies the dynamics of a system, its states and its physical and forced constrains with the objective of minimizing a cost function which includes the desired functioning of the system. This cost function is globally minimized using the predictions of the robot states in a limited future called horizon. Therefore, MPC uses future information to compute the controls and states in this limited horizon, thus achieving more accurate results. A model capable of representing the dynamics of the system is necessary to compute the states predictions. This is why a standard MPC consists of three basic parts: the system dynamics model, the objective function and the constrains. Figure 6 represents the states predicted by an MPC using the predicted controls, as well as the desired or reference states and the current state.

This control method uses two different vectors to define the system. These vectors are the system states and controls. From this point onwards, these vectors will be referenced as $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively. The state vector, $\boldsymbol{x}$, represents the current state of the system at a certain time step k. The control vector, $\boldsymbol{u}$, represents the actions that the actuators need to perform at time step k in order to reach the desired states at time step $k+1$.

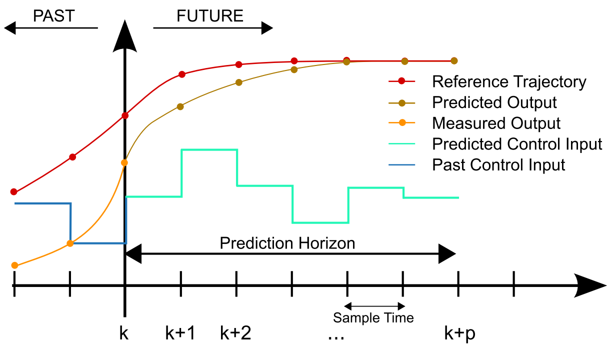

Figure 6: Representation of reference, current and predicted states and predicted controls. Picture from Wikipedia.

3.1.1 System dynamics

System dynamics describe the behaviour of a system when a certain control $\boldsymbol{u}_k$ is applied. Therefore, an MPC needs a mathematical model to represent these dynamics. A model can either be linear or nonlinear depending on the system. When the model is linear, fast MPC implementations can be achieved and problems are convex if the objective function is quadratic. However, convexity is not ensured when using nonlinear models and, furthermore, the computational cost is more expensive.

System dynamics are modeled by differential equations, which are, in general, a function of states and controls and are represented as:

$$\dot{\boldsymbol{x}} \quad = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{1}$$

where $\dot{\boldsymbol{x}}$ denotes $\frac{d\boldsymbol{x}}{dt}$.

When equation (1) is linear, its formulation can be simplified and represented as:

$$\dot{\boldsymbol{x}} \quad = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{2}$$

3.1.2 Objective function

The objective function represents the desired behavior of the system. It must penalize the error between the desired and current system behavior. Therefore, many objective functions consist of the quadratic difference between the desired and predicted states. Furthermore, in order to reduce the controls slew rate ($\Delta\boldsymbol{u}$), a quadratic penalizing term regarding this slew rate is generally included in the objective function as well. Thus, objective functions are generally a function of $\boldsymbol{x}$, $\boldsymbol{u}$ and $\Delta\boldsymbol{u}$ and are represented by

$$J(\boldsymbol{x}, \boldsymbol{u}, \Delta\boldsymbol{u}) \tag{3}$$

3.1.3 Constrains

In general, controllers need to take into account the physical constraints of systems, as actuators have limited ranges of actions. Furthermore, forced constraints can be applied by the controller for different reasons in order to achieve desired behaviors. These constraints are introduced into the MPC via a set of linear and nonlinear inequalities represented in the following equations:

$$\underline{\boldsymbol{z}} \leq \boldsymbol{z} \leq \bar{\boldsymbol{z}} \tag{4}$$

$$\underline{\boldsymbol{h}}(\boldsymbol{z}) \leq \boldsymbol{h}(\boldsymbol{z}) \leq \bar{\boldsymbol{h}}(\boldsymbol{z}) \tag{5}$$

where $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \\ \Delta\boldsymbol{u} \end{bmatrix}$, underlined and overlined variables denote the lower and upper bounds, respectively, and $\boldsymbol{h}(\boldsymbol{z})$ is a nonlinear function.

3.1.4 General nonlinear MPC

Many real applications are modeled as nonlinear systems and, therefore, the use of nonlinear MPCs is necessary. The standard formulation of these MPCs is to find the minimum of

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=1}^{N} J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k) \\
\text{subject to} \quad & \dot{\boldsymbol{x}}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\
& \boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta\boldsymbol{u}_k \\
& \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \\
& \Delta\boldsymbol{u}_{min} \leq \Delta\boldsymbol{u}_k \leq \Delta\boldsymbol{u}_{max} \\
& \boldsymbol{x}_{min} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{max} \\
& \underline{\boldsymbol{h}}(\boldsymbol{x}_k) \leq \boldsymbol{h}(\boldsymbol{x}_k) \leq \bar{\boldsymbol{h}}(\boldsymbol{x}_k) \\
& \underline{\boldsymbol{h}}(\boldsymbol{u}_k) \leq \boldsymbol{h}(\boldsymbol{u}_k) \leq \bar{\boldsymbol{h}}(\boldsymbol{u}_k) \\
& \boldsymbol{x}_1 = \boldsymbol{x}_{init} \\
& \boldsymbol{u}_1 = \boldsymbol{u}_{init}
\end{aligned} \tag{6}
$$

3.2 Dynamic Model

Vehicle dynamics are governed by highly empirical nonlinear models. Therefore, the task of accurately modeling the dynamics of a racing vehicle is remarkably difficult. A precise model should include the behavior of tires, weight distribution, steering, drivetrain, suspension and aerodynamics as described by W. Milliken and D. Milliken. in [4].

3.2.1 Dynamic Bicycle Model

If we assume that Formula Student cars race in flat surfaces and that the load transfer can be neglected, a dynamic bicycle model can be used to represent the dynamics of our race car. This model assumes that the system lives in the two dimensional special Euclidean group (SE(2)), which means that the model is governed by two translational and a rotational degrees of freedom. This is the reason why pitch, roll and the z components of the model can be neglected. This model has proven to be accurate enough to control Formula Student cars, as top teams such as AMZ and KA-RaceIng use this very same model in their controllers [2] [3]. Only longitudinal, lateral and yaw dynamics are taken into account by this model. A representation of the dynamic bicycle model can be visualized in Figure 7. It is described by the following set of differential equations:

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos\Theta - v_y \sin\Theta \\ v_x \sin\Theta + v_y \cos\Theta \\ \omega \\ \frac{1}{m}(F_x - F_{y,F}\sin\delta) + \omega v_y \\ \frac{1}{m}(F_{y,F}\cos\delta + F_{y,R}) - \omega v_x \\ \frac{1}{I}(F_{y,F}L_f\cos\delta - F_{y,R}L_r) \end{bmatrix} = \boldsymbol{f}_d(\boldsymbol{x}, \boldsymbol{u}, \Delta\boldsymbol{u}) \tag{7}
$$

In equation (7), one can distinguish a state vector and a control vector. The car state is denoted as $\boldsymbol{x} = [x, y, \Theta, v_x, v_y, \omega]^T$, where $[x, y, \Theta]$ represent the position and heading in the global frame and $[v_x, v_y, \omega]$ describe the longitudinal and lateral velocities and the yaw rate at the car center of gravity (CoG), respectively. The control vector $\boldsymbol{u} = [\delta, a]^T$ represents the front tires steering angle and the acceleration applied by the drivetrain.

$F_{y,F}$ and $F_{y,R}$ are the lateral forces exerted by the front and rear tires, respectively. F_x represents the combined longitudinal force. L_f and L_r are the distances from the center of gravity to the front and rear axles, respectively. The car's mass and inertia are denoted as m and I. Finally, α_f and α_r represent the front and rear tires slip angle.

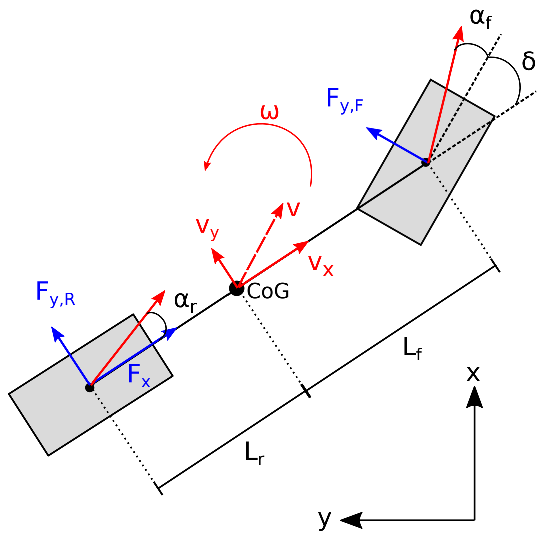

Figure 7: Dynamic Bicycle Model.

3.2.2 Longitudinal Force

In this model, the combined longitudinal force F_x can be calculated using equation (8), which is the sum of the drivetrain, drag and rolling resistance forces. Note that the acceleration a is applied as a torque command to the rear wheels only, as our car does not have a four wheel drive technology. The drag force is modeled by the drag coefficient C_d which is experimentally obtained, the frontal cross-sectional area A_r and the air density ρ. Finally, the rolling resistance force depends on the car weight and the rolling resistance coefficient μ_r, which is empirically obtained as well

$$F_x = ma - \frac{1}{2} C_d \rho A_r v_x^2 - \mu_r mg \tag{8}$$

3.2.3 Lateral Forces

The lateral tire forces $F_{y,F}$ and $F_{y,R}$ are highly important in the dynamic bicycle model as tires are the only part of the vehicle interacting with the surface. The Simplified Magic Formula (9) was chosen to model the lateral forces. This model was developed by Hans B. Pacejka [5] and it has been widely used to model tire dynamics.

$$\begin{aligned} F_{y,F} &= D_F F_{z,F} \sin(C_F \arctan(B_F \alpha_f)) \\ F_{y,R} &= D_R F_{z,R} \sin(C_R \arctan(B_R \alpha_r)) \end{aligned} \tag{9}$$

Coefficients D_i, C_i and B_i, with $i \in \{F, R\}$ depend on the normal force $F_{z,i}$, the tire pressure, camber, slip ratio and many other tire characteristics. They are empirically obtained by neglecting and setting constant some of these parameters. Furthermore, if we neglect the load transfer effect, the normal force $F_{z,i}$ exerted at the bicycle model front and rear tires can be calculated as following:

$$\begin{aligned} F_{z,F} &= mg\frac{L_R}{L_F + L_R} \\ F_{z,R} &= mg\frac{L_F}{L_F + L_R} \end{aligned} \tag{10}$$

The front and rear slip angles (α_f, α_r) are the angles formed by the longitudinal and lateral velocities at each tire. These angles can be calculated by means of the following equations:

$$\begin{aligned} \alpha_f &= \arctan(\frac{v_y + L_f\omega}{v_x}) - \delta \\ \alpha_r &= \arctan(\frac{v_y - L_r\omega}{v_x}) \end{aligned} \tag{11}$$

3.3 Formulating the Nonlinear MPC

In Section 3.2, the dynamics of a Formula Student car have been presented. As the model is highly nonlinear, a nonlinear MPC (NMPC) is necessary. Even though NMPCs tend to require more computational time, the use of FORCESPRO [6] [7], a fast solver specially tailored to solve MPC problems, allowed us to use NMPC.

As explained in Section 3.1, the MPC requires a system dynamics model, an objective function and a set of constrains. In the following sections, these modules will be formulated, altogether with the solver formulation.

3.3.1 System dynamics

Section 3.2 describes the dynamic model used to predict the next states in our MPC. However, the model only states the differential equations (7), which must be integrated to predict the next state. A general method to integrate the differential equation is the Euler method, which is represented as follows

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k)T_s \tag{12}$$

where T_s denotes the sampling time.

However, a more precise integration can be achieved by using an explicit 4-th order Runge-Kutta integrator [8], which is widely used in nonlinear MPCs. Therefore, the integration model becomes:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_{RK}(\boldsymbol{x}_k, \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k), T_s) \tag{13}$$

3.3.2 Objective Function

The main objective of the controller is to follow the trajectory and velocity profile computed by the planner and the GRO. Therefore, the job of the solver is to find the controls, δ and a, that make the car follow the desired trajectory with the smallest possible error. For this reason, a cost function that minimizes the quadratic error between the desired trajectory and velocities and the computed states is necessary.

Furthermore, in order to minimize the nervousness of the system, the cost function should include a term that minimizes the controls' slew rate. Otherwise, the solver could find local minimums which minimize the error between the planner and solver states but with unfeasible changes in the control variables that would destabilize the system.

Therefore, our objective function follows equation (14), where Q and R represent the weights related to the error between the states with respect to the planned trajectory and the controls' slew rate, respectively. In order to be consistent with standard MPC formulations, equation (15) must be fulfilled. This final equation is due to the normalization of the cost function, which is useful to correctly adapt the weights to achieve a desired behavior of the MPC.

$$J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k) = Q \cdot q(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k) + R \cdot r(\Delta\boldsymbol{u}_k) \tag{14}$$

$$R + Q = 1 \tag{15}$$

In equation (14), $q(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k)$ denotes the planner reference error term and $r(\Delta\boldsymbol{u}_k)$ represents the slew rate term.

Planner Reference Error

The main goal of the planner reference error is to follow the planned trajectory and velocities as accurately as possible. The planner consists of a set of points $\boldsymbol{p}_k = [p_{x_k}, p_{y_k}, p_{\Theta_k}, p_{vx_k}]^T$, containing a global position (p_{x_k}, p_{y_k}), a heading (p_{Θ_k}) and a desired longitudinal velocity (p_{vx_k}).

A first approach would be to directly minimize the global positions, the heading and the longitudinal velocity. However, a better approach is to minimize the longitudinal and lateral error between the computed position of the car and the planner positions, which can be achieved with a simple transformation using the planner heading. In this approach, the error equations are:

$$q(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k) = \frac{Q_{lon}}{\epsilon_{lon}^2} e_{lon}^2(\boldsymbol{x}_k, \boldsymbol{p}_k) + \frac{Q_{lat}}{\epsilon_{lat}^2} e_{lat}^2(\boldsymbol{x}_k, \boldsymbol{p}_k) + \frac{Q_{v_x}}{\epsilon_{v_x}^2} e_{v_x}^2(\boldsymbol{x}_k, \boldsymbol{p}_k) \tag{16}$$

$$\begin{aligned} e_{lon}(\boldsymbol{x}_k, \boldsymbol{p}_k) &= (x_k - p_{x_k}) \sin p_{\Theta_k} - (y_k - p_{y_k}) \cos p_{\Theta_k} \\ e_{lat}(\boldsymbol{x}_k, \boldsymbol{p}_k) &= -(x_k - p_{x_k}) \cos p_{\Theta_k} - (y_k - p_{y_k}) \sin p_{\Theta_k} \\ e_{v_x}(\boldsymbol{x}_k, \boldsymbol{p}_k) &= v_{x_k} - p_{vx_k} \end{aligned} \tag{17}$$

Figure 8 shows a graphical visualization of the longitudinal (e_{lon}) and lateral (e_{lat}) errors of equations (17). Third equation in (17) shows that the error between the computed and planned longitudinal velocities (e_{v_x}) is directly introduced into the cost function.

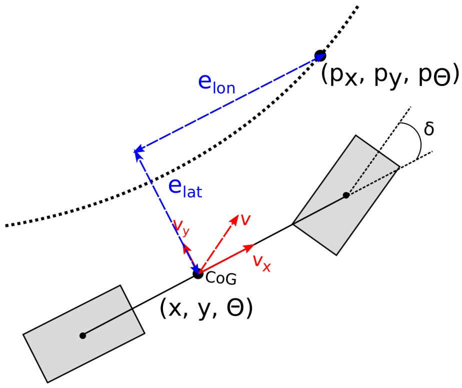

Figure 8: Longitudinal and lateral errors.

For each time-step k inside the horizon N, equation (16) is introduced into the cost function (14). Note that each quadratic term inside the equation (16) is multiplied by a weight term $\frac{Q_i}{\epsilon_i^2}$. The weight $Q_i \in [0, 1]$ represents the importance given to each quadratic error term. This factor must be defined such as its gains weight when its corresponding error is important and the system could be destabilized. For instance, in our case the lateral error is very important and should be kept as small as possible, therefore Q_{lat} must be considerably incremented. Furthermore, in order to ease the parameters tuning the following equation (18) is included.

$$Q_{lon} + Q_{lat} + Q_{v_x} = 1 \tag{18}$$

Finally, the term ϵ_i is used in equation (16) as a normalizing factor. This term represents the range of the quadratic term, therefore its value is equal to the maximum achievable error. This factor is necessary because each error term has a different range. This means that two errors with the same value do not have the same impact to the result if they are not normalized. For instance, the longitudinal velocity error has a larger range and does not affect the result as much as the lateral error, which has an smaller error range.

Note that the error between the yaw angle, Θ, and the planned yaw angle, p_Θ, is not minimized in equation (16) because it yields wrong behaviors when using the planner computed by the GRO, as this error comes from the initial angle difference between the current state of the car and the planned yaw angle of the first step. In addition, note that the angle is optimized when the lateral and longitudinal errors are minimized. Therefore, trying to follow the planned yaw angle would increment these errors.

AMZ [2] and Driverless UPC [9] (using a kinematic model) used a similar approach with the minimization of the lateral and longitudinal errors. However, AMZ maximized the velocity instead of following a velocity profile and Driverless UPC added a minimization of the yaw angle error.

Slew Rate Term

The slew rate term is used to penalize large changes applied to steering angle and acceleration commands. It is introduced in the objective function using the following equation:

$$\begin{gathered} r(\Delta \boldsymbol{u}_k) = \Delta \boldsymbol{u}_k^T \boldsymbol{R} \Delta \boldsymbol{u}_k \\ \boldsymbol{R} = \begin{bmatrix} \frac{R_\delta}{\epsilon_\delta^2} & 0 \\ 0 & \frac{R_a}{\epsilon_a^2} \end{bmatrix} \\ \Delta \boldsymbol{u}_k = \boldsymbol{u}_k - \boldsymbol{u}_{k-1} \end{gathered} \tag{19}$$

where ϵ_δ and ϵ_a are normalizing factors as explained in Planner Reference Error section. The terms R_δ and R_a denote the weights given to the steering angle and acceleration slew rates, respectively. They must fulfill the following equation:

$$R_\delta + R_a = 1 \tag{20}$$

Note that the slew rate term must always be quadratic because changes in the control variables can reach negative values.

3.3.3 Constraints

Control constraints are necessary as our actuators have a physical limited range of actions and some solutions are unfeasible. Furthermore, some state constraints can be introduced in order to have more control over the solutions.

Control Constraints

In our design control constraints over the control vector $\boldsymbol{u}_k = [\delta_k, a_k]^T$ are applied over both the steering angle and acceleration. These constraints are described in the following inequations:

$$\begin{aligned} -\delta_{max} \leq \delta_k \leq \delta_{max} \\ a_{dec\_max} \leq a_k \leq a_{acc\_max} \end{aligned} \tag{21}$$

where δ_{max} represents the maximum steering angle that the actuator can apply to both directions of the steering wheel and a_{dec_max} and a_{acc_max} denote the maximum deceleration and acceleration that the brakes and motor can apply, respectively. We denote $\boldsymbol{u}_{min} = [-\delta_{max}, a_{dec\_max}]^T$ and $\boldsymbol{u}_{max} = [\delta_{max}, a_{acc\_max}]^T$.

Furthermore, constraints over the slew rate, $\Delta\boldsymbol{u}_k = [\Delta\delta_k, \Delta a_k]$, are applied as well. These constraints follow inequalities:

$$\begin{aligned} -\Delta\delta_{max} \leq \Delta\delta_k \leq \Delta\delta_{max} \\ \Delta a_{dec\_max} \leq \Delta a_k \leq \Delta a_{acc\_max} \end{aligned} \tag{22}$$

The goal of these constraints is to minimize the number of solutions containing large changes of the control variables which would not make sense and would most likely be local minimums of the MPC problem.

State Constraints

In our case state constraints over the state vector $\boldsymbol{x}_k = [x_k, y_k, \Theta_k, v_{x_k}, v_{y_k}, \omega_k]^T$ are only applied over velocities. For this reason, the constraints are defined by the following inequations:

$$\begin{aligned} -\infty \leq x_k \leq \infty \\ -\infty \leq y_k \leq \infty \\ -\infty \leq \Theta_k \leq \infty \\ v_{x_{min}} \leq v_{x_k} \leq v_{x_{max}} \\ v_{y_{min}} \leq v_{y_k} \leq v_{y_{max}} \\ \omega_{min} \leq \omega_k \leq \omega_{max} \end{aligned} \tag{23}$$

Note that $\boldsymbol{x}_{min} = [-\infty, -\infty, -\infty, v_{x_{min}}, v_{y_{min}}, \omega_{min}]^T$ and $\boldsymbol{x}_{max} = [\infty, \infty, \infty, v_{x_{max}}, v_{y_{max}}, \omega_{max}]^T$.

3.3.4 Formulation

At this point, we have defined all the elements necessary for the NMPC formulation, which are integrated to define the following minimization problem:

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=1}^{N} J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k) \\
\text{s.t} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}_{RK}(\boldsymbol{x}_k, \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k, \Delta\boldsymbol{u}_k), T_s) \\
& \boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \Delta\boldsymbol{u}_k \\
& \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \\
& \Delta\boldsymbol{u}_{min} \leq \Delta\boldsymbol{u}_k \leq \Delta\boldsymbol{u}_{max} \\
& \boldsymbol{x}_{min} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{max} \\
& \boldsymbol{x}_1 = \boldsymbol{x}_{init} \\
& \boldsymbol{u}_1 = \boldsymbol{u}_{init}
\end{aligned}
\tag{24}
$$

Finally, the last steps are to modify this formulation in order to produce a suitable formulation for the used solver (FORCESPRO) and to transform the NMPC outputs to the actuators desired inputs. These procedures can be found in Appendix A.2.

3.4 NMPC Implementation

Following the control structure described in the Appendix section A.1.3, the MPC has been implemented using a ROS framework in C++. The basic MPC block diagram is represented in Figure 9. The basic algorithm in pseudo-code as well as an explanation of each block of the MPC is presented in Section 3.4.1.

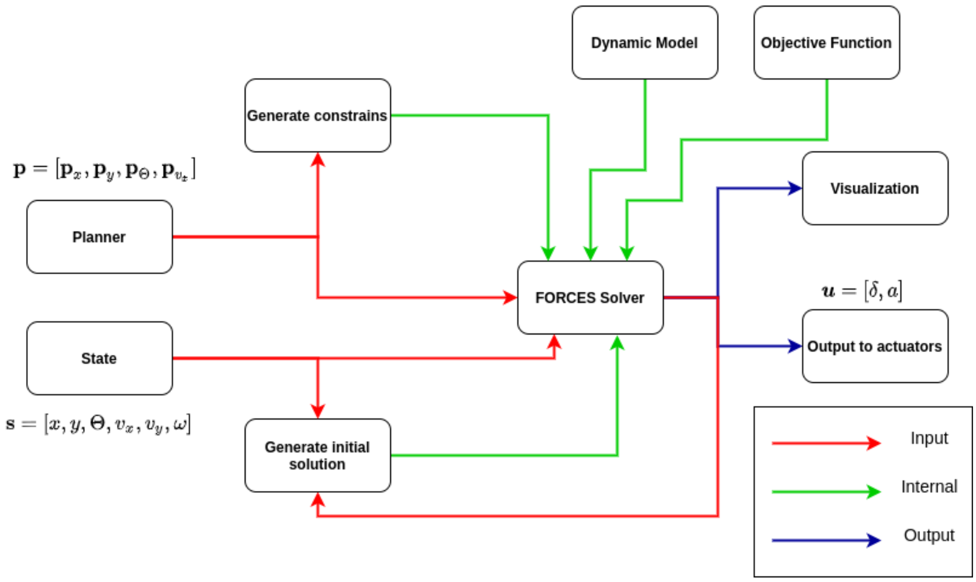

Figure 9: MPC block diagram.

Figure 10 shows how the MPC algorithm described in figure 9 is included into the ROS structure. It can be seen in this Figure that the MPC uses the current car state (`/state/car`) and the planned trajectory (`/planner/global` or `/gro/planner`) in order to compute the car commands which are included in `/state/commands`. The message `/state/debug` contains the current and computed velocities and the individual costs of each term in the objective function. This message is included for debugging purposes.

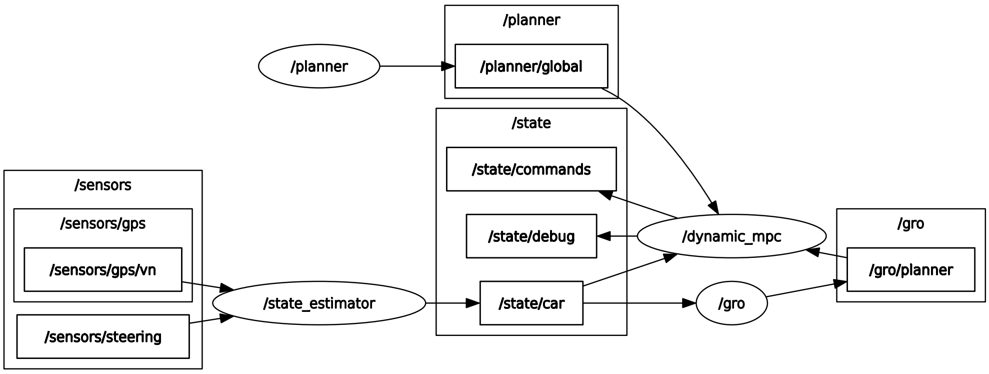

Figure 10: MPC ROS graph. ROS topics (messages) are represented with boxes. ROS nodes (programs)are represented with ellipses. Arrows denote a relationship between a topic end its publishers and subscribers.

3.4.1 Algorithm

This section includes a detailed description of each part of the algorithm used to implement the MPC. The algorithm used to implement the MPC block diagram represented in Figure 9 is detailed in Algorithm 1 below.

In step 3, the algorithm starts by initializing the parameters vector, which needs to be passed to the FORCES solver. This vector, $\boldsymbol{t}$, includes the following variables: Q, Q_{lon}, Q_{lat}, Q_{v_x}, R, R_δ, R_a, m, I, L_R, L_F, D_R, D_F, C_R, C_F, B_R, B_F, μ_r, g, C_d, ρ, A_r and $\boldsymbol{p}$. All these parameters, except for the planner $\boldsymbol{p}$, can be dynamically modified using the ROS plugin *rqt_reconfigure*, as seen in steps 7 and 8.

Then, the MPC node reads the planner and state topics mentioned in Figure 10 using the callback functions (steps 9 and 10). The planner objectives are saved in vector $\boldsymbol{p}$ and the information from the state topic is stored in vector $\boldsymbol{s} = [x, y, \Theta, v_x, v_y, \omega, \delta_0, a_{x0}]^T$. The yaw values p_{Θ_k} of the planner $\boldsymbol{p}$ have to be modified so that the angles remain continuous (step 11). This is necessary because the Path Planning and Global Race Optimizer (GRO) algorithms return a planner with $p_{\Theta_k} \in [-\pi, \pi]$ and the solver fails when the angles are not continuous. For instance, two consecutive yaw angles $p_{\Theta_k} = -\pi + \epsilon_1$ and $p_{\Theta_{k+1}} = \pi - \epsilon_2$ (with $\epsilon_1 \geq 0$ and $\epsilon_2 \geq 0$) would make the solver fail to achieve a solution.

Afterwards, regarding the MPC constraints, an initial solution for the solver and the current state and controls must be set (steps 13 to 16). Once this is done, the FORCES solver is called (step 17). This solver is implemented in MATLAB using the High Level Interface provided by FORCESPRO, which requires an objective function, the system dynamics, the number of stages and the number of dynamic parameters. The solver is then compiled into an executable library callable in C++, which improves the solver efficiency and quickness. These necessary functions are implemented following the guidelines explained

Algorithm 1 Dynamic bicycle MPC

procedure DYNAMICMPC
    ros::Rate r(20Hz)
    $\boldsymbol{t} \leftarrow \boldsymbol{t}_{init}$ ▷ Initialize parameters
    $\boldsymbol{X}_{last} \leftarrow \mathbf{0}$
    $\boldsymbol{U}_{last} \leftarrow \mathbf{0}$

    while ros::ok **do** ▷ ROS master is running
        if ros::DynamicReconfigure **then** ▷ *rqt_reconfigure* option
            $\boldsymbol{t} \leftarrow \boldsymbol{t}_{new}$

        $\boldsymbol{p} \leftarrow$ ros::callback_planner() ▷ Update planner and state from topics
        $\boldsymbol{s} \leftarrow$ ros::callback_state()
        $\boldsymbol{t} \leftarrow$ cont_yaw($\boldsymbol{p}$, $\boldsymbol{s}$) ▷ Make yaw angles continuous
        bounds $\leftarrow [\bar{\boldsymbol{u}}, \underline{\boldsymbol{u}}, \bar{\boldsymbol{x}}, \underline{\boldsymbol{x}}, \bar{\Delta \boldsymbol{u}}, \underline{\Delta \boldsymbol{u}}]^T$ ▷ Set MPC constrains/bounds
        $\boldsymbol{X}_{init} \leftarrow \boldsymbol{X}_{last}$ ▷ Set initial solution
        $\boldsymbol{U}_{init} \leftarrow \boldsymbol{U}_{last}$
        $\boldsymbol{x}_1 \leftarrow \boldsymbol{s}(0:5)$ ▷ Set current state and controls
        $\boldsymbol{u}_1 \leftarrow \boldsymbol{s}(6:7)$

        $\boldsymbol{X}_{last}, \boldsymbol{U}_{last} \leftarrow$ FORCES_NLP_Solver(**bounds**, $\boldsymbol{X}_{last}$, $\boldsymbol{U}_{last}$, $\boldsymbol{x}_1$, $\boldsymbol{u}_1$, $\boldsymbol{t}$)
        visualization($\boldsymbol{X}_{last}$, $\boldsymbol{U}_{last}$) ▷ Visualization tool for *rviz*
        $\delta \leftarrow \boldsymbol{U}_{last}(delay, 0)$ ▷ Get controls to apply
        $a \leftarrow \boldsymbol{U}_{last}(delay, 1)$
        if $a \geq 0$ **then** ▷ Get torque and pressure
            $\Gamma \leftarrow$ get_Torque(a)
            $\mathrm{P} \leftarrow 0$
        else
            $\Gamma \leftarrow 0$
            $\mathrm{P} \leftarrow$ get_Pressure(a)

        cmd $\leftarrow [\Gamma, \delta, \mathrm{P}]^T$
        ros::Publish(**cmd**) ▷ Publish commands
        r.sleep() ▷ Sleep, while loop is done every 100ms

in Section 3.3. Regarding the MPC horizon and sampling time, the prediction horizon is set to $N = 40$ to achieve a $2s$ prediction, as the sampling time $T_s = 50ms$.

Next, a visualization tool (ROS plugin *rviz*) is used in order to see a representation of the acceleration and the steering angle values at each time–step of the prediction horizon (step 18). Then, the output values are transformed following equations (43), (44) and (45) (steps 19 to 26) and they are sent to the CAN encoder (steps 27 and 28). Finally, the algorithm sleeps during a certain amount of time so that the while loop is executed every 50ms (step 29), so that the MPC sends commands at a frequency of 20Hz, the working frequency of the Control pipeline.

Once the algorithm is implemented, the parameters tuning process can be performed. This process is detailed in Appendix A.2.

3.5 LPV–MPC Formulation

In Section 3.3, a non–linear MPC implementation to control a Formula Student Driverless vehicle has been proposed using the non–linear dynamics described in Section 3.2. However, the computational time of the non–linear MPC is a major flaw of this formulation. Even though the FORCESPRO solver can reduce its computational time, other approaches can be followed to reduce the computational time.

E. Alcalá et. al [10] [11] proposed a method that uses a linear-like dynamics of the dynamic bicycle model to achieve a linear MPC. This method, called Linear Parameter Varying Model Predictive Control (LPV-MPC), consists on representing the differential equations that describe the dynamic model using linear like representation, such that can be represented by equation (2). Furthermore, a reformulation of the cost function is applied.

In the following subsections, the Linear Parameter Varying (LPV) method is described and the generation of the new linear dynamics and cost function is detailed. In addition, the LPV-MPC formulation is presented. Note that the constraints follow the same structure defined in the NMPC (Section 3.3.3). In addition, the LPV-MPC formulation is presented.

3.5.1 LPV Dynamic Model

The set of differential equations in (7) can be represented using th linear like representation presented in Appendix A.3. From this representation, we can extract the model dynamics and create a new objective function, which are detailed below.

Linear Dynamics

From the linear differential equations obtained with the linearization process (equation (48) in Appendix A.3) the following linear system can be defined to update the states of the car:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + (\boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k) T_s \tag{25}$$

In (25), $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices that depend on the varying parameters and are defined as (see [10] and [11] for details):

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & -\hat{v}_y & 1 & 0 & 0 \\ 0 & 0 & 0 & \hat{v}_x & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & A_{11} & A_{12} & A_{13} \\ 0 & 0 & 0 & 0 & 0 & A_{22} & A_{23} \\ 0 & 0 & 0 & 0 & 0 & A_{32} & A_{33} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ B_{11} & B_{12} \\ B_{21} & 0 \\ B_{31} & 0 \end{bmatrix} \tag{26}$$

where

$$\begin{aligned} A_{11} = -\frac{\mu_r mg + \frac{1}{2}C_d \rho A_r \hat{v}_x^2}{m\hat{v}_x}, A_{12} = -\frac{C_F \sin\hat{\delta}}{m\hat{v}_x}, A_{13} = -\frac{C_F L_F \sin\hat{\delta}}{m\hat{v}_x} + \hat{v}_y, \\ A_{22} = \frac{C_R + C_F \cos\hat{\delta}}{m\hat{v}_x}, A_{23} = \frac{C_F L_F \cos\hat{\delta} - C_R L_R}{m\hat{v}_x} - \hat{v}_x, \\ A_{32} = \frac{C_F L_F \cos\hat{\delta} - L_R C_R}{I\hat{v}_x}, A_{33} = \frac{C_F L_F^2 \cos\hat{\delta} + L_R^2 C_R}{I\hat{v}_x}, \\ B_{11} = \frac{C_F \sin\hat{\delta}}{m}, B_{12} = 1, B_{21} = -\frac{C_F \cos\hat{\delta}}{m}, B_{31} = -\frac{C_F L_F \cos\hat{\delta}}{I} \end{aligned} \tag{27}$$

The varying parameters change at every time-step k, therefore matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ vary as well. Note that this procedure transforms the MPC into a linear time variant problem.

3.5.2 Objective function

In Section 3.3.2, a particular objective function adapted to our model that included non-linearities was defined (see equation (16)). However, in order to follow the standard formulation of linear MPCs, the objective function of the LPV-MPC is quadratic with respect the difference between the states and the references, as it can be seen in the objective function below:

$$J_{LPV_k} = (\boldsymbol{x}_k - \boldsymbol{p}_k)^T \boldsymbol{Q} (\boldsymbol{x}_k - \boldsymbol{p}_k) + \Delta \boldsymbol{u}_k^T \boldsymbol{R} \Delta \boldsymbol{u}_k \tag{28}$$

Matrices Q and R represent the weights given to each quadratic difference. These weight matrices are diagonal and have the following structure:

$$\boldsymbol{Q} = Q \begin{bmatrix} \frac{Q_{lon}}{\epsilon_{lon}^2} & & & & & & \\ & \frac{Q_{lat}}{\epsilon_{lat}^2} & & & & & \\ & & \frac{Q_{theta}}{\epsilon_{theta}^2} & & & & \\ & & & 0 & & & \\ & & & & \frac{Q_{vx}}{\epsilon_{vx}^2} & & \\ & & & & & 0 & \\ & & & & & & 0 \end{bmatrix}, \boldsymbol{R} \ = R \begin{bmatrix} R_\delta & 0 \\ 0 & R_a \end{bmatrix} \tag{29}$$

As in the NMPC formulation, the weights follow the same constraints and their sum must be equal to 1. Note that a new term Q_{theta} is included in the states weight matrix, as, unlike the NMPC formulation, this term is minimized in the LPV formulation.

3.5.3 LPV_MPC minimization problem

Using the linear system dynamics and cost function of the LPV formulation, the following linear MPC problem is proposed:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=1}^{N} [(\boldsymbol{x}_k - \boldsymbol{p}_k)^T \boldsymbol{Q} (\boldsymbol{x}_k - \boldsymbol{p}_k) + \Delta \boldsymbol{u}_k^T \boldsymbol{R} \Delta \boldsymbol{u}_k] \\ \text{s.t} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + (\boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k) T_s \\ & \boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \Delta \boldsymbol{u}_k \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \\ & \Delta \boldsymbol{u}_{min} \leq \Delta \boldsymbol{u}_k \leq \Delta \boldsymbol{u}_{max} \\ & \boldsymbol{x}_{min} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{max} \\ & \boldsymbol{x}_1 = \boldsymbol{x}_{init} \\ & \boldsymbol{u}_1 = \boldsymbol{u}_{init} \end{aligned} \tag{30}$$

The linear dynamics and quadratic cost function yield an important reduction in the solving time of the problem. Furthermore, the use of FORCESPRO decreases the time needed to solve the minimization problem. In this study, the convex multistage quadratically constrained program (QCQP) FORCESPRO formulation is chosen to describe the LPV-MPC. The developement of the FORCESPRO solver formulation can be found in Appendix A.3.

3.6 LPV-MPC Implementation

The LPV-MPC structure and block diagram follows the basic NMPC design defined in Section 3.4. However, a new block is introduced to initialize the matrices and vectors defined in the previous section. These variables need to be initialized using the varying parameters $\hat{v}_x$, $\hat{v}_y$ and $\hat{\delta}$. The longitudinal and lateral velocities are set to the current estimated velocities v_{x_0} and v_{y_0}. The planned longitudinal velocities p_{vx_k} could be used as well to initialize the longitudinal velocity varying parameter. However, using the current longitudinal velocity for all $\hat{v}_{x_k}$ has resulted in a better behavior. As for $\hat{\delta}$, a different value is used depending on the time-step k. Using the solution from the previous LPV-MPC iteration has proven to be a valid solution to initialize each $\hat{\delta}_k$. Therefore, a vector containing different values of the steering angle varying parameter is defined in equation (31) below, where $\delta_{last,i}$ represents the MPC solution for the previous iteration and i represents the time-step inside the prediction horizon N.

$$\hat{\boldsymbol{\delta}} = \begin{bmatrix} \delta_0 \\ \delta_{last,latency+1} \\ \delta_{last,latency+2} \\ \vdots \\ \delta_{last,N-2} \\ \delta_{last,N-1} \\ \delta_{last,N-1} \\ \vdots \\ \delta_{last,N-1} \end{bmatrix} \tag{31}$$

As in the NMPC, we use a prediction horizon of $N = 40$ and a sampling time T_s of $50ms$ to achieve a prediction of two seconds. Furthermore, the procedures explained in the Appendix section A.2.3 are applied to achieve accurate parameters to control our Formula Student vehicle.

3.6.1 Algorithm

The LPV-MPC algorithm shares many similarities with the NMPC algorithm described in Section 3.4.1. The main differences are that a few new steps are included to initialize the matrices and that the solver function is linear. The algorithm used to implement this controller is detailed in Algorithm 2.

Algorithm 2 structure follows the NMPC Algorithm 1 structure. A while loop with a working frequency of 20 Hz is implemented. Furthermore, the same commands conversions are used to send the correct commands to the actuators and a similar visualization tool is applied. In Algorithm 2, vector $\boldsymbol{t}$ contains all the parameters present in the LPV-MPC formulation (step 3 and 8), which are: $\boldsymbol{Q}$, $\boldsymbol{R}$, m, I, L_R, L_F, C_R, C_F, μ_r, g, C_d, ρ and A_r. However, new functions are introduced to complete the algorithm. Function *get_D1_c1* (step 13) initializes $\boldsymbol{c}_1$ and $\boldsymbol{D}_1$ using the current state information and following equations (52) and (54). Function *get_Dk_Ck* (step 14) fills matrices $\boldsymbol{C} = [\boldsymbol{C}_1, \boldsymbol{C}_2, \dots, \boldsymbol{C}_{N-1}]^T$ and $\boldsymbol{D} = [\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_{N-1}]^T$ using the planner information, MPC weights, car parameters and varying parameters (current velocities and last iteration steering output) and following equations (52) and (53), described in Appendix A.3. Then, functions *get_H* (step 15) and *get_f* (step 16) initialize the matrices of the cost function $\boldsymbol{H} = [\boldsymbol{H}_1, \boldsymbol{H}_2, \dots, \boldsymbol{H}_N]^T$ and $\boldsymbol{f} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \dots, \boldsymbol{f}_N]^T$ using planner information and MPC weights. Finally, the FORCESPRO QCQP solver (step 17) is used to get the outputs of the LPV-MPC. These outputs are visualized for debug purposes and are sent to the actuators.

As in the NMPC, the FORCESPRO solver is implemented in MATLAB. However, the

Algorithm 2 LPV-MPC

procedure LPV-MPC

ros::Rate r(20Hz)

$\boldsymbol{t} \leftarrow \boldsymbol{t}_{init}$ ▷ Initialize parameters

$\boldsymbol{X}_{last} \leftarrow \mathbf{0}$

$\boldsymbol{U}_{last} \leftarrow \mathbf{0}$

while ros::ok **do** ▷ ROS master is running

if ros::DynamicReconfigure **then** ▷ *rqt_reconfigure* option

$\boldsymbol{t} \leftarrow \boldsymbol{t}_{new}$

$\boldsymbol{p} \leftarrow$ ros::callback_planner() ▷ Update planner and state from topics

$\boldsymbol{s} \leftarrow$ ros::callback_state()

$\boldsymbol{p} \leftarrow$ cont_yaw($\boldsymbol{p}$, $\boldsymbol{s}$) ▷ Make yaw angles continuous

bounds $\leftarrow [\bar{\boldsymbol{u}}, \underline{\boldsymbol{u}}, \bar{\boldsymbol{x}}, \underline{\boldsymbol{x}}, \bar{\Delta}\boldsymbol{u}, \underline{\Delta \boldsymbol{u}}]^T$ ▷ Set MPC constrains/bounds

$\boldsymbol{D}_1, \boldsymbol{c}_1 \leftarrow$ get_D1_c1($\boldsymbol{s}$) ▷ Initialize matrices

$\boldsymbol{D}, \boldsymbol{C} \leftarrow$ get_Dk_Ck($\boldsymbol{s}$, $\boldsymbol{U}_{last}$, $\boldsymbol{t}$)

$\boldsymbol{H} \leftarrow$ get_H($\boldsymbol{p}$, $\boldsymbol{t}$)

$\boldsymbol{f} \leftarrow$ get_f($\boldsymbol{p}$, $\boldsymbol{t}$)

$\boldsymbol{X}_{last}, \boldsymbol{U}_{last} \leftarrow$ FORCES_QCQP_Solver(**bounds**, $\boldsymbol{D}_1$, $\boldsymbol{c}_1$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{H}$, $\boldsymbol{f}$)

visualization($\boldsymbol{X}_{last}$, $\boldsymbol{U}_{last}$) ▷ Visualization tool for *rviz*

$\delta \leftarrow \boldsymbol{U}_{last}(delay, 0)$ ▷ Get controls to apply

$a \leftarrow \boldsymbol{U}_{last}(delay, 1)$

if $a \geq 0$ **then** ▷ Get torque and pressure

$\Gamma \leftarrow$ get_Torque(a)

$\mathrm{P} \leftarrow 0$

else

$\Gamma \leftarrow 0$

$\mathrm{P} \leftarrow$ get_Pressure(a)

cmd $\leftarrow [\Gamma, \delta, \mathrm{P}]^T$

ros::Publish(**cmd**) ▷ Publish commands

r.sleep() ▷ Sleep, while loop is done every 50ms

QCQP solver is defined using the Low Level Interface provided by FORCESPRO, which requires the number of stages and the dynamic parameters.

4 Graph-Based Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a common problem in the field of robotics, as it consists in localizing the robot inside an environment whilst mapping its landmarks. When applied to a Formula Student Driverless vehicle, SLAM algorithms map the cones of the circuit and localize the vehicle inside it. As knowing the circuit beforehand gives a clear advantage to path and planning algorithms, SLAM algorithms are crucial in order to achieve higher speeds.

As it has been already explained, different algorithms capable of solving the SLAM problem for a Formula Student Driverless exists, such as EKF_SLAM, FastSLAM2.0 and Graph-Based SLAM. The following sections present a description and an implementation of a Graph-Based SLAM algorithm for a Formula Student vehicle. The algorithm consists in transforming the SLAM problem into a graph, as presented by S. Thrun et. al [12] and G. Grisetti et. al [13]. Then, a least squares approach is implemented.

4.1 Problem Statement

The car drives through an unknown environment where it describes a trajectory and recognises landmarks. The trajectory and landmarks are stored in a sequence of random variables $\boldsymbol{x}_{SLAM} = [\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_T, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_M]^T$, which contains T car positions denoted as $\mathbf{p}_t = [x_t, y_t, \Theta_t]^T \in SE(2)$ and M landmarks denoted as $\boldsymbol{m}_i = [x_i, y_i, c_i, s_i]^T$, where x_i and y_i represent the global two-dimensional position of the landmark, c_i denotes the landmark color and s_i is the landmark identifier. Furthermore, as the robot moves around the environment, it acquires a set of odometry measurements $\boldsymbol{o}_{1:T} = [\boldsymbol{o}_{01}, \boldsymbol{o}_{12}, ..., \boldsymbol{o}_{T-1\,T}]^T$, with $\boldsymbol{o}_{t-1\,t} = [\Delta x_{t-1\,t}, \Delta y_{t-1\,t}, \Delta\Theta_{t-1\,t}]^T$ ($t-1$ and t denote the indices of positions $\mathbf{p}_{t-1}$ and $\mathbf{p}_t$). The robot also gets a set of landmark observations $\boldsymbol{l}_{1:T} = [\boldsymbol{l}_0, ..., \boldsymbol{l}_T]^T$, where $\boldsymbol{l}_t = [\boldsymbol{l}_{it}, \boldsymbol{l}_{jt}, \ldots]^T$ ($\boldsymbol{l}_{it} = [x_l^{it}, y_l^{it}, c^{it}]^T$ represents the observation of landmark $\boldsymbol{m}_i$ from position $\mathbf{p}_t$, where x_l^{it} and y_l^{it} are the local positions of the landmark observation and c^{it} represents the color of the observation). In addition, each landmark observation $\boldsymbol{l}_{it}$ is associated to a landmark $\boldsymbol{m}_j$ in a process called data association.

With this information, Graph-Based SLAM algorithms aim to estimate the map and trajectory by maximizing the following conditional probability function:

$$\boldsymbol{x}^*_{SLAM} = \underset{\boldsymbol{x}_{SLAM}}{\arg\max}\, p(\boldsymbol{x}_{SLAM} | \boldsymbol{l}_{1:T}, \boldsymbol{o}_{1:T}, \boldsymbol{x}_{SLAM_0}) \tag{32}$$

where $\boldsymbol{x}_{SLAM_0}$ contains the initial position of the car and cannot be modified by the algorithm.

The following subsections introduce the odometry measurements and landmark observations models, as well as the data association process. These three elements are needed to solve equation (32). Furthermore, the mathematical development of the odometry measurements and landmark observation models, as well as the algorithm used to solve the least squares problem are detailed in Appendix A.4.

4.1.1 Odometry Measurements

A set of odometry measurements $\boldsymbol{o}_{1:T}$ is used in the Graph-Based SLAM algorithm to predict the next state of the vehicle. However, $\boldsymbol{o}_{1:T}$ can not be measured directly and therefore a model must be used to obtain $\boldsymbol{o}_{1:T}$. In particular, estimated longitudinal, lateral and angular velocities, as well as the following expressions are used to obtain the odometry measurements:

$$\begin{aligned}\Delta x &= (v_x \cos\Theta - v_y \sin\Theta)\Delta T\\ \Delta y &= (v_y \cos\Theta + v_x \sin\Theta)\Delta T\\ \Delta\Theta &= \omega\Delta T\end{aligned}$$

In the expressions above, ΔT denotes the time between consecutive odometry measurements. Note that this model is equivalent to that used in the three first equations of (7).

These measurements will be used to predict the next states of the car by accumulation using the following equations:

$$\begin{aligned}x_t &= x_{t-1} + \Delta x_{t-1}\\ y_t &= y_{t-1} + \Delta y_{t-1}\\ \Theta_t &= \Theta_{t-1} + \Delta\Theta_{t-1}\end{aligned} \tag{33}$$

Furthermore, Global Positioning System (GPS) information, denoted as $\mathbf{p}_{GPS}$, is also taken into account (see Appendix subsection A.4.3 for details).

4.1.2 Observations and Data Association

Landmark observations include 2D positions with respect to the car CoG frame and color information. These observations are associated to certain landmarks of the global map and they give important information about the position of landmarks as well as about the location of the vehicle inside the map.

Previously to implement the data association process, we need to place the landmark observations in the global coordinate frame. This is achieved by applying the following transformation, which consists of a rotation plus a translation:

$$\boldsymbol{l}_{it}^{global} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \boldsymbol{R}_{\Theta_t}\boldsymbol{l}_{it}^{x,y} \tag{34}$$

In (34), $\boldsymbol{R}_{\Theta_t} = \begin{bmatrix} \cos\Theta_t & -\sin\Theta_t \\ \sin\Theta_t & \cos\Theta_t \end{bmatrix}$ is the rotation matrix calculated at the car current position and $\boldsymbol{l}_{it}^{x,y}$ is the local position of the landmark $\boldsymbol{l}_{it}$.

Once landmark observations are expressed in the global coordinate frame, the data association process can be implemented. Basically, the goal of this process is to associate each landmark observation $\boldsymbol{l}_{it}^{global}$ to the current set of mapped landmarks $\boldsymbol{m} = [\boldsymbol{m_1}, \ldots, \boldsymbol{m}_P]^T$. An sketch of the data association process is shown in Figure 11. It is crucial that the observations are associated to their corresponding landmarks, as wrong data associations can destabilize the algorithm. In our case, the data association is done with the Euclidean distance metric detailed in the following subsection.

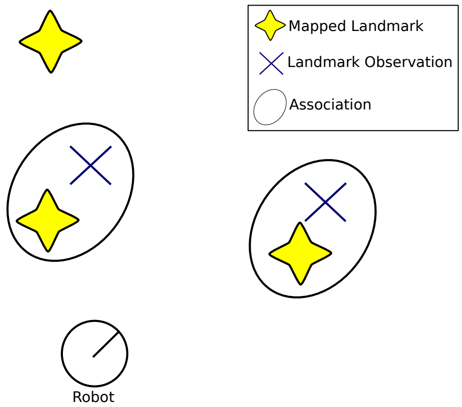

Figure 11: Data association process. Two observations are associated to two mapped landmarks.

Euclidean Distance

Euclidean distance data association consists in computing the Euclidean distance between the observations and the mapped landmarks to decide the correct association. Once the P Euclidean distances, $\boldsymbol{d}_{it}^{euc} = [d_{it,1}^{euc}, \ldots, d_{it,P}^{euc}]$, between observation $\boldsymbol{l}_{it}^{global}$ and the set of P current landmarks are computed, the landmark with the minimum euclidean distance is chosen and its distance is compared to a threshold (χ^{euc}). If the minimum Euclidean distance is smaller than χ^{euc}, then the landmark observation $\boldsymbol{l}_{it}^{global}$ is associated to the landmark with the minimum Euclidean distance. Otherwise, a new landmark is created and added to the map $\boldsymbol{m}$.

Furthermore, in order to decrease the computational time of computing all the Euclidean distances, a KD-Tree algorithm [14] is used to find the K nearest neighbors (nearest landmarks) of the landmark observation. This algorithm achieves a $\mathcal{O}(\log n)$ time complexity, compared to the $\mathcal{O}(n)$ of computing all the Euclidean distances between the observation and the landmarks. Thus, the Euclidean distance between the landmark observations and the subset of K landmarks is computed and the association is done as explained above.

This algorithm is detailed in Algorithm 3 of Section 4.3.1.

4.2 Graph-Based SLAM formulation

Once the robot has acquired the necessary odometry measurements and observations and has performed the data association process, the Graph-Based SLAM algorithm can be applied to get the map and trajectory estimation. In order to do so, the information needs to be represented in a graph called Dynamic Bayesian Network (DBN). Furthermore, Graph-Based SLAM computes predictions of landmark observations and odometry measurements in order to define a cost function which is used to solve the SLAM problem. Therefore, landmark observation and odometry measurement prediction models have to be defined. The following subsections give a detailed description of the DBN construction and the cost function definition.

4.2.1 Dynamic Bayesian Network

As it was explained above, when the car moves around the circuit it acquires landmark observations which are obtained in a certain position of the trajectory. Furthermore, each position of the trajectory is related to an odometry measurement. Keeping this in mind, a graph that contains the positions and observations can be created. The landmarks can be added to the graph, as the landmark observations will be related to them, and the odometry that relates two consecutive positions can be added as well. This graph can be generated using a DBN. Figure 12 shows a representation of a DBN containing positions, landmarks, landmark observations and odometry measurements. Note that Figure 12 contains two different types of nodes: hidden and normal nodes. Normal nodes are those that contain information that can be directly measured, which in the SLAM problem are odometry and landmark observations. On the other hand, hidden nodes can not be measured directly and need to be estimated by the SLAM algorithm. Consequently, the Graph-based SLAM objective is to produce an estimation of the hidden nodes. In order to do so, the DBN can be slightly transformed to remove the normal nodes (odometry and landmark observations) and introduce them as edges relating the hidden nodes. These edges can be interpreted as constraints that relate the hidden nodes. Figure 13 shows a DBN formed by hidden nodes and constraints.

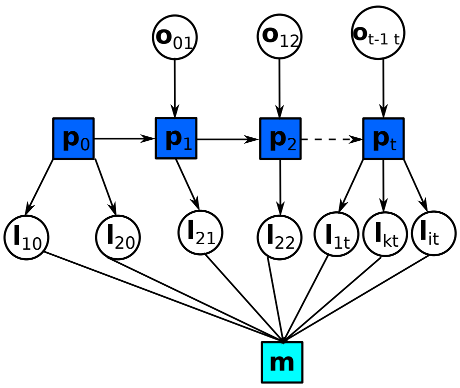

Figure 12: Dynamic Bayesian Network of the SLAM problem. Each position $\mathbf{p}_t$ is related to an odometry measurement $\boldsymbol{o}_{t-1\,t}$ and landmark observations $\boldsymbol{l}_{it}$. Landmark observations are related to landmarks of the map, $\boldsymbol{m}_i$, with the data association process. Hidden nodes are represented as squares whilst observation and measurements are represented as circles. Note that $\boldsymbol{m} = [\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots]^T$.

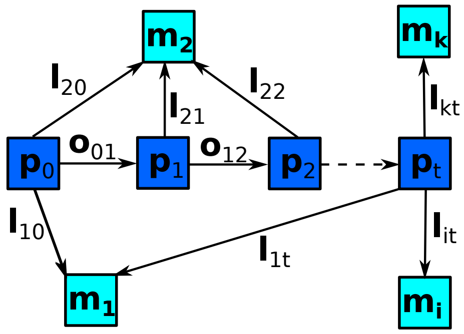

Figure 13: Dynamic Bayesian Network of the SLAM problem with only hidden nodes. The edges denote constraints between the nodes. These constrains contain the odometry and landmark observations information.

In the DBN presented in Figure 13, clear relationships are visualized between the odometry measurements and the positions as well as between the landmark observations and the landmarks, which are represented as edges (constraints). Each of these constraints has an associated conditional probability that relates the landmark observations and odometry measurements to the observations and measurements predicted by the algorithm. These predictions are made following the measurements and observations models detailed in Appendix A.4, which define the odometry measurement error $\mathbf{e}^{\boldsymbol{o}}_{t-1\,t}$ and the landmark observation error $\mathbf{e}^{\boldsymbol{l}}_{it}$.

4.2.2 Cost Function

Following the odometry measurement and landmark observation models detailed in Appendix subsections A.4.1 and A.4.2, Figure 13 can be modified and transformed into Figure 14, where edges represent probabilities.

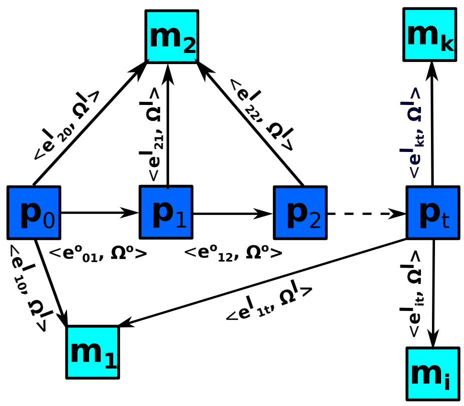

Figure 14: Dynamic Bayesian Network of the SLAM problem with the associated probabilities.

With the DBN of Figure 14, a cost function that contains all the probabilities associated to the edges can be defined as:

$$\mathbf{F}(\boldsymbol{x}_{SLAM}) = \sum_{\forall(i,j)} \mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij} \tag{35}$$

where $\mathbf{e}_{ij}$ includes $\mathbf{e}_{ij}^{\boldsymbol{o}}$ and $\mathbf{e}_{ij}^{\boldsymbol{l}}$, and $\boldsymbol{\Omega}_{ij}$ includes $\boldsymbol{\Omega}^{\boldsymbol{o}}$ and $\boldsymbol{\Omega}^{\boldsymbol{l}}$.

Now, using equation (32), the objective of the Graph-Based SLAM algorithm is to obtain $\boldsymbol{x}_{SLAM}^*$ satisfying:

$$\boldsymbol{x}_{SLAM}^* = \arg\min_{\boldsymbol{x}_{SLAM}} \mathbf{F}(\boldsymbol{x}_{SLAM}) \tag{36}$$

Equation (36) is a non-linear least-squares problem that can be solved using different approaches. In this study, the Gauss-Newton algorithm (see Appendix subsection A.4.3 for a detailed explanation) is used to obtain a solution of the problem.

4.3 Implementation

The Graph-Based SLAM implementation follows a Front-End/Back-End structure. The Front-End collects the odometry measurements, landmark observations and GPS positions, implements the data association and prepares the data for the Back-End. On the other hand, the Back-End applied the Gauss-Newton algorithm explained in the Appendix Section A.4.3. The following subsections detail the design and the algorithms of the Front-End and the Back-End.

4.3.1 Front-End

The Front-End implementation follows the block diagram of Figure 15. As shown in this diagram, the Front-End algorithm receives raw measurements and GPS positions ($\boldsymbol{s} = [x, y, \Theta, v_x, v_y, \omega]^T$). These measurements are transformed into odometry measurements following the model explained in Section 4.1.1 and then the log-likelihood error probability (see equation (57) of the Appendix subsection A.4.1) is obtained. Furthermore, the landmark observations ($\boldsymbol{l}$) are sent to the Data Association algorithm and the log-likelihood error probability (see equation (60) of the Appendix subsection A.4.2) is computed. The outputs of the Front-End are the initial estimated landmarks ($\tilde{\boldsymbol{m}}$), the initial estimated positions ($\tilde{\mathbf{p}}$) and the log-likelihood probabilities associated to the edges of the DBN graph ($\mathbf{e}_{ij}$). The outputs of the algorithm are only sent to the Back-End when a loop closure is detected. In this study, the loop closure is detected when the vehicle is near the initial position.

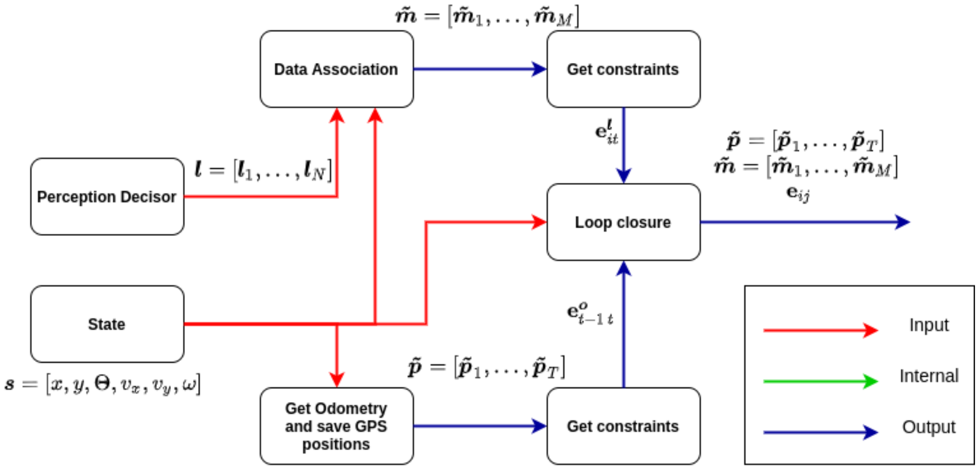

Figure 15: Block diagram of the Graph-Based SLAM Front-End

The algorithm that implements the block diagram of Figure 15 is detailed in Algorithm 4. The odometry prediction error is computed in steps 2 to 5 and then the error is saved into the global error vector $\mathbf{e}$ in step 6. In step 7, the Data Association implemented in Algorithm 3 is called. Next, in steps 8 to 10, the landmark observations predictions errors are calculated and saved into the global error vector as well. Finally, if the algorithm detects the loop closure, the Back-End Algorithm 5 returns the map and trajectory and the information is published.

Algorithm 3 Data Association

function DATA_ASSOCIATION($\mathbf{p}_t, \boldsymbol{l}_t, \boldsymbol{m}$)
  for $\forall i$ **do**
    $\boldsymbol{l}^{global}_{it} \leftarrow \mathbf{p}^{x,y}_t + \boldsymbol{R}_{\mathbf{p}_t[2]} \boldsymbol{l}_t[i]^{x,y}$
    $\boldsymbol{m}_{KD} \leftarrow$ KD_TREE($\boldsymbol{l}^{global}_{it}$, $\boldsymbol{m}$) ▷ Get nearest neighbor with KD_Tree
    $\boldsymbol{d}^{euc} \leftarrow$dist($\boldsymbol{l}^{global}_{it}$, $\boldsymbol{m}_{KD}$) ▷ Calculate euclidean distances
    $d^{euc}_{min} \leftarrow$ min($\boldsymbol{d}^{euc}$) ▷ Get minimum distance

    if $d^{euc}_{min} < \chi^{euc}$ **then** ▷ Compare with threshold
      $\boldsymbol{c}[i] \leftarrow$ arg_min(d^{euc}) ▷ Associate to landmark
    else
      $P \leftarrow$size($\boldsymbol{m}$) ▷ Create new landmark
      $\boldsymbol{m}[P] \leftarrow \boldsymbol{l}^{global}_{it}$
  return $\boldsymbol{m}, \boldsymbol{c}$

Algorithm 4 Graph-Based SLAM Front-End

procedure FRONT-END_CALLBACK$(x_t, y_t, \Theta_t, v_{x_t}, v_{y_t}, \omega_t, \boldsymbol{l}_t, \boldsymbol{m})$

$\quad \boldsymbol{o}_{t\,t+1} \leftarrow \begin{bmatrix} (v_{x_t}\cos\Theta_t - v_{y_t}\sin\Theta_t)\Delta T \\ (v_{x_t}\sin\Theta_t + v_{y_t}\cos\Theta_t)\Delta T \\ \omega_t \Delta T \end{bmatrix}$ ▷ Apply model to get odometry

$\quad \mathbf{p}[t] \leftarrow [x_t, y_t, \Theta_t]^T$ ▷ Save GPS position

$\quad \tilde{\boldsymbol{o}}_{t-1\,t} \leftarrow \mathbf{p}[t] \ominus \mathbf{p}[t-1]$ ▷ Get odometry prediction

$\quad \mathbf{e}^{\boldsymbol{o}}_{t-1\,t} \leftarrow \tilde{\boldsymbol{o}}_{t-1\,t} \ominus \boldsymbol{o}_{t-1\,t}$ ▷ Get odometry prediction error

$\quad \mathbf{e}[t-1, t] \leftarrow \mathbf{e}^{\boldsymbol{o}}_{t-1\,t}$ ▷ Save odometry prediction error

$\quad \boldsymbol{m}, \boldsymbol{c} \leftarrow \text{Data\_Association}(\mathbf{p}[t], \boldsymbol{l}_t, \boldsymbol{m})$ ▷ Associate landmark observations

$\quad$ **for** $\forall i$ **do**

$\quad\quad \tilde{\boldsymbol{l}}_i \leftarrow \boldsymbol{R}^T_{\Theta_t}(\boldsymbol{m}^{x,y}[\boldsymbol{c}[i]] - \mathbf{p}[t]^{x,y})$ ▷ Get observation prediction

$\quad\quad \mathbf{e}^{\boldsymbol{l}}_{it} \leftarrow \tilde{\boldsymbol{l}}_i - \boldsymbol{l}_t[i]$ ▷ Get observation prediction error

$\quad\quad \mathbf{e}[i, t] \leftarrow \mathbf{e}^{\boldsymbol{l}}_{it}$ ▷ Save observation prediction error

$\quad$ **if** $\text{dist}(\mathbf{p}[t], \mathbf{p}[0]) \approx 0$ **then** ▷ Loop closure

$\quad\quad \mathbf{p}, \boldsymbol{m} \leftarrow \text{Back\_End}(\mathbf{e}, \mathbf{p}, \boldsymbol{m})$ ▷ Call Back-End

$\quad\quad \text{ros::Publish}(\boldsymbol{m})$ ▷ Publish map

$\quad\quad \boldsymbol{m} \leftarrow \mathbf{0}$ ▷ Clear graph

$\quad\quad \mathbf{p} \leftarrow \mathbf{0}$

$\quad\quad \mathbf{e} \leftarrow \mathbf{0}$

$\quad$ **else**

$\quad\quad \text{save}(\boldsymbol{m})$

4.3.2 Back-End

The Back-End algorithm is in charge of applying the Gauss-Newton algorithm and obtaining an estimation of the map and trajectory. As explained in Appendix subsection A.4.3, this algorithm must iteratively apply the linearization via the first order Taylor expansion and solve the linear system (66). In order to do so, this study uses the *g2o* framework [15] to solve the linear system. This framework allows the introduction of the analytical Jacobians (64) and (65) and the solution of the linear system (66). It uses a Cholesky decomposition, which increases the efficiency of the algorithm. Furthermore, *g2o* automatically generates matrix $\mathbf{H}$ (63) and vector $\mathbf{b}$ (62). The necessary parameters for the g2o framework are the error prediction vectors $\mathbf{e}$, the initial solution $\tilde{\boldsymbol{x}}_{SLAM}$ computed in the Front-End, and the information matrices $\boldsymbol{\Omega}^{\boldsymbol{l}}$ and $\boldsymbol{\Omega}^{\boldsymbol{o}}$. Figure 16 shows the block diagram of the Back-End design where the *g2o* is the central block of the diagram.

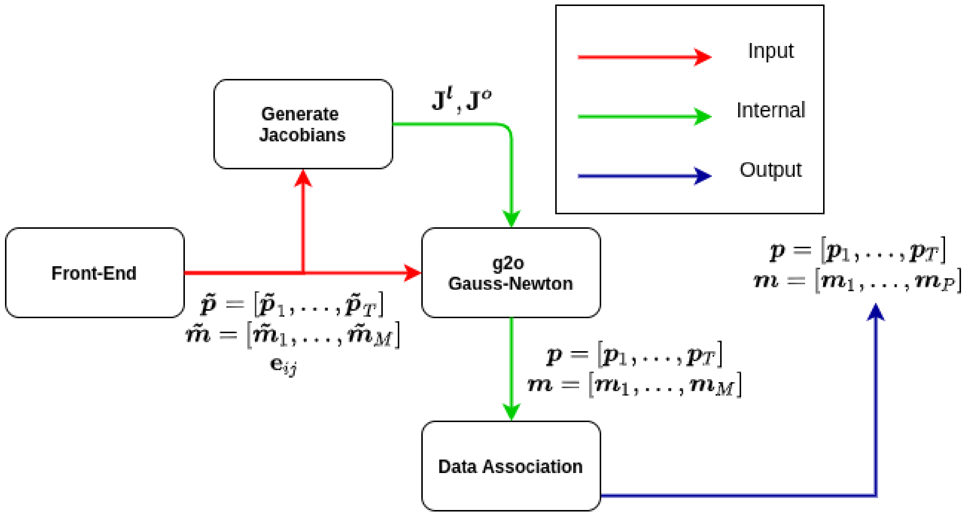

Figure 16: Block diagram of the Graph-Based SLAM Back-End

Algorithm 5 implements the block diagram of Figure 16. The Jacobians of the predicted error functions are defined and are passed to the solver, as well as the initial solution and edges (steps 2 to 6). Then, the *g2o* solver implements an efficient version of the Gauss-Newton Algorithm 7. Note that the information matrices ($\boldsymbol{\Omega}^o$ and $\boldsymbol{\Omega}^l$), which are constant matrices empirically obtained, are directly passed to the solver. Finally, a second data association is done to merge landmarks that represent the same cone and to remove outliers. This second data association is implemented in Algorithm 6. Note that it has a similar structure compared to the normal Data Association Algorithm 3.

Algorithm 5 Graph-Based SLAM Back-End

1: **function** BACK_END($\mathbf{e}, \tilde{\mathbf{p}}, \tilde{\boldsymbol{m}}$)

2: $\quad \tilde{\boldsymbol{x}}_{\boldsymbol{SLAM}} \leftarrow \tilde{\mathbf{p}}, \tilde{\boldsymbol{m}}$ ▷ Set initial solution

3: $\quad \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij] = \dfrac{\partial \mathbf{e}^{\boldsymbol{o}}_{ij}}{\partial \mathbf{p}_i}$ ▷ Define Jacobians of the prediction error functions

4: $\quad \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij] = \dfrac{\partial \mathbf{e}^{\boldsymbol{o}}_{ij}}{\partial \mathbf{p}_j}$

5: $\quad \mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij] = \dfrac{\partial \mathbf{e}^{\boldsymbol{l}}_{ij}}{\partial \boldsymbol{m}_i}$

6: $\quad \mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij] = \dfrac{\partial \mathbf{e}^{\boldsymbol{l}}_{ij}}{\partial \mathbf{p}_j}$

7: $\quad \mathbf{p}, \boldsymbol{m} \leftarrow$ g2o_solve($\tilde{\boldsymbol{x}}_{\boldsymbol{SLAM}}$, $\mathbf{e}$, $\mathbf{J}^o_i[ij]$, $\mathbf{J}^o_j[ij]$, $\mathbf{J}^l_i[ij]$, $\mathbf{J}^l_j[ij]$, $\boldsymbol{\Omega}^o$, $\boldsymbol{\Omega}^l$) ▷ Call the solver

8: $\quad$ **for** $\forall i$ **do**

9: $\quad\quad \boldsymbol{m} \leftarrow$ Second_Data_Association($\boldsymbol{m}[i], \boldsymbol{m}$) ▷ Associate landmarks that represent the same cone

$\quad$ **return** $\mathbf{p}, \boldsymbol{m}$

Algorithm 6 Second_Data_Association

function SECOND_DATA_ASSOCIATION($\boldsymbol{m}_i, \boldsymbol{m}$)
    $\boldsymbol{m}_{KD} \leftarrow$ KD_TREE($\boldsymbol{m}_i$, $\boldsymbol{m}$) ▷ Get nearest neighbor with KD_Tree
    $\boldsymbol{d}^{euc} \leftarrow$dist($\boldsymbol{m}_i$, $\boldsymbol{m}_{KD}$) ▷ Calculate euclidean distances
    $d^{euc}_{min} \leftarrow \min(\boldsymbol{d}^{euc})$ ▷ Get minimum distance

    if $d^{euc}_{min} < \chi^{euc}$ **then** ▷ Compare with threshold
        $\boldsymbol{m}_j \leftarrow$ arg_min(d^{euc}) ▷ Get landmark with min distance
        g2o_merge($\boldsymbol{m}_i, \boldsymbol{m}_j$) ▷ Merge landmarks
    return $\boldsymbol{m}$

4.3.3 ROS Implementation

Once the Front-End and Back-End algorithms are defined, the ROS structure can be added to the Graph-Based SLAM algorithm. As in the MPCs of Section 3, the ROS structure is added following a Callback and Spinning architecture implemented in C++. Figure 17 represents the ROS graph diagram of the Graph-Based SLAM algorithm. As shown in this figure, the algorithm reads the state topic with the odometry and GPS information (`/state/car`) and the landmark observations (`/cones/positions` and `/cones/corrections`). When the Back-End algorithm is called and the Graph-Based SLAM problem is solved, the message containing the map of the circuit (`/cones/graph_sl-am/global`) is published.

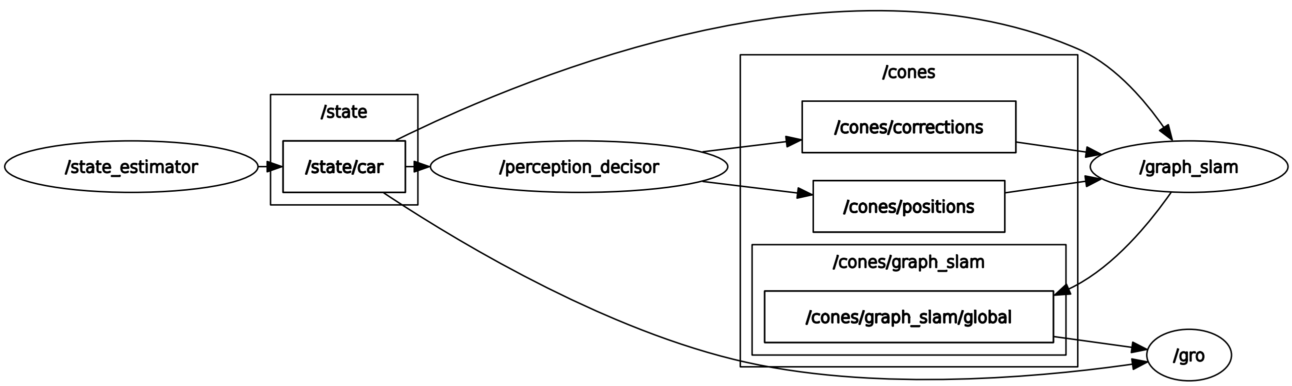

Figure 17: Graph-Based SLAM ROS graph. ROS topics (messages) are represented with boxes. ROS nodes (programs)are represented with ellipses. Arrows denote a relationship between a topic end its publishers and subscribers.

5 Testing and Results

This section aims to present the results of the Model Predictive Control algorithms described in Section 3 and the Graph-based Simultaneous Localizzation and Mapping algorithm defined in Section 4. Two different scenarios were used for testing purposes: simulation and vehicle testing. Furthermore, computational time is analysed as well.

5.1 Nonlinear Model Predictive Control

5.1.1 Simulation

In order to test the system and tune the NMPC parameters, a simulation capable of taking into account the vehicle dynamics is necessary before trying the system in our vehicle. The chosen work-space to test the NMPC was Formula Student Simulator (FSSIM developed by AMZ [2]), a Formula Student Driverless simulator developed with the purpose of working with ROS.

FSSIM uses Gazebo [16] to simulate the dynamics of the car, which are modeled using a mixed dynamic and kinematic bicycle model. The following videos show two of the simulations performed in this study using FSSIM. They correspond to two different tracks, the one used in Formula Student Germany 2019 (FSG 2018), and the one used in Formula SAE Italy 2019 (FSI 2019).

- Formula Student Germany 2019
- Formula SAE Italy 2019

The parameters used in the above simulations are stated in Table 3. It is clearly visible in the videos that with this set of parameters the NMPC performs well in simulation, accomplishing small lateral and longitudinal positioning errors and small longitudinal velocity error. A plot with the following longitudinal velocities: target, NMPC solution and current is shown in Figure 18. Figure 19 displays the lateral velocity computed by the NMPC along with the current one. Note that differences between the NMPC and the simulation vehicle dynamics models are clearly visible in this figure, because of the high sensibility of the lateral velocity towards variations in the vehicle model.

The performance of the NMPC improves when the GRO computes the global trajectory, which is more stable and accurate than the trajectory computed by the planner. This is why velocities achieved during the second lap are much higher than those of the first lap. This behavior is clearly illustrated in Figures 20, 21, 22 and 23, where heat maps of the second lap velocities are represented. Note that the longitudinal velocities reach values as high as 23 m/s (83 km/h) in the simulations.

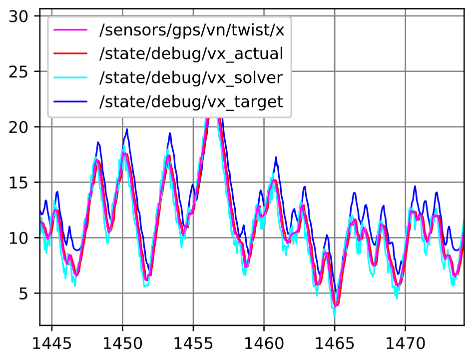

Figure 18: Longitudinal velocity: current (pink and red), solution (cyan) and target (blue).

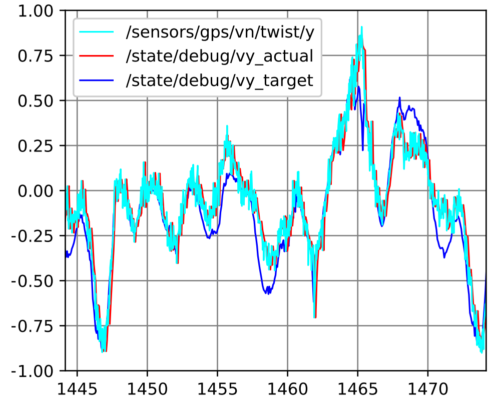

Figure 19: Lateral velocity: current (red and cyan) and solution (blue).

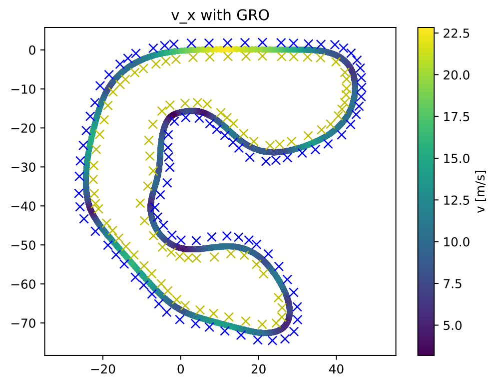

Figure 20: Longitudinal velocity (FSG 2019 track).

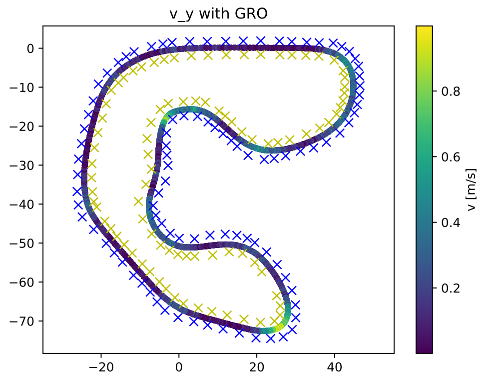

Figure 21: Lateral velocity (FSG 2019 track).

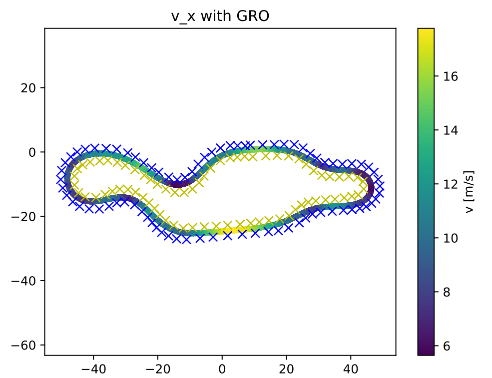

Figure 22: Longitudinal velocity (FSI 2019 track).

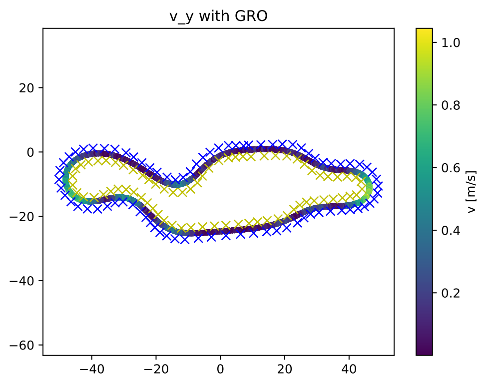

Figure 23: Lateral velocity (FSI 2019 track).

Table 3: NMPC Parameters used in the simulations.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
q	0.5	ϵ_δ	$2°$	$v_{y_{min}}$	$-3m/s$	C_F	-1.1705
r	0.5	ϵ_a	$2m/s^2$	$v_{y_{max}}$	$3m/s$	B_F	10.8529
Q_{lat}	0.45	δ_{max}	$23°$	ω_{min}	$-6\pi rad/s$	D_R	2.5007
Q_{lon}	0.45	a_{acc_max}	$10m/s^2$	ω_{max}	$6\pi rad/s$	C_R	-1.1705
Q_{v_x}	0.1	a_{dec_max}	$-10m/s^2$	m	$240kg$	B_R	10.1507
ϵ_{lat}	$0.2m$	$\Delta\delta_{max}$	$2°$	g	$9.81m/s^2$	μ_r	0.061
ϵ_{lon}	$0.5m$	Δa_{dec_max}	$-2m/s^2$	I	$100kg \cdot m^2$	C_d	0
ϵ_{v_x}	$1m/s$	Δa_{acc_max}	$2m/s^2$	L_R	$0.822m$	ρ	$1.225kg/m^3$
R_δ	0.9	$v_{x_{min}}$	$2m/s^2$	L_F	$0.708m$	A_r	$0m^2$
R_a	0.1	$v_{x_{max}}$	$30m/s$	D_F	2.5465		

5.1.2 Testing

After a long period of simulation, the NMPC was finally integrated into XALOC's software stack and was tested altogether with all the other algorithms integrated in XALOC's software. The testing circuits has the characteristics of a Trackdrive circuit to simulate a competition scenario and the tracks were about 200 m long. Tests were performed in two different scenarios: a first lap in which the XALOC was recognizing the environment and as a consequence the planner was not stable, and the rest of laps with an optimized trajectory (GRO).

Results were outstanding as XALOC completed the circuit with an average lap time of 24 seconds, an average longitudinal velocity of 11 m/s (40 km/h) and a peak longitudinal velocity of 15 m/s (54 km/h). The following video shows the performance of XALOC in the described testing circuit: XALOC (CAT13d) - BCN eMotorsport Driverless. The NMPC parameters used to achieve the performance shown in the video are summarized in Tables 4 and 5 (note that we use two sets of parameters, one for the first lap and one for the rest of laps with the GRO).

In the video, it can be seen that the NMPC tries to follow the desired trajectory computed by the GRO taking into account the current state of the car and its dynamics. Figures 24 and 25 show velocity heat maps using the planner obtained with the GRO. Figure 26 shows the target, the NMPC solution and the current longitudinal velocity. Figures 27 and 28 show the current and the NMPC solution lateral velocity and yaw rate. Finally, Figure 29 shows the steering angle computed by the NMPC and the current steering angle. From these figures, it can be concluded that the steering angle commands are smooth and generate the desired yaw rate. Furthermore, the NMPC follows the target longitudinal velocity with an acceptable error. This error is a consequence of the fact that the NMPC gives a higher priority to follow the desired trajectory than to follow the the target velocity. That is, if the NMPC decides that the car cannot follow the desired trajectory with the desired velocity, the desired trajectory has a higher priority.

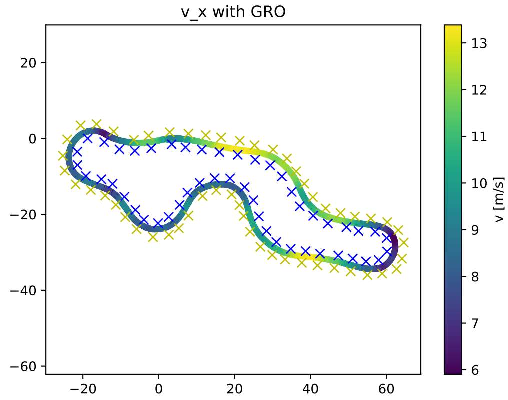

Figure 24: Longitudinal velocity in real testing.

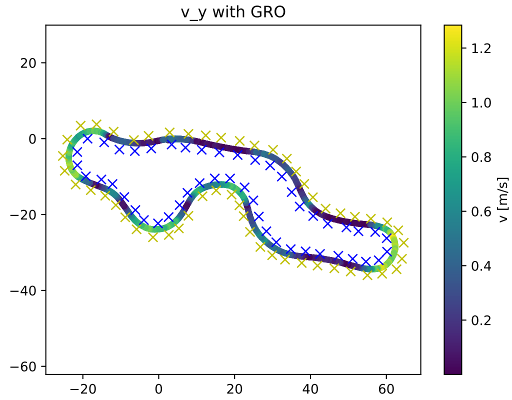

Figure 25: Lateral velocity in real testing.

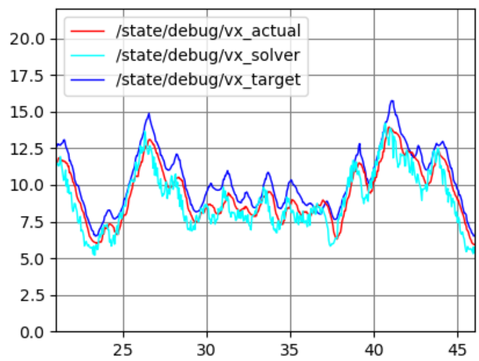

Figure 26: Longitudinal velocity in real testing: current (red), solution (cyan) and target (blue).

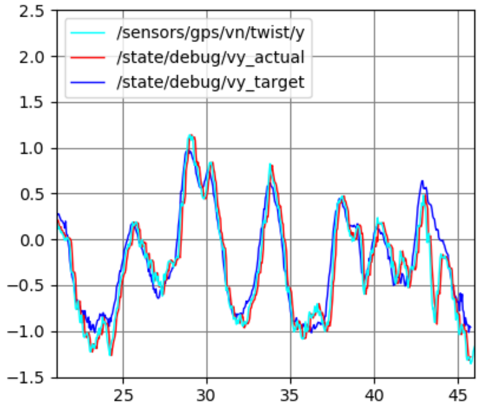

Figure 27: Lateral velocity in real testing: current (red and cyan) and solution (blue).

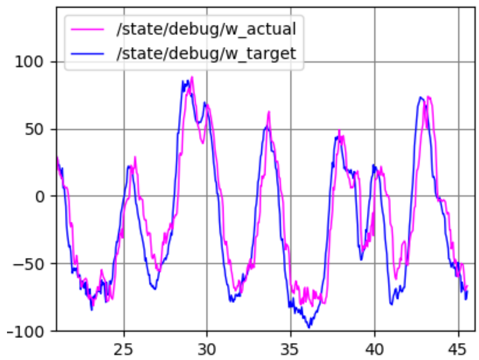

Figure 28: Yaw rate in real testing: current (pink), solution (blue).

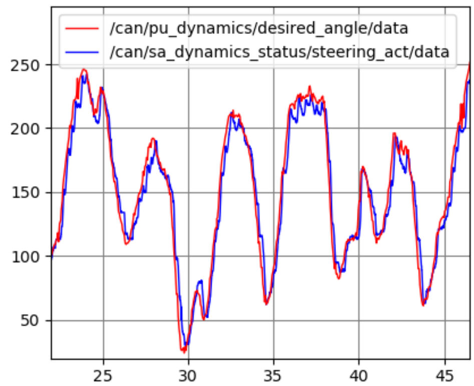

Figure 29: Desired (red) and current (blue) steering actuator position in testing.

Table 4: NMPC parameters for XALOC: first lap.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
q	0.5	ϵ_δ	$0.375°$	$v_{y_{min}}$	$-3m/s$	C_F	-1.1705
r	0.5	ϵ_a	$1m/s^2$	$v_{y_{max}}$	$3m/s$	B_F	15.5585
Q_{lat}	0.5	δ_{max}	$23°$	ω_{min}	$-6\pi rad/s$	D_R	0.5125
Q_{lon}	0.45	a_{acc_max}	$6m/s^2$	ω_{max}	$6\pi rad/s$	C_R	-1.1705
Q_{v_x}	0.05	a_{dec_max}	$-10m/s^2$	m	$240kg$	B_R	15.5585
ϵ_{lat}	$0.2m$	$\Delta\delta_{max}$	$0.375°$	g	$9.81m/s^2$	μ_r	0.061
ϵ_{lon}	$0.5m$	Δa_{dec_max}	$-1m/s^2$	I	$93kg \cdot m^2$	C_d	0
ϵ_{v_x}	$1m/s$	Δa_{acc_max}	$1m/s^2$	L_R	$0.822m$	ρ	$1.225kg/m^3$
R_δ	0.9	$v_{x_{min}}$	$0m/s^2$	L_F	$0.708m$	A_r	$0m^2$
R_a	0.1	$v_{x_{max}}$	$40m/s$	D_F	0.4415		

Table 5: NMPC parameters for XALOC: GRO laps.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
q	0.5	ϵ_δ	$0.375°$	$v_{y_{min}}$	$-3m/s$	C_F	-1.1705
r	0.5	ϵ_a	$1m/s^2$	$v_{y_{max}}$	$3m/s$	B_F	10.8529
Q_{lat}	0.5	δ_{max}	$23°$	ω_{min}	$-6\pi rad/s$	D_R	2.8934
Q_{lon}	0.45	a_{acc_max}	$6.5m/s^2$	ω_{max}	$6\pi rad/s$	C_R	-1.1705
Q_{v_x}	0.05	a_{dec_max}	$-10m/s^2$	m	$240kg$	B_R	10.1507
ϵ_{lat}	$0.2m$	$\Delta\delta_{max}$	$5°$	g	$9.81m/s^2$	μ_r	0.061
ϵ_{lon}	$0.5m$	Δa_{dec_max}	$-2.5m/s^2$	I	$93kg \cdot m^2$	C_d	0
ϵ_{v_x}	$1m/s$	Δa_{acc_max}	$2.5m/s^2$	L_R	$0.822m$	ρ	$1.225kg/m^3$
R_δ	0.9	$v_{x_{min}}$	$0m/s^2$	L_F	$0.708m$	A_r	$0m^2$
R_a	0.1	$v_{x_{max}}$	$40m/s$	D_F	2.2020		

5.2 LPV-MPC

The LPV-MPC algorithm was tested within the same simulation framework as the NMPC, that is FSSIM. To compare the LPV-MPC and NMPC performance, we choose the Formula Student Germany 2019 track. The LPV-MPC performance during the first lap was quite extraordinary, as the car followed both the velocity profile and the desired trajectory with small errors. Performance in the rest of laps with the GRO were satisfactory as the car perfectly followed the target velocity. Hoewever, a considerable amount of steady state lateral position error was present. These behaviors can be visualized in the following video: LPV-MPC FSG 2018.

Figures 30 and 31 represent the longitudinal and lateral velocities in the simulated circuit. It can bee seen in figure 30 that the longitudinal velocity is almost perfectly followed. Furthermore, the difference between the LPV-MPC and the simulation vehicle models is clearly visible in the lateral velocities of Figure 31. Finally, the LPV-MPC parameters used in this simulation are listed in Table 6 .

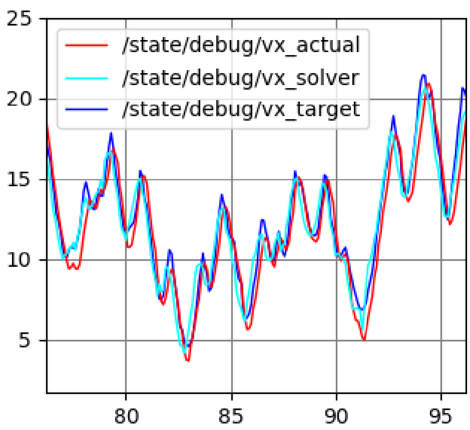

Figure 30: Longitudinal velocity in simulation: target (blue), LPV-MPC solution (cyan) and current (red).

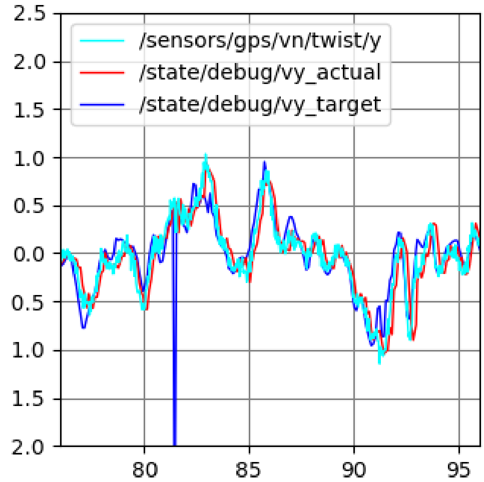

Figure 31: Lateral velocity in simulation: LPV-MPC solution (blue) and current (cyan and red).

Table 6: LPV-MPC parameters used in simulation.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
q	0.5	ϵ_δ	$5°$	$v_{y_{min}}$	$-3m/s$	C_F	$-27055N$
r	0.5	ϵ_a	$5m/s^2$	$v_{y_{max}}$	$3m/s$	C_R	$-24752N$
Q_{lat}	0.5	δ_{max}	$23°$	ω_{min}	$-6\pi rad/s$	C_d	0
Q_{lon}	0.0	a_{acc_max}	$15m/s^2$	ω_{max}	$6\pi rad/s$	ρ	$1.225kg/m^3$
Q_{v_x}	0.5	a_{dec_max}	$-10m/s^2$	m	$240kg$	A_r	$0m^2$
ϵ_{lat}	$0.2m$	$\Delta\delta_{max}$	$5°$	g	$9.81m/s^2$		
ϵ_{lon}	$0.5m$	Δa_{dec_max}	$-5m/s^2$	I	$93kg \cdot m^2$		
ϵ_{v_x}	$1m/s$	Δa_{acc_max}	$5m/s^2$	L_R	$0.822m$		
R_δ	0.9	$v_{x_{min}}$	$0m/s^2$	L_F	$0.708m$		
R_a	0.1	$v_{x_{max}}$	$40m/s$	μ_r	0.061		

5.3 MPC computational time

Computational time is one of the most important elements of our MPC algorithm, as our system works in real time. The NMPC solving time is, on average, 22ms and rarely exceeds the Control period time $Tc = 50ms$. These data shows that this NMPC is well fitted to solve the Formula Student Control problem regarding computational time. On the other hand, the LPV-MPC average computational time is 19ms. Even though the average computational time of the LPV-MPC algorithm is smaller than the NMPC average computational time, the difference of average computational times is of only 3ms. Therefore, this difference in computational time does not compensate the loss of accuracy in the vehicle dynamics model that exists when using the LPV-MPC.

A Whisker plot of the computational times of both algorithms (NMPC and LPV-MPC is shown in Figure 32. These results show that the computational time difference between both algorithms is not as big as previously expected. In order to compute the average time, the MPC algorithms were executed in a Cincoze DX-1000 (XALOC's processing unit).

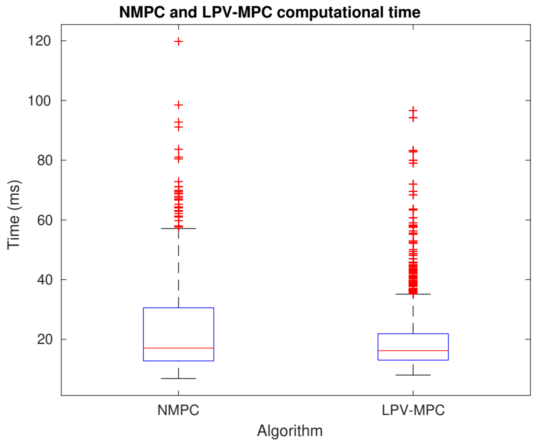

Figure 32: Timing results of LPV-MPC and NMPC.

5.4 Graph-based SLAM

5.4.1 Testing

The Graph-based SLAM algorithm was tested using the data collected with XALOC during testing days. This system does not require real-time testing and, therefore, it can be tested with saved data. The main goal of the test was to check the feasibility of the algorithm in mapping the cones of the circuit while disregarding the outliers. In order to do so, the algorithm was executed and several plots, detailed in what follows, were produced

Figure 33 shows the results of the algorithm tested with real data. The bottom plot shows all the landmark observations with respect to the car current position, whilst the top plot shows the output of the Graph-Based SLAM algorithm. As it can be seen in the top plot, the majority of outliers have been removed (only one outlier remains) and the landmarks have been correctly mapped.

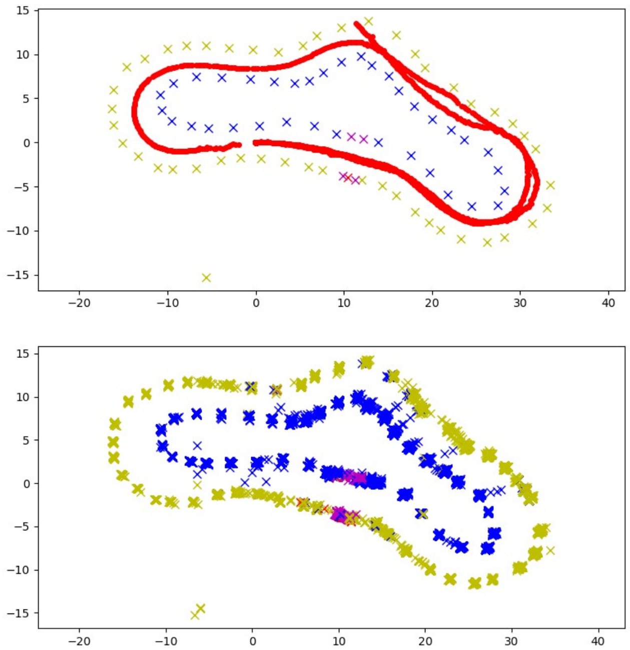

Figure 33: Top plot shows the Graph-Based SLAM mapped landmarks (yellow, blue, red and purple crosses) and the corrected trajectory (red dots). Bottom plot shows the raw landmark observations.

5.4.2 Computational time

The computational time of the Graph-Based SLAM algorithm is one of its major flaws. Its high computational time makes it impossible for Graph-Based SLAM to be executed in real-time. However, as Graph-Based SLAM can be executed offline between the Autocross and Trackdrive events (taking into account that the same circuit is used) the computational time is no longer an important characteristic.

The computational time increases with the number of landmark observations. However, for circuits similar to a Trackdrive, its computational time always remains below 1000ms and its average time is 600ms.

6 Budget

The development of this project requires both hardware and software products. The hardware needed to develop this project comprises a LiDAR, two cameras, two GPS, a processing unit, a Graphic Processing Unit (GPU) and the autonomous actuators. Materials and products needed to build the monocoque, dynamics, aerodynamics, electronics and powertrain are not included in this budget. Regarding the software, open source distributions were mainly used except for the FORCESPRO solver. Finally, the engineering students have dedicated 44 weeks with a workload of 25 hours per week, which is a total of 1100 hours. The degree tutors have dedicated 2 hours every two weeks during a 22 week period, which is a total of 22 hours. The wages for junior engineers and degree thesis tutors are 10€/h and 30€/h, respectively.

The budget of this project is detailed in the following table (Table 7). Note that the products that have a final cost of 0,00€ have been obtained via sponsorships.

Table 7: Budget.

Product	Q	Cost per unit	Total Cost	Final Cost
Velodyne VLP-32C	1	13000,00€	13000,00€	0,00€
The Imaging Source DFK DFK33UX252 cameras + lens + USB3.0 cable	2	1200,00€	2400,00€	0,00€
Vectornav VN300	1	4000,00€	4000,00€	0,00€
SBG Ellipse-N	1	4000,00€	4000,00€	0,00€
Cincoze DX-1000	1	5000,00€	5000,00€	3350,00€
NVIDIA Jetson Xavier	1	590,00€	590,00€	0,00€
D-Link Ethernet Switch	1	46,31€	46,31€	0,00€
FORCESPRO License	3	14626,65€	43879,95€	0,00€
Service Brake and EBS	1	3100,00€	3100,00€	0,00€
Steering Actuator	1	1500,00€	1500,00€	500,00€
Junior Engineer (10€/h)	1100	11000,00€	11000,00€	11000,00€
Degree thesis tutor (30€/h)	22 × 2	1320,00€	1320,00€	1320,00€
TOTAL			**89836,26€**	**16170,00€**

7 Conclusions and Future Work

7.1 Conclusions

In this thesis, an algorithm to solve the localization and mapping issue, and two algorithms to solve the control problem of a Formula Student Driverless car have been proposed, detailed and implemented as part of the Estimation and Control architectures of an autonomous car. The main goal of these algorithms, the Graph-Based Simultaneous and Mapping and the Model Predictive Control, is to improve the competitiveness of the BCN eMotorsport driverless team by leading the car to reach high speeds and by reducing lap time. These improvements are crucial to achieve successful results in the 2021 Formula Student Driverless competitions.

Being a member of a Formula Student Driverless team requires more than just designing and developing algorithms to improve the performance of the car. Many hardware and software problems occur during the season and these must be solved even if these problemsdo not constitute the main tasks that were assigned to the members of the team. Furthermore, in order to attend the competitions the team must prepare and hand in several documents, as well as prepare the static events. These tasks require time and dedication from all members of the team, as these documents are important to achieve favorable results in the competitions.

Regarding the algorithms developed in this thesis, they were implemented to complete Autocross, Trackdrive and Skidpad events at high speeds and with reliability. Reliability is one of the most important characteristics of an autonomous system pipeline for a Formula Student car, as going out of course in a dynamic event or knocking down cones of the circuit penalizes the score of the team in the event. For this reason, the main objective of the SLAM algorithm is to map the cones as accurately as possible to ensure that calculated planned trajectory will not make the car go out of course or knock cones. As for the MPC, its main objective is to follow the planned trajectory with the smallest possible error and, with less priority, to follow the planned velocities.

In terms of the NMPC algorithm, real-time computation was needed to calculate the optimal controls. This restriction made low computational time an important characteristic. Without the use of a fast solver, the implementation of a non-linear MPC would not have been possible. Furthermore, the tuning of the parameters used in the NMPC was one of the most important and demanding tasks. The use of a robust simulation framework and many days of real-time testing with XALOC were of great importance to obtain a suitable set of parameters. Finally, the results proved that the NMPC developed in this study yielded a great performance, as it incremented the average longitudinal velocity by more than 5 times when compared to the 2018-2019 control and it still has potential to increment the velocity even more.

Regarding the LPV-MPC, it was a great challenge to achieve an implementation that was successful in the simulation scenario. The correct use of varying parameters as well as the constant reformulations needed to adapt the problem to a FORCESPRO solver complicated the task of implementing this algorithm. Furthermore, as the main goal was to implement a reliable MPC to achieve high speeds and the NMPC was capable of doing so in real time, the majority of efforts were invested into accomplishing a fine tuning of the NMPC parameters instead of improving the performance of the LPV-MPC. However, the results of the LPV-MPC in simulation proved that the algorithm will be capable of working and reaching a successful result when integrated to XALOC's software in a near future.

As for the Graph-Based SLAM algorithm, results were satisfactory as it correctly mapped all the cones of the circuits and eliminated the majority of outliers. Furthermore, the landmarks position error remained below the required threshold. In addition, in order to reduce the dependence of GPS localization, the algorithm can be executed with the use of odometry measurements and still produce an acceptable result even though not as accurate as the one obtained using the GPS.

To conclude, having the opportunity of being a member of the BCN eMotorsport for the last two years and developing my final thesis in the team, helping it achieve its main objectives, has been one of the most gratifying experiences of my years as a student. I have been able to use and apply skills learnt during my Bachelor's degree, such as signal processing techniques, predictive filtering methods and even electronic concepts, to develop the algorithms detailed in this thesis and to help the team to achieve its goals. Finally, I have improved my team working and leading skills during these last two years in the team, as well as my knowledge in the field of robotics, automatic control and vehicle dynamics.

7.2 Future Work

This thesis comprises three different algorithms that have proven to perform well and to be useful to achieve the desired objectives. However, several improvements can be introduce to enhance the performance of these algorithms.

Regarding the NMPC, the parameters can be further tuned to achieve a better race performance. Furthermore, the aerodynamic coefficients have not yet been included into the algorithm, as all the parameters related to drag force and down force are set to zero. The coefficients are going to be empirically acquired once the front wing and rear wing are assembled into the car before the competitions. These improvements aim to achieve an average longitudinal velocity of 15 m/s before the start of the 2021 competitions.

In terms of the LPV-MPC, some improvements can be included to reduce the steady state lateral error when using the GRO. This task can be done by tuning the parameters and the weights of the cost function. Finally, the algorithm could be tested with XALOC to check its performance. These tasks could be performed during the following season, as improving the LPV-MPC can be beneficial to increase the working frequency of the control pipeline and, therefore, achieve more successful results.

Finally, regarding the Graph-Based SLAM, this algorithm should be compared to the performance of FastSLAM 2.0 (which has been also implemented this season by another member of the team) to decide which algorithm performs better. The best algorithm will be the one used in the competitions. Furthermore, a mixture of Graph-Based SLAM and FastSLAM 2.0 could be also implemented, as FastSLAM 2.0 can be executed in real-time but it does not remove as many outliers as Graph-Based SLAM. Therefore, Graph-Based SLAM could be executed periodically and its output landmarks could serve as an accurate initial solution for the FastSLAM 2.0 algorithm.

With that being said, the majority of these improvements are going to be introduced this very same year during the summer before the start of the competitions. Only the LPV-MPC improvements will remain as a possible *TODO* for the following season.

A Appendix

A.1 Estimation and Control Appendix

This section of the Appendix presents the algorithms incorporated into the XALOC 2.0 software stack, which have been developed by different members of the team and serve as the starting point of the two algorithms developed in this thesis. The XALOC 2.0 software stack consists of three main modules: Perception, Estimation and Control. The Perception pipeline uses a Velodyne Laser Imaging Detection and Ranging (LiDAR) and two stereo cameras to identify, by means of sensor fusion, the color of the cones and their positions with respect to the car center of gravity (CoG). The basic functionalities of this sensor fusion were developed by A. Roche [17] and E. Bové [18]. Regarding the Estimation and Control modules, an overall explanation of the algorithms is given in this section.

A.1.1 Estimation

The estimation module comprises state estimation (velocity and position) and map estimation. This module serves as a union point between the Perception and Control systems, as Control uses the estimated map and state of the car to apply the optimal controls.

State Estimation

The Driverless UPC state estimation algorithm was designed by F. Martí [9]. It uses the phonic wheels and Global Navigation Satellite System/Inertial Navigation System (GNSS/INS) data to estimate the velocities using a Kalman Filter. Then, the position is estimated using a Kalman Filter with the estimated velocities (longitudinal, lateral and yaw rate) and the Global Positioning System (GPS) measurements. This system can be improved using the estimation methods proposed by M. Doumiati [19] and S. Srinivasan et al. [20], who use Extended Kalman Filters and a dynamic vehicle model. Actually, the latter estimation methods are currently being implemented by some members of the team.

The main goal of state estimation algorithms is to obtain an accurate estimation of the state vector $\boldsymbol{x} = [x, y, \Theta, v_x, v_y, \omega]^T$ using the following measurements:

- $\boldsymbol{w} = [w_{FR}, w_{FL}, w_{RR}, w_{RL}]^T$, where w denotes the linear velocity of a wheel. The subscripts FR, FL, RR and RL represent front right, front left, rear right and rear left, respectively.
- $\boldsymbol{a} = [a_x, a_y]^T$, where a_x and a_y represent longitudinal and lateral accelerations, respectively.
- ω, yaw rate.
- $\boldsymbol{v} = [v_x, v_y]$, where v_x and v_y denote GPS measured longitudinal and lateral velocities, respectively.
- $\boldsymbol{p} = [x, y, \Theta]$, where x, y and Θ denote the GPS measured position and heading of the car in a given frame, see Figure 34.

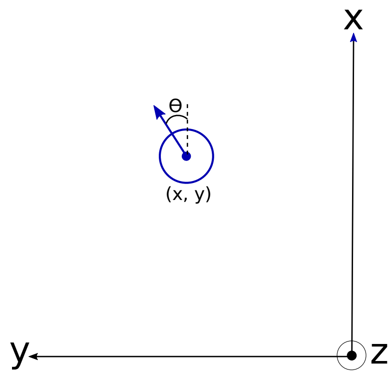

Figure 34: Global frame of coordinates with the car (in blue) in a certain position and with a certain heading Θ.

Mapping

Having the knowledge of the position of the cones is a key advantage to increment the car velocity and to reduce lap time. This is why mapping techniques are necessary for Formula Student Driverless cars. Furthermore, cones can be used to localize our car inside the track. The problem of localizing the car whilst mapping the environment in which the car is placed is known as Simultaneous Localization and Mapping (SLAM). A visual representation of the SLAM problem is shown in Figure 35.

The SLAM problem has been studied by several authors. S. Thrun, in his well known book *Probabilistic Robotics* [21], proposed an algorithm based on an Extended Kalman Filter (EKF) to solve the SLAM problem. M. Montmerlo et. al [22] developed a fast particle filter algorithm which proved to be robust. S. Thrun et. al [12] presented a graph-based method, in which cones and positions form a graph in order to solve the problem. These three algorithms have proved to be efficient to solve the SLAM problem for Formula Student Driverless cars.

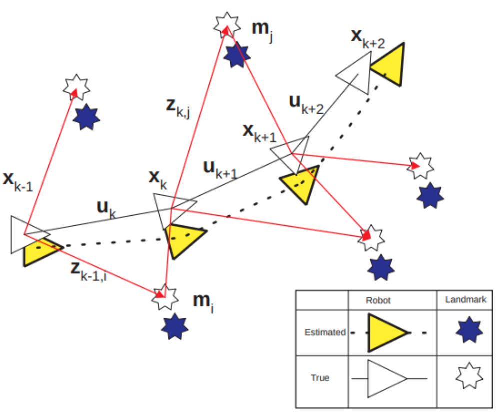

Figure 35: Visual representation of why SLAM is necessary to estimate the landmarks and positions, as these can never be measured directly. Picture from H. Durrant-Whyte and T. Bailey et. al [23].

Currently, our team BCN eMotorsport DV possesses three algorithms namely EKF-SLAM, FastSLAM 2.0 and GraphSLAM to solve the SLAM problem. A detailed explanation of how the Graph-based SLAM algorithm can be implemented for a Formula Student Driverless car can be found in Section 4.

A.1.2 Control

Four algorithms are included in the control architecture: TrackLimits, Path & Planning, Global Racetrack Optimizer and Model Predictive Control. These algorithms figure out the desired positions and velocities and determine the actions that the car should apply, taking into count the vehicle dynamics.

TrackLimits

SLAM provides an accurate estimation of both position and color of cones. However, outliers and wrong color detections can remain present for a certain amount of time before SLAM corrects them. This is why a robust algorithm is needed to find the track boundaries given the current map and the car position.

A Voronoi Diagram is computed with the given cones to create a graph. The main idea of the algorithm is to give weights to each possible combination of edges, without repetition. The weights depend on the distance between the cones and on the colors of the involved cones. The combination of edges with the highest score is chosen and the cones associated to these edges form the track limits. Figure 36 is a visual representation of how this algorithm works.

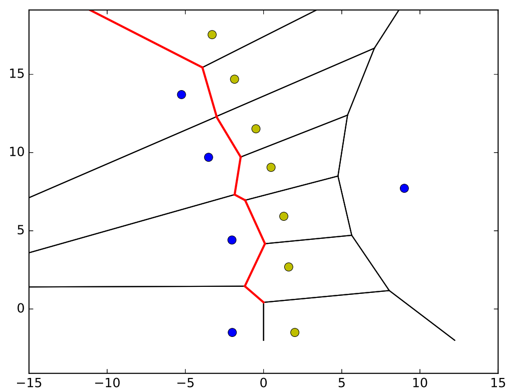

Figure 36: TrackLimits algorithm results. The Voronoi Diagram is computed using the positions of the shown cones. The selected path is shown in red. The cones associated to the red edges form the track boundaries.

Path & Planning

Path & Planning algorithm consists of two parts. In the first part, the desired path that goes through the certer line of the gicen track limits is calculated by means of Spline approximation. In the second part, a velocity profile for the points of the trajectory is computed. This latter part consists of computing the desired longitudinal velocity at each point of the Spline. This algorithm was developed by I. Mallén and is directly related to his work [24]. The output of the Path Planning algorithm feeds the controller during the first lap.

Global Race Optimizer

An advantage of knowing the entire map of the circuit is that an optimal trajectory can be computed and the Track Limits estimation and Path & Planning algorithms can be disregarded. A curvature minimization approach was implemented by I. Mallén [24] to achieve this optimal trajectory and its associated velocity profile. Once the car has completed an entire lap, the Global Race Optimizer (GRO) is activated using the cones estimated by Graph-based SLAM or FastSLAM 2.0. Finally, the controller is fed with the output of the GRO.

Model Predictive Control

Having used a simple controller in the 2018-2019 season and a kinematic bicycle MPC in the 2019-2020 season [9], to design a more accurate controller capable of reaching higher speeds was the objective of this season. The dynamic bicycle MPC was the chosen controller.

The kinematic MPC was able to reach speeds of 10-12 m/s whilst still maintaining the reliability of the model. However, at higher speeds the dynamics of the system become important and the kinematic model comes to be inaccurate. For this reason, the dynamic bicycle MPC, developed in this thesis, was proposed as an alternative to give a step forward, reach higher speeds and become a more competitive team.

The idea behind the dynamic bicycle MPC is to use the dynamic bicycle model to describe the dynamics of our car and predict the future car states and actions of the controllers. The MPC tries to follow the planned states given by the Path Planning or GRO algorithms, taking into account the dynamics of the car. This controller is explained in more detail in Section 3.

A.1.3 Robot Operating System

Our race car, XALOC 2.0, acts as a robot with complex hardware and software systems with a clear main goal: drive autonomously inside a Formula Student circuit. However, developing an entire software stack to perform the perception, estimation and control tasks is highly difficult without a modular system and a set of tools such as synchronization, communication and visualization. For this reason, the full BCN eMotorsport software stack has been implemented and integrated using ROS.

Introduction

ROS [25] is a set of open source libraries that help to develop robot applications. ROS gives a set of tools to synchronize programs and sensors, develop modular systems with easy communication protocols between programs, visualize results and keep track of different reference systems within the robot. It also provides many other tools that ease the task of programming an entire robot application.

ROS implements a communication protocol to allow the implementation of modular systems, where the programs can be seen as black boxes that read information from messages sent by other programs and write their output information into other messages. This structure is implemented with the so called nodes, topics, publishers and subscribers. A node is a program that is subscribed to topics (messages from other nodes) and publishes its outputs in other topics. Figure 37 represents the communication between two nodes using the publisher/subscriber architecture.

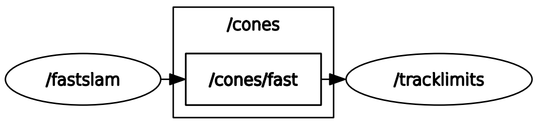

Figure 37: The node *fastslam* publishes the topic */cones/fast* and the node *tracklimits* is subscribed to this topic.

Furthermore, tools such as the following ones:

- *rviz*: graphical interface to visualize topics in real time.
- *rqt_plot*: graphical interface to plot values from ROS messages in real time.
- *rqt_reconfigure*: dynamic tool that allows changing parameter values from different nodes while running the nodes.
- *rostopice echo*: command to print the values of a certain topic.
- *rqt_graph*: visualization tool to see the active nodes and topics.
- *Time Synchronizer*: synchronization tool to start processing a certain amount of topics when these are received at the same or similar time stamp.

allow an easier implementation of software stack. More over, as ROS works with a Transmission Control Protocol/Internet Protocol (TCP/IP) protocol to send and receive topics, a communication with the processing unit of our car is easily achieved via WiFi and, therefore, an online monitoring of the programs is done using ROS.

ROS Integration and Architecture

The estimation and control algorithms have been implemented using a ROS architecture following its communication protocols. Each algorithm is implemented as a ROS node with publishers and subscribers and these programs only interact using ROS. Furthermore, these programs use a Callback and Spinning methodology, in which a node starts its task when the topic in which it is subscribed is published. This means that algorithms do not run at a specified update rate. However, two of these algorithms do work at a specified rate to guarantee the desired behavior of the estimation and control pipelines. These two nodes are the State Estimator (50Hz) and the Model Predictive Control (20Hz).

The block diagram shown in Figure 38 represents the integration of ROS with the estimation and control algorithms. CAN Encoder and Decoder nodes are introduced to communicate the processing unit with the rest of ECUs, as well as a State Machine node that monitors all estimation and control algorithms.

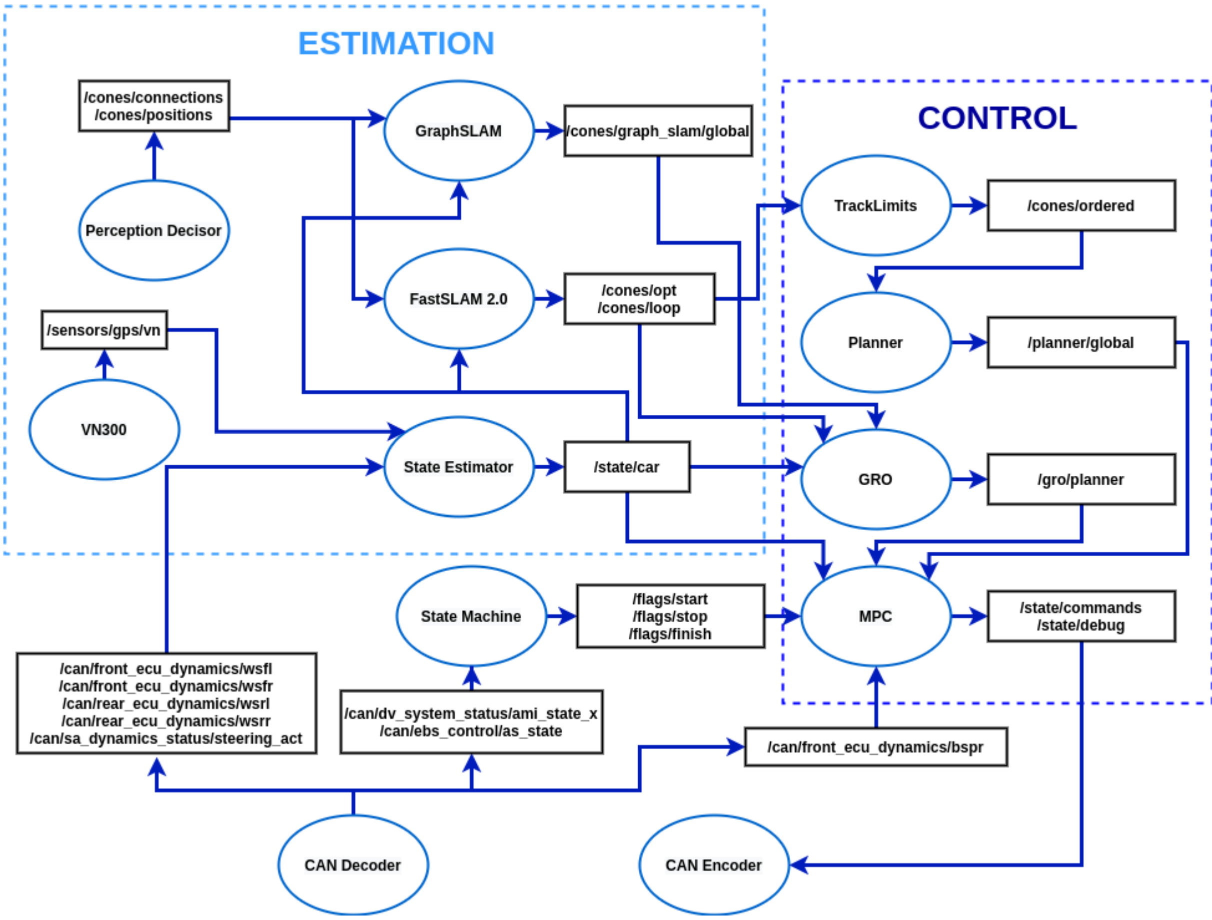

Figure 38: Estimation and Control ROS architecture. Ellipses represent ROS nodes, rectangles represent ROS topics and arrows relate the topics to their publishers and subscribers.

A.2 Nonlinear MPC Appendix

A.2.1 Solving with FORCESPRO

In order to obtain a solution of the MPC problem, a fast and reliable solver is absolutely necessary. Many open source and commercial solvers capable of solving these kind of problems exist. However, FORCESPRO was chosen for its capability to generate a tailored MPC solver and its quickness. The FORCESPRO nonlinear programming (NLP) formulation is stated as:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \sum_{k=1}^{N-1} f_k(\boldsymbol{z}_k, \boldsymbol{t}_k) \\ \text{s.t} \quad & \boldsymbol{z}_1 = \boldsymbol{z}_{init} \\ & \boldsymbol{E}_k \boldsymbol{z}_{k+1} = \boldsymbol{c}_k(\boldsymbol{z}_k, \boldsymbol{t}_k) \\ & \underline{\boldsymbol{z}} \leq \boldsymbol{z}_k \leq \bar{\boldsymbol{z}} \end{aligned} \tag{37}$$

In this formulation, $\boldsymbol{z}$ includes all the variables used in the MPC. Note that as we want to introduce constraints for $\Delta \boldsymbol{u}$, this vector must also be included into $\boldsymbol{z}$. Consequently:

$$\boldsymbol{z}_k = [\Delta \boldsymbol{u}_k, \boldsymbol{u}_{k-1}, \boldsymbol{x}_k]^T \tag{38}$$

The objective function is:

$$f_k(\boldsymbol{z}_k, \boldsymbol{t}_k) = J_k(\boldsymbol{z}_k, \boldsymbol{t}_k) \tag{39}$$

where $\boldsymbol{t}_k$ is a vector of parameters that includes the planner $\boldsymbol{p}_k$, the parameters of the dynamic bicycle model and the cost function weights and J_k follows equation (14).

In the second restriction of (37), $\boldsymbol{c}_k(\boldsymbol{z}_k, \boldsymbol{t}_k)$ represents the system dynamics model and $\boldsymbol{E}_k$ is the selection matrix that chooses which variables are updated by the model at each time-step. In our case, the updated variables are $\boldsymbol{x}_{k+1}$ and $\boldsymbol{u}_{k-1}$. The state vector $\boldsymbol{x}_{k+1}$ uses the dynamic model (7) and the Runge-Kutta integrator (13), whilst the control vector $\boldsymbol{u}_k$ is updated by accumulation, as described by the following equations:

$$\begin{aligned} \boldsymbol{E}_k &= \begin{bmatrix} \mathbf{0} \ \mathbf{I} \end{bmatrix} \\ \boldsymbol{c}_k(\boldsymbol{z}_k, \boldsymbol{t}_k) &= \begin{bmatrix} \boldsymbol{u}_{k-1} + \Delta \boldsymbol{u}_k \\ \boldsymbol{f}_{RK}(\boldsymbol{x}_k, \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_{k-1} + \Delta \boldsymbol{u}_k), T_s) \end{bmatrix} \end{aligned} \tag{40}$$

where $\mathbf{0}$ is a 8×2 null matrix and $\mathbf{I}$ is a 8×8 identity matrix.

The constrains $\underline{\boldsymbol{z}}$ and $\bar{\boldsymbol{z}}$ are modeled by inequalities described in equations (21), (22) and (23). Finally, an initial state is given to the solver using the current state of the car and actuators:

$$\boldsymbol{z}_{init} = \begin{bmatrix} \Delta \boldsymbol{u}_0 \\ \boldsymbol{u}_{-1} \\ \boldsymbol{x}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_{init} \\ \mathbf{0} \\ \boldsymbol{x}_{init} \end{bmatrix} \tag{41}$$

with

$$\boldsymbol{u}_{init} = \begin{bmatrix} \delta_0 \\ a_0 \end{bmatrix} \text{ and } \boldsymbol{x}_{init} = \begin{bmatrix} x_0 \\ y_0 \\ \Theta_0 \\ v_{x0} \\ v_{y0} \\ \omega_0 \end{bmatrix} \tag{42}$$

A.2.2 Actuators Outputs

After solving the NMPC problem, the output control vector $\boldsymbol{u} = \begin{bmatrix} \delta \\ a \end{bmatrix}$ contains the raw outputs of the MPC. A set of transformations are needed to adapt these output to those required by the car actuators. These transformations are detailed below.

Regarding the steering wheel actuator, the MPC works with the front wheel steering angle which has a range of $[-23°, 23°]$. However, the steering wheel actuator has a range of $[-90°, 90°]$ and it uses one byte to encode this information. In order to send the correct the information to the steering wheel actuator the following transformation must be applied:

$$\delta_{actuator} = (3.913\,\delta + \pi)\frac{255}{\pi} \tag{43}$$

As for the acceleration commands, our car has two different actuators: an electric motor and a hydraulic service brake. When the acceleration is positive, a torque value (Γ) is sent to the inverter that controls the electric motor using the following equation:

$$\Gamma = m \cdot a \cdot r \tag{44}$$

where m is the vehicle mass, a is the MPC output acceleration and r is the rear wheels radius.

On the other hand, when the acceleration is negative a pressure value (P) is sent to the service brake controller. This pressure, which is sent as an integer from 0 to 30 bars, is related to the deceleration using the following equation.

$$P = \left\lceil \frac{m \cdot a \cdot r_{dyn}}{10^5 \mu_{brakes} R_{reff}(A_{pisF} N_f + A_{pisR} N_r)} \right\rceil \tag{45}$$

where r_{dyn} denotes the wheels dynamic radius, μ_{brakes} represents the brake disks coefficient, N_f and N_r are the number of brake pads at each axle, A_{pisF} and A_{pisR} are the front and rear pistons' area.

A.2.3 Parameters Tuning

Parameters play a key role in the MPC performance and need to be tuned in order to achieve the best possible performance. Tunable parameters include the MPC weights described in equations (16), (18), (19) and (20), Pacejka Tyre Model parameters (9), longitudinal force formula's parameters (8) and constrains (21), (22) and (23). Once the basic MPC algorithm is implemented and ready to run, the different approaches described below are followed to adjust the parameters.

Weights

Weights are adjusted by increasing those associated to the error terms which generate a higher error in the cost function. In order to implement this methodology, each independent term inside the cost function is plotted in a time/error plot to visualize the terms with a higher influence as shown in Figure 39. The ROS tool *rqt_plot* is used to generate these graphics. It is visible that the lateral error is the most important (highest) error.

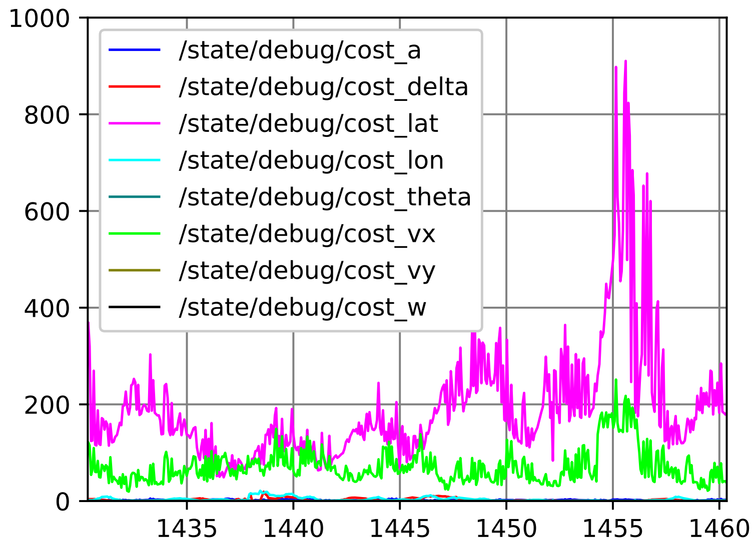

Figure 39: MPC cost function terms.

Pacejka Tyre Model

An initial approximation of the Magic Formula (9) parameters was achieved thanks to the BCN eMotorsport Vehicle Dynamics department work [26]. This department achieved an accurate set of parameters that play a key role in the Magic formula. Figure 40 shows how this model behaves for some arbitrary F_z values. By setting F_z to the current XALOC 2.0 values, a good approximation of the parameters can be estimated. Then, these parameters can be slightly changed depending on the car performance during simulation and testing.

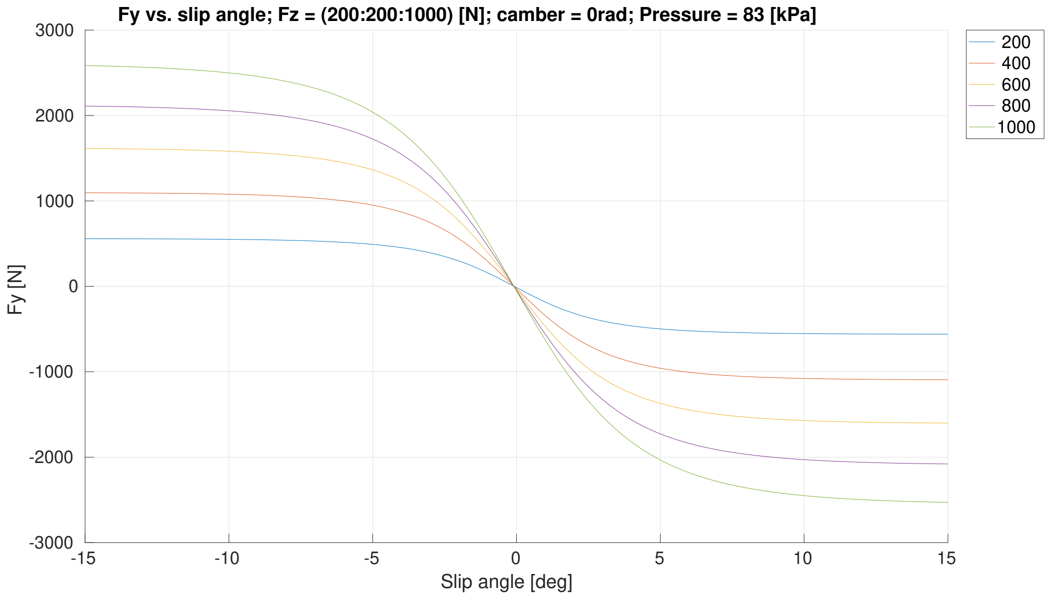

Figure 40: Simplified Magic Formula Model. Fy as a function of the slip angle.

Constrains

In our model, the constraints are set as parameters in order to be able to dynamically change them if necessary. Constraints for the control variables (δ, a) are set to their physical limits. On the other hand, constraints for the state variables are set to values that ensures the car safety. Finally, slew rate constrains are set to avoid abrupt changes that could damage the actuators. Smooth changes of the control variables allow a better behavior of the actuators controllers.

These parameters are included in a *YAML* file, which can be dynamically read using the ROS tool *rqt_rqconfigure*. This method allows changing the parameters values in real time when the algorithm is running. A working set of parameter values is shown in Section 5.

A.3 Linear Parameter Varying MPC Appendix

A.3.1 Linearization of the Dynamic Bycicle Model

- The global position variables, x and y, are substituted by two new variables, x_l and y_l, that represent the position with respect to the car current position (change of the origin of the coordinate frame). For the sake of simplicity, from now on variables x_l and y_l will be denoted as x and y, respectively.
- A new variable, denoted as ψ, is included into the state vector $\boldsymbol{x}$ to represent the local yaw angle of the vehicle (yaw angle with respect to the current vehicle position). This variable is necessary to linearize the differential equations that model the local

longitudinal and lateral displacement (first two equations in (7)). Assuming small angle condition for ψ, the dynamics for the local longitudinal and lateral positions become linear ($\cos\psi \approx 1$, $\sin\psi \approx \psi$).

- In equation (8), second ($\frac{1}{2}C_d\rho A_r v_x^2$) and third terms ($\mu_r mg$) are multiplied and divided by v_x. Then, we set the v_x in the denominators and the quadratic v_x as varying parameters denoted as $\hat{v}_x$ (a value that is constant but depends on the previous iteration steps and current vehicle state).
- The lateral forces $F_{y,F}$ and $F_{y,R}$ are linearized following equations (46) below, where the lateral forces are parameterized as linear functions that depends on the slip angles (α).
- The slip angles are considered to be small and the tan^{-1} can be neglected following equations (47) below.
- The remaining longitudinal velocities that appear in the differential equations for $\dot{y}$, $\dot{v}_x$, $\dot{v}_y$ and $\dot{\omega}$ are set as varying parameters ($\hat{v}_x$).
- The lateral velocity in the differential equation for $\dot{x}$ is set as a varying parameter ($\hat{v}_y$).
- The steering angles that are inside trigonometric functions are set as varying parameters ($\hat{\delta}$).

$$\begin{aligned} F_{y,F} &= C_F\alpha_F \\ F_{y,R} &= C_R\alpha_R \end{aligned} \tag{46}$$

where C_F and C_R denote the tire stiffness coefficients and represent the slope of the linear equation.

$$\begin{aligned} \alpha_F &\approx \frac{v_y + L_F\omega}{v_x} - \delta \\ \alpha_R &\approx \frac{v_y - L_R\omega}{v_x} \end{aligned} \tag{47}$$

As a result of the linearizations, the following set of linear differential equations is obtained:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \\ \dot{\psi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x - \hat{v}_y\psi \\ \hat{v}_x\psi + v_y \\ \omega \\ \omega \\ a - \frac{\mu_r mg + \frac{1}{2}C_d\rho A_r\hat{v}_x^2}{m\hat{v}_x}v_x - \frac{C_F\sin\hat{\delta}}{m\hat{v}_x}v_y - \frac{C_F L_F\sin\hat{\delta}}{m\hat{v}_x}\omega + \frac{C_F\sin\hat{\delta}}{m}\delta + \hat{v}_y\omega \\ \frac{C_R}{m\hat{v}_x}v_y - \frac{C_R L_R}{m\hat{v}_x}\omega + \frac{C_F\cos\hat{\delta}}{m\hat{v}_x}v_y + \frac{C_F L_F\cos\hat{\delta}}{m\hat{v}_x}\omega - \frac{C_F\cos\hat{\delta}}{m}\delta - \hat{v}_x\omega \\ \frac{C_F L_F\cos\hat{\delta}}{I\hat{v}_x}v_y + \frac{C_F L_F^2\cos\hat{\delta}}{I\hat{v}_x}\omega - \frac{C_F L_F\cos\hat{\delta}}{I}\delta - \frac{C_R L_R}{I\hat{v}_x}v_y + \frac{C_R L_R^2}{I\hat{v}_x}\omega \end{bmatrix} \tag{48}$$

A.3.2 Solving with FORCESPRO

Using the LPV-MPC formulation of equation (30), the problem must be written using the following form in order to solve it with FORCESPRO:

$$
\begin{aligned}
\min_{\boldsymbol{z}} \quad & \sum_{k=1}^{N} \left(\frac{1}{2}\boldsymbol{z}_k^T \boldsymbol{H}_k \boldsymbol{z}_k + \boldsymbol{f}_k^T \boldsymbol{z}_k\right) \\
\text{s.t} \quad & \boldsymbol{D}_1 \boldsymbol{z}_1 = \boldsymbol{c}_1 \\
& \boldsymbol{C}_{k-1}\boldsymbol{z}_{k-1} + \boldsymbol{D}_k \boldsymbol{z}_k = \boldsymbol{c}_k \,, \quad k > 1 \\
& \underline{\boldsymbol{z}}_k \leq \boldsymbol{z}_k \\
& \bar{\boldsymbol{z}}_k \geq \boldsymbol{z}_k
\end{aligned}
\tag{49}
$$

To fit the LPV-MPC formulation (30) into the FORCESPRO minimization problem (49), a set of vectors and matrices must be defined. Vector $\boldsymbol{z}_k = [\Delta \boldsymbol{u}_k, \boldsymbol{u}_{k-1}, \boldsymbol{x}_k]^T$ represents state variables, controls and controls slew rate. Matrix $\boldsymbol{H}_k$ includes the weights associated to the quadratic terms of the sates. Therefore:

$$
\boldsymbol{H}_k = 2 \begin{bmatrix} \boldsymbol{R} & & \\ & \boldsymbol{0} & \\ & & \boldsymbol{Q} \end{bmatrix}
\tag{50}
$$

Vector $\boldsymbol{f}_k^T$ represents the weights associated to the linear terms and it is defined as following:

$$
\boldsymbol{f}_k^T = -\text{diag}(\boldsymbol{H})
\tag{51}
$$

Matrices $\boldsymbol{D}_1$, $\boldsymbol{D}_k$, $\boldsymbol{C}_{k-1}$ and vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_k$ are defined as:

$$
\begin{aligned}
\boldsymbol{D}_1 &= -\boldsymbol{I}_{11\times 11} \\
\boldsymbol{D}_k &= -\begin{bmatrix} \boldsymbol{0}_{9\times 2} & \boldsymbol{I}_{9\times 9} \end{bmatrix}
\end{aligned}
\tag{52}
$$

$$
\boldsymbol{C}_{k-1} = \begin{bmatrix} \boldsymbol{I}_{2\times 2} & \boldsymbol{I}_{2\times 2} & \boldsymbol{0}_{2\times 7} \\ \boldsymbol{B}_{k-1} & \boldsymbol{B}_{k-1} & \boldsymbol{I}_{7\times 7} + \boldsymbol{A}_{k-1} T_s \end{bmatrix},
\tag{53}
$$

$$\boldsymbol{c}_1 = \begin{bmatrix} \delta_0 \\ a_{x0} \\ 0 \\ 0 \\ 0 \\ 0 \\ \Theta_0 \\ 0 \\ v_{x0} \\ v_{y0} \\ \omega_0 \end{bmatrix} \tag{54}$$

where $\boldsymbol{D}_1$ and $\boldsymbol{D}_k$ are the selection matrices for the first and the k^{th} iteration, with $k > 1$, respectively. In the first iteration, all variables are initialized. However, in the following steps only $\boldsymbol{x}_k$ and $\boldsymbol{u}_{k-1}$ are updated, as $\Delta\boldsymbol{u}_k$ is a free variable with constrains. $\boldsymbol{C}_{k-1}$ represents the dynamics of the system and includes $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$, whilst $\boldsymbol{c}_k$ is set to $\mathbf{0}$. Finally, $\boldsymbol{c}_1$ is set to the initial states and controls.

Once the MPC problem (49) is solved, the outputs are transformed and sent to the actuators following equations described in Section A.2.2.

A.4 Graph-Based SLAM Appendix

A.4.1 Odometry measurement model and error

Let $\boldsymbol{o}_{t-1\,t}$ be the odometry measurement between positions $\mathbf{p}_{t-1}$, $\mathbf{p}_t \in \boldsymbol{x}_{SLAM}$. A predicted odometry measurement ($\hat{\boldsymbol{o}}_{t-1\,t}$) for $\boldsymbol{o}_{t-1\,t}$ can be defined as:

$$\hat{\boldsymbol{o}}_{t-1\,t} = \mathbf{p}_t \ominus \mathbf{p}_{t-1} = \begin{bmatrix} \boldsymbol{R}^T_{\Theta_{t-1}}(\mathbf{p}^{x,y}_t - \mathbf{p}^{x,y}_{t-1}) \\ normalize(\Theta_{t-1} - \Theta_t) \end{bmatrix} \tag{55}$$

where $\boldsymbol{R}^T_{\Theta_{t-1}}$ denotes the rotation matrix associated to $\mathbf{p}_{t-1}$, $\mathbf{p}^{x,y}_{t-1}$ and $\mathbf{p}^{x,y}_t$ denote the translation vectors associated to $\mathbf{p}_{t-1}$ and $\mathbf{p}_t$, respectively, and the function $normalize()$ normalizes angles so that they are within $[-\pi, \pi]$.

With this being said, if we assume that a zero mean Gaussian noise is added to $\boldsymbol{o}_{t-1\,t}$, we can define an associated information matrix denoted as $\boldsymbol{\Omega^o}$. Then, a probability distribution function of the error between real and predicted measurements is defined as:

$$p(\boldsymbol{o}_{t-1\,t}|\mathbf{p}_{t-1}, \mathbf{p}_t) = K \cdot e^{-\frac{1}{2}(\hat{\boldsymbol{o}}_{t-1\,t} \ominus \boldsymbol{o}_{t-1\,t})^T \boldsymbol{\Omega^o} (\hat{\boldsymbol{o}}_{t-1\,t} \ominus \boldsymbol{o}_{t-1\,t})} \tag{56}$$

where K is a constant value.

If we now define an odometry measurement error as:

$$\mathbf{e}^{\boldsymbol{o}}_{t-1\,t} = \hat{\boldsymbol{o}}_{t-1\,t} \ominus \boldsymbol{o}_{t-1\,t} = \begin{bmatrix} \boldsymbol{R}^T_{\Delta\Theta_{t-1\,t}}(\boldsymbol{R}^T_{\Theta_{t-1}}(\mathbf{p}^{x,y}_t - \mathbf{p}^{x,y}_{t-1}) - \Delta\mathbf{p}_{t-1\,t}) \\ normalize(\Delta\Theta_{t-1\,t} + \Theta_t - \Theta_{t-1}) \end{bmatrix} \tag{57}$$

Note that $\mathbf{e}^{\boldsymbol{o}}_{t-1\,t}$ is a function of $\mathbf{p}_{t-1}$ and $\mathbf{p}_t$. Taking logarithms in equation (56) and ignoring constant parts, a proportional log-likelihood probability ${\mathbf{e}^{\boldsymbol{o}}_{t-1\,t}}^T\boldsymbol{\Omega}^{\boldsymbol{o}}\mathbf{e}^{\boldsymbol{o}}_{t-1\,t}$ can be assigned to the edge relating nodes $\mathbf{p}_{t-1}$ and $\mathbf{p}_t$.

A.4.2 Landmark observation model and error

Let $\boldsymbol{l}_{it}$ be the observation of landmark $\boldsymbol{m}_i$ detected in position $\mathbf{p}_t$, where $\boldsymbol{m}_i$, $\mathbf{p}_t \in \boldsymbol{x}_{SLAM}$. A model to give a predicted landmark observation ($\hat{\boldsymbol{l}}_{it}$) for $\boldsymbol{l}_{it}$ can be defined as:

$$\hat{\boldsymbol{l}}_{it} = \boldsymbol{R}^T_{\Theta_t}(\boldsymbol{m}^{x,y}_i - \mathbf{p}^{x,y}_t) \tag{58}$$

where $\boldsymbol{m}^{x,y}_i$ denotes the global position (x, y) of landmark $\boldsymbol{m}_i$.

With this being said, if we assume that a zero mean Gaussian noise is added to $\boldsymbol{l}_{it}$, we can define an associated information matrix denoted as $\boldsymbol{\Omega}^{\boldsymbol{l}}$. Then, a probability distribution function of the error between real and predicted observations is defined as:

$$p(\boldsymbol{l}_{it}|\boldsymbol{m}_i, \mathbf{p}_t) = K \cdot e^{-\frac{1}{2}(\hat{\boldsymbol{l}}_{it} - \boldsymbol{l}_{it})^T\boldsymbol{\Omega}^{\boldsymbol{l}}(\hat{\boldsymbol{l}}_{it} - \boldsymbol{l}_{it})} \tag{59}$$

where K is a constant value.

If we now define a landmark observation error as:

$$\mathbf{e}^{\boldsymbol{l}}_{it} = \mathbf{e}^{\boldsymbol{l}}_{it}(\boldsymbol{m}_i, \mathbf{p}_t) = \hat{\boldsymbol{l}}_{it} - \boldsymbol{l}_{it} \tag{60}$$

Note that $\mathbf{e}^{\boldsymbol{l}}_{it}$ is a function of $\boldsymbol{m}_i$ and $\mathbf{p}_t$. Taking logarithms in equation (59) and ignoring constant parts, a proportional log-likelihood probability ${\mathbf{e}^{\boldsymbol{l}}_{it}}^T\boldsymbol{\Omega}^{\boldsymbol{l}}\mathbf{e}^{\boldsymbol{l}}_{it}$ can be assigned to the edge relating nodes $\boldsymbol{m}_i$ and $\mathbf{p}_t$.

A.4.3 Gauss-Newton algorithm

The first step of the Gauss Newton algorithm is the linearization of the cost function (36). This linearization is done via a first order Taylor expansion around an initial guess, denoted as $\tilde{\boldsymbol{x}}_{SLAM}$. Applying the linearization, the resulting cost function is (see [13] for the full mathematical development of the linearization):

$$\begin{aligned} \mathbf{F}(\tilde{\boldsymbol{x}}_{SLAM} + \Delta\boldsymbol{x}_{SLAM}) &= \sum_{\forall(i,j)} \mathbf{F}_{ij}(\tilde{\boldsymbol{x}}_{SLAM} + \Delta\boldsymbol{x}_{SLAM}) = \\ &= c + 2\mathbf{b}^T\Delta\boldsymbol{x}_{SLAM} + \Delta\boldsymbol{x}^T_{SLAM}\mathbf{H}\Delta\boldsymbol{x}_{SLAM} \end{aligned} \tag{61}$$

In equation (61), $\tilde{\boldsymbol{x}}_{SLAM} = [\tilde{\mathbf{p}}_1, \ldots, \tilde{\mathbf{p}}_T, \tilde{\boldsymbol{m}}_1, \ldots, \tilde{\boldsymbol{m}}_M]^T$ is the initial guess. For positions $\tilde{\mathbf{p}}$, the GPS positions $\mathbf{p}_{GPS}$ are used. For landmarks $\tilde{\boldsymbol{m}}$, the landmarks created by the data association process are used. Terms c, $\mathbf{b}$ and $\mathbf{H}$ are defined as:

$$
\begin{aligned}
c &= \sum \mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij} \\
\mathbf{b} &= \sum \mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij} \\
\mathbf{H} &= \sum \mathbf{J}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij}
\end{aligned}
\tag{62}
$$

where all terms are evaluated in $\tilde{\boldsymbol{x}}_{SLAM}$ and $\mathbf{J}_{ij} = [\mathbf{J}_i[ij], \mathbf{J}_j[ij]]^T$ is the Jacobian of $\mathbf{e}_{ij}(\boldsymbol{x}_{SLAM_i}, \boldsymbol{x}_{SLAM_j})$. Using the Jacobian $\mathbf{J}_{ij}$, matrix $\mathbf{H}$ can be defined. This matrix contains a set of submatrices $\mathbf{H}_{ij}$, represented as:

$$
\mathbf{H}_{ij} = \begin{bmatrix}
\ddots & & & & \\
& \mathbf{J}_i[ij]^T \boldsymbol{\Omega}_{ij} \mathbf{J}_i[ij] & \ldots & \mathbf{J}_i[ij]^T \boldsymbol{\Omega}_{ij} \mathbf{J}_j[ij] & \\
& \vdots & \ddots & \vdots & \\
& \mathbf{J}_j[ij]^T \boldsymbol{\Omega}_{ij} \mathbf{J}_i[ij] & \ldots & \mathbf{J}_j[ij]^T \boldsymbol{\Omega}_{ij} \mathbf{J}_j[ij] & \\
& & & & \ddots
\end{bmatrix}
\tag{63}
$$

where $\mathbf{J}_i[ij]$ and $\mathbf{J}_j[ij]$ are the derivatives of the error function with respect to $\boldsymbol{x}_{SLAM_i}$ and $\boldsymbol{x}_{SLAM_j}$, respectively. This matrix contains null elements (denoted as dots in the matrix) which are placed in the indices of the other hidden nodes different than i and j. This way, $\mathbf{H} = \sum_{\forall(i,j)} \mathbf{H}_{ij}$, as $\mathbf{H}$ and $\mathbf{H}_{ij}$ are matrices of the same size.

Note that Jacobians $\mathbf{J}_i[ij]$ and $\mathbf{J}_j[ij]$ are different for landmark and position nodes.

For **position** nodes, the derivatives are:

$$
\begin{aligned}
\mathbf{J}_{\mathbf{p}_i}^{\boldsymbol{o}}[ij] = \frac{\partial \mathbf{e}_{ij}^{\boldsymbol{o}}}{\partial \mathbf{p}_i} &= \begin{bmatrix} -\boldsymbol{R}_{\Delta\Theta_{ij}}^T & \boldsymbol{R}_{\Delta\Theta_{ij}}^T \boldsymbol{R}_{\Theta_i}^T (\mathbf{p}_j^{x,y} - \mathbf{p}_i^{x,y}) \\ \mathbf{0}_{1\times 2} & -1 \end{bmatrix} \\
\mathbf{J}_{\mathbf{p}_j}^{\boldsymbol{o}}[ij] = \frac{\partial \mathbf{e}_{ij}^{\boldsymbol{o}}}{\partial \mathbf{p}_j} &= \begin{bmatrix} \boldsymbol{R}_{\Delta\Theta_{ij}}^T \boldsymbol{R}_{\Theta_i}^T \boldsymbol{R}_{\Theta_j} & \mathbf{0}_{2\times 1} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix}
\end{aligned}
\tag{64}
$$

For **landmark** nodes, the derivatives are:

$$
\mathbf{J}_{\boldsymbol{m}_i}^{\boldsymbol{l}}[ij] = \frac{\partial \mathbf{e}_{ij}^{\boldsymbol{l}}}{\partial \boldsymbol{m}_i} = \begin{bmatrix} -\mathbf{I}_{2\times 2} & (\boldsymbol{R}'_{\Theta_j})^T (\boldsymbol{m}_i - \mathbf{p}_j^{x,y}) \end{bmatrix}, \ \mathbf{J}_{\mathbf{p}_j}^{\boldsymbol{l}}[ij] = \frac{\partial \mathbf{e}_{ij}^{\boldsymbol{l}}}{\partial \mathbf{p}_j} = \boldsymbol{R}_{\Theta_j}^T
\tag{65}
$$

where $\boldsymbol{R}'_{\Theta} = \dfrac{\partial \boldsymbol{R}_{\Theta}}{\partial \Theta}$.

If we now take the derivative with respect to $\Delta\tilde{\boldsymbol{x}}_{SLAM}$ on equation (61), we get the following linear system:

$$\mathbf{H}\Delta\boldsymbol{x}_{SLAM} = -\mathbf{b} \tag{66}$$

We can now solve the system and obtain $\Delta\boldsymbol{x}_{SLAM} = [\Delta\mathbf{p}, \Delta\boldsymbol{m}]^T$. The next step is to add this vector to the current initial guess $\tilde{\boldsymbol{x}}_{SLAM}$. This operation is different for positions and landmarks. For positions, a translation and a rotation must be applied giving place to the following equation:

$$\tilde{\mathbf{p}}_i = \tilde{\mathbf{p}}_i \oplus \Delta\mathbf{p}_i = \begin{bmatrix} \mathbf{R}_{\Theta_i}\tilde{\mathbf{p}}_i^{x,y} + \Delta\mathbf{p}_i^{x,y} \\ normalize(\Theta_i + \Delta\Theta_i) \end{bmatrix} \tag{67}$$

On the other hand, a simple sum operation is done for landmarks following the next equation:

$$\tilde{\mathbf{m}}_i^{x,y} = \tilde{\mathbf{m}}_i^{x,y} + \Delta\mathbf{m}_i^{x,y} \tag{68}$$

For the sake of simplicity, equations (67) and (68) are simplified into the following equation:

$$\tilde{\boldsymbol{x}}_{SLAM} = \tilde{\boldsymbol{x}}_{SLAM} \oplus \Delta\boldsymbol{x}_{SLAM} \tag{69}$$

Once the linearization is done and the linear system is presented we have all the elements to apply the Gauss-Newton algorithm.

Algorithm

The Gauss-Newton algorithm is an iterative algorithm that applies the linearization (61), solves the linear system (66) and applies equation (69) at every step until a convergence condition is met. The Gauss-Newton Algorithm for the Graph-SLAM problem is defined in Algorithm 7.

Algorithm 7 SLAM Gauss-Newton Algorithm

1: **procedure** GAUSS-NEWTON($\tilde{\boldsymbol{x}}_{SLAM}$)
2: $\quad\boldsymbol{x} = \tilde{\boldsymbol{x}}_{SLAM}$
3: $\quad$**while** $\Delta\boldsymbol{x}_{SLAM} < TH$ **do** $\triangleright$ Algorithm threshold
4: $\quad\quad$**for** $\forall i, j$ **do**
5: $\quad\quad\quad$**if** $\boldsymbol{x}_i == \mathbf{p}_i$ && $\boldsymbol{x}_j == \mathbf{p}_j$ **then** $\triangleright$ Odometry edge
6: $\quad\quad\quad\quad\mathbf{H}[ij] \leftarrow \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij],\ \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij]$ $\triangleright$ Fill H matrix
7: $\quad\quad\quad\quad\mathbf{H}[ij] \leftarrow \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij],\ \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij]$
8: $\quad\quad\quad\quad\mathbf{b}[i] \leftarrow \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_i}[ij]\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{e}^{\boldsymbol{o}}_{ij}$ $\triangleright$ Fill vector $\mathbf{b}$
9: $\quad\quad\quad\quad\mathbf{b}[j] \leftarrow \mathbf{J}^{\boldsymbol{o}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{o}}_{ij}\mathbf{e}^{\boldsymbol{o}}_{ij}$
10: $\quad\quad\quad$**else** $\triangleright$ Landmark observation edge
11: $\quad\quad\quad\quad\mathbf{H}[ij] \leftarrow \mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij],\ \mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij],$ $\triangleright$ Fill H matrix
12: $\quad\quad\quad\quad\mathbf{H}[ij] \leftarrow \mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij],\ \mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij]$
13: $\quad\quad\quad\quad\mathbf{b}[i] \leftarrow \mathbf{J}^{\boldsymbol{l}}_{\boldsymbol{m}_i}[ij]\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{e}^{\boldsymbol{l}}_{ij}$ $\triangleright$ Fill vector $\mathbf{b}$
14: $\quad\quad\quad\quad\mathbf{b}[j] \leftarrow \mathbf{J}^{\boldsymbol{l}}_{\mathbf{p}_j}[ij]^T\boldsymbol{\Omega}^{\boldsymbol{l}}_{ij}\mathbf{e}^{\boldsymbol{l}}_{ij}$

15: $\quad\quad\quad\Delta\boldsymbol{x} \leftarrow -\mathbf{H}^{-1}\mathbf{b}$ $\triangleright$ Solve linear system
16: $\quad\quad\quad\boldsymbol{x} = \boldsymbol{x} \oplus \Delta\boldsymbol{x}_{SLAM}$ $\triangleright$ Add solution to previous SLAM estimate
$\quad$**return** $\boldsymbol{x}$

The algorithm applies the first order Taylor expansion to fill $\mathbf{H}$ and $\mathbf{b}$ in steps 5 to 14, depending on the type of edge. If the edge is an odometry measurement, equations (64) are applied to obtain the Jacobians and (57) is used to obtain the log-likelihood probability $\mathbf{e}_{ij}$. On the other hand, if the edge is a landmark observation, equations (65) are applied to obtain the Jacobians and (60) is used to obtain the log-likelihood probability $\mathbf{e}_{ij}$. Then the linear system (66) is solved and equation (69) is applied. The SLAM Gauss-Newton algorithm iterates this process until the linear system solution $\Delta\boldsymbol{x}_{SLAM}$ is smaller than a certain threshold TH.

References

[1] Formula Student Germany. Fsg competition handbook 2021. 2021.

[2] Juraj Kabzan, M. I. Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, A. Bühler, N. Gosala, Mehak Gupta, R. Sivanesan, Ankit Dhall, E. Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, I. Sa, Renaud Dubé, Abel Gawel, M. Pfeiffer, Alexander Liniger, J. Lygeros, and R. Siegwart. Amz driverless: The full autonomous racing system. *ArXiv*, abs/1905.05150, 2019.

[3] Sherif Nekkah, Josua Janus, Mario Boxheimer, Lars Ohnemus, Stefan Hirsch, B. Schmidt, Y. Liu, David Borb'ely, Florian Keck, Katharina Bachmann, and L. Bleszynski. The autonomous racing software stack of the kit19d. *ArXiv*, abs/2010.02828, 2020.

[4] W. F. Milliken and D. L. Milliken. Race car vehicle dynamics. 1994.

[5] Hans B. Pacejka. Tire and vehicle dynamics (third edition). Butterworth-Heinemann, Oxford, third edition edition, 2012.

[6] Alexander Domahidi and Juan Jerez. Forces professional. Embotech AG, url=https://embotech.com/FORCES-Pro, 2014–2019.

[7] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari. Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, pages 1–17, 2017.

[8] B. P.; Teukolsky S. A.; Press, W. H.; Flannery and W. T. Vetterling. Numerical recipes in fortran: The art of scientific computing. *Cambridge University Press*, pages 704–716, 1992.

[9] Félix Martí Valverde. Development of a control algorithm for a driverless vehicle using ros. 2020.

[10] E. Alcalá, V. Puig, J. Quevedo, and Ugo Rosolia. Autonomous racing using linear parameter varying-model predictive control (lpv-mpc). *Control Engineering Practice*, 95:104270, 2020.

[11] E. Alcalá, V. Puig, and J. Quevedo. Lpv-mpc control for autonomous vehicles. *IFAC-PapersOnLine*, 52:106–113, 2019.

[12] S. Thrun and M. Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 2006.

[13] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2:31–43, 2010.

[14] J. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.

[15] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. Gg2o: A general framework for graph optimization. *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[16] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154 vol.3, 2004.

[17] Arnau Roche López. Lidar cone detection as part of a perception system in a formula student car. 2019.

[18] Eloi Bové Canals. Position estimation using a stereo camera as part of the perception system in a formula student car. 2019.

[19] Moustapha Doumiati, Ali Charara, Alessandro Victorino, Daniel Lechner, and Bernard Dubuisson. *Estimation Methods Based on Kalman Filtering*, chapter 2, pages 37–60. John Wiley Sons, Ltd, 2012.

[20] S. Srinivasan, Inkyu Sa, Alex Zyner, Victor Reijgwart, M. I. Valls, and R. Siegwart. End-to-end velocity estimation for autonomous racing. *IEEE Robotics and Automation Letters*, 5:6869–6875, 2020.

[21] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[22] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that probably converges. In *IJCAI*, 2003.

[23] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (slam) : Part i the essential algorithms. 2006.

[24] Ignasi Mallén. Development of the global race optimizer. 2020.

[25] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[26] Victor Hernández. Estudio de la viabilidad de neumáticos de perfil bajo para un formula student, 2020.