Duckrace: Iterative Learning Control for Autonomous Racing

Tesi di Laurea Magistrale in
Automation and Control Engineering

Author: **Giulio Vaccari**

Student ID: 966357
Advisor: Prof. Simone Formentin
Co-advisors: Valentina Breschi, Riccardo Busetto
Academic Year: 2021-22

Abstract

This thesis presents a Learning Model Predictive Control (LMPC) approach for controlling a Duckiebot in a racing environment based on Duckietown. By leveraging online learning and optimization-based control, our proposed approach enables the Duckiebot to adapt and minimize its lap times while navigating through the environment. Through our work we demonstrate how the LMPC can be applied to the Duckiebot and we present a new approach based on a convex hull to constraint the robot inside the track margins.

We establish a dedicated Duckietown laboratory at Politecnico di Milano, specifically designed for racing and research purposes, equipped with multiple Duckiebots and a variety of hardware and software tools.

The thesis starts by exploring a Model Predictive Controller (MPC) for trajectory following and evolves it into the LMPC. In the thesis, we demonstrate that the originally proposed formulation of LMPC is not very effective in the real environment with the considered model. Therefore, we propose a new formulation that integrates preview into LMPC. The new formulation in the circuit is able to improve lap time by 44% from the first iteration while consistently respecting the track margins.

Keywords: LMPC, Duckietown, MPC, Autonomous Driving

Abstract in lingua italiana

Questo lavoro di tesi presenta un approccio di controllo basato su Learning Model Predictive Control (LMPC) per guidare un Duckiebot in un ambiente di gara basato su Duckietown. Sfruttando l'apprendimento online e il controllo basato sull'ottimizzazione, il nostro approccio proposto consente al Duckiebot di adattarsi e minimizzare i tempi sul giro mentre naviga nell'ambiente di guida. Attraverso il nostro lavoro, dimostriamo come il LMPC possa essere applicato al Duckiebot e presentiamo un nuovo approccio basato su convex hull per vincolare il robot all'interno dei margini della pista.

Abbiamo stabilito un laboratorio Duckietown dedicato presso il Politecnico di Milano, specificamente progettato per scopi di ricerca e gara, dotato di numerosi Duckiebots e una varietà di strumenti hardware e software.

La tesi inizia esplorando un controllore di tipo Model Predictive Controller (MPC) per il trajectory following e lo sviluppa successivamente in un LMPC. Nella tesi dimostriamo che la formulazione originariamente proposta del LMPC non è molto efficace nell'ambiente reale col modello considerato. Proponiamo quindi una nuova formulazione che integra la preview nel LMPC. La nuova formulazione nel circuito è in grado di migliorare del 44% il tempo sul giro già dalla prima iterazione, rispettando allo stesso tempo in modo coerente i margini della pista.

Parole chiave: LMPC, Duckietown, MPC, Autonomous Driving

Contents

1 | Introduction

Autonomous driving has gained significant attention in recent years, with the potential to revolutionize the transportation industry and improve safety on the roads. While much of the research in this field has focused on the development of autonomous vehicles for everyday use, there has also been a growing interest in applying autonomous technology to the realm of racing. Racing presents a unique challenge for autonomous driving, as it requires not only the ability to navigate complex environments, but also the ability to perform at high speeds and make quick decisions.

One approach to achieving high performance in autonomous racing is through the use of iterative learning control (ILC). ILC is a control technique that involves the repetition of a control process over multiple time steps, with the aim of improving the performance of the system through the accumulation of small adjustments. This approach has the potential to enable autonomous vehicles to continuously improve their performance over time, making it well-suited for the dynamic and high-stakes environment of racing.

In this thesis, we will explore the use of ILC for autonomous racing, including its potential benefits and limitations. We will focus on the use of Learning Model Predictive Control (LMPC), a recently proposed ILC technique that has the potential to revolutionize the field. We will also discuss the challenges and considerations involved in the implementation of ILC in autonomous racing systems, and present case studies demonstrating its effectiveness in real-world racing scenarios. Through this analysis, we aim to provide a comprehensive overview of the use of LMPC for autonomous racing and to contribute to the development of more advanced and efficient autonomous racing systems.

1.1. Duckietown

Duckietown is a research and educational platform for the development and testing of autonomous vehicle technologies. It is an open-source project that provides a miniature city environment for the development and testing of self-driving vehicles, with a focus on simplicity, affordability, and scalability.

Duckietown has gained widespread popularity in the field of autonomous vehicle research, and has been used by researchers and educators around the world to develop and test various autonomous vehicle algorithms and systems. It has also been widely adopted as a teaching tool in universities and other academic institutions, providing students with a hands-on experience in the development of autonomous vehicles.

Its usability and its realism are making it an industry standard to benchmark robotics algorithms in a controlled and reproducible environment. Duckietown started as a class at MIT in 2016 and since then has expanded worldwide: it now includes several open source courses as well as an international competition. The organization target is being able to learn how to build code for autonomy without having to worry about designing the simulation and developing the necessary hardware, making it possible to be used already in high schools. Duckietown in Politecnico di Milano has been introduced in 2021 by the Automation Engineering Association, with the idea of competing to the Artifical Intelligence Driving Olympics, an international competition that takes place twice during the two of the most prestigious international conferences in machine learning and robotics. The aim of the competition is to be able to drive as far as possible in a given time window. Duckietown consists in a model city, where cars are Duckiebots with ducks at the wheel. Other than the physical track a very accurate gym environment is available to test the algorithms in a virtual environment. As we will see the environment is deeply customizable and is ideal for a wide range of different use cases.

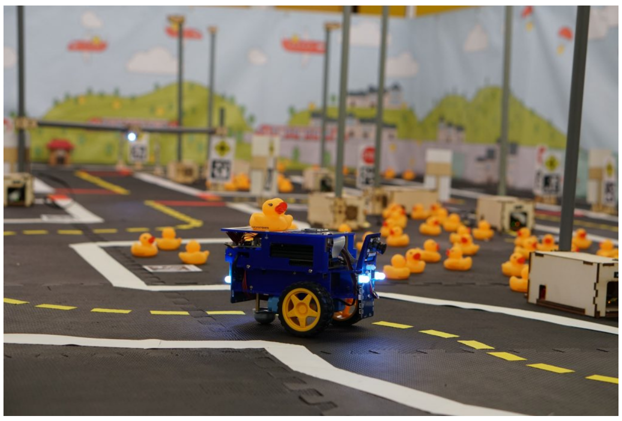


Figure 1.1: A Duckiebot in a Duckietown. Source: duckietown.org

1.2. Previous work on Duckietown

Being Duckietown highly flexible and customizable but easy to reproduce many different works have been done using the platform. Many pubblications directly propose new approach to control, such as an approach based on Imitation Learning trained both on simulation and in real world[9], robust Reinforcement Learning [11] and multi robot cooperation [7]. Some other interesting examples span to more broad areas such as an approach to automatic camera calibrations[5], a system to predict if an input is too different that what the model has been trained on[8] and even a blockchain based validation procedure [6].

Overall, the publications related to Duckietown provide a wealth of information on this important platform and its various applications in autonomous vehicle research and education. They offer valuable insights and lessons learned that can be applied to the development of autonomous vehicle technologies, and contribute to the ongoing advancement of this rapidly growing field.

To the best of our knowledge, there are currently no published studies that have utilized the Duckietown platform to validate autonomous racing algorithms or iterative controllers based on model predictive control (MPC). This gap in the literature suggests a potential opportunity for further research in this area, using the Duckietown platform as a testbed for the development and evaluation of autonomous racing algorithms and MPC-based iterative controllers. Such a study could provide valuable insights into the performance and limitations of these approaches in the dynamic and high-stakes environment of racing, and contribute to the ongoing development of more advanced and efficient autonomous racing systems.

1.3. Autonomous racing

We are living in an historical moment where we are very close to be able to fully roboticize our favorite road vehicles but there are still many challenges ahead, and it is still very hard to test the new technologies on the field. Historically Formula 1 and other car racing competitions have been very effective in pushing the development of cutting edge technologies that could then be adopted for general purposes. This is the reason why autonomous racing is being pushed a lot by various different and independent parties. Being able to win an autonomous race does not mean being able to drive in a urban environment because of all the external factors that do not exist in a controlled racing track. The improvements realized for racing are still very useful both to build a robust perception system and to be able to safely control the car in safety critical situations, when the car needs to execute rapid and complex maneuvers.

Despite Duckietown is originally designed to be used as an urban environment its modularity makes it an excellent tool to design tracks of different shapes. Furthermore, despite its limited dimensions, the duckiebot can reach a good speed, making it possible to have a wide range of different speeds to take into account different scenarios.

1.4. Optimal path planning for autonomous racing

Optimal path planning is a crucial aspect of autonomous racing, as it determines the trajectory that an autonomous vehicle will follow during the race. The goal of optimal path planning is to find the fastest and safest path for the vehicle to complete the race, taking into account various factors such as track geometry, vehicle dynamics, and environmental conditions.

Optimal trajectory planning is a topic that has received significant attention in the field of autonomous racing, with a wide range of approaches and techniques proposed, including heuristic algorithms, machine learning techniques, and optimization algorithms. However, there are still many challenges and open questions in this area, including the development of robust and scalable approaches that can handle a wide range of racing scenarios and conditions, and the integration of optimal path planning with other key autonomous vehicle technologies such as perception, localization, and control. Furthermore, many of these approaches have focused on finding the optimal trajectory offline and following it throughout the race, without considering the impact of variations in the trajectory dictated by the presence of other vehicles or obstacles on the track, or the handling of uncommon environments such as dirt roads and rallies. This approach may not always be optimal, as it does not allow for the real-time adaptation of the trajectory to changing conditions or the exploitation of opportunities to maximize the performance of a specific vehicle in a specific environment [10].
A different online approach has been developed by Rosolia and Borelli [12]. In this approach an Iterative Model Predictive Controller is used to learn and improve the trajectory at each lap, using the last iterations as baselines to improve. As we will see this approach has the advantage to be able to adapt sudden changes in the track conditions such as obstacles and other moving cars, while maximizing the car performance. Using the MPC we are also able to estimate car model parameters online, further improving the quality of the newly generated trajectory, that strongly relies on the vehicle model.
An Iterative Learning MPC for autonomous driving is not a new thing, there have already been some publications on the topic, but the model and the approach to build the LMPC is always very close to the original publication [13]. Duckietown represents a unique challenge in the development of an iterative MPC: it is difficult to build a model of the duckiebot that takes into account in its state an accurate distance from the track border, making it harder to set the constraint to stay inside the track. As we will see a

solution to this problem will also open a new way to dynamically take into account fixed and moving obstacles in the track. On the other hand the Duckietown gym environment is ideal to virtually collect iterative high quality data at each lap, making it the perfect solution to test an iterative controller.

1.5. Innovative Contributions

The proposed thesis work makes several innovative contributions to the field of autonomous racing control. Firstly, it investigates the use of Learning-based Model Predictive Control (LMPC) with a Duckiebot using a model in Cartesian coordinates. The LMPC approach can capture the nonlinear dynamics of the system and adapt to varying track conditions and vehicle models, providing better control performance compared to traditional MPC approaches.

Secondly, the thesis proposes an innovative way of handling the track margins in Cartesian coordinates. Previous work always handled curvilinear coordinates. This raises new challenges in respecting the constraints.

Thirdly, the thesis uses a physical track to test the LMPC in the real world, providing a realistic testing ground for evaluating the proposed LMPC formulation's performance. The environment allows for easy experimentation and parameter tuning, and demonstrates the issues that may arise in real world scenarios. In particular the thesis demonstrates that the Duckiebot does not work well with the originally proposed formulation of the LMPC and in general with MPCs without preview. This will require some rework of the original formulation.

Overall, the proposed thesis work presents a novel LMPC formulation that can improve the handling of track margins in autonomous racing and different car models. The incorporation of a safety margin constraint and the use of a simulation environment enhances the safety and reliability of autonomous racing vehicles, paving the way for their deployment in real-world scenarios.

2 | Problem statement

Autonomous racing is an emerging field that demands efficient and accurate control algorithms to enable high-speed maneuvering around a racetrack. Model Predictive Control (MPC) is a popular approach for controlling autonomous racing vehicles, which involves generating a trajectory based on the vehicle's dynamics and following it controlling the car with a MPC. However, these approaches have limitations in maximizing the car performance and in behaving with different track conditions (like wet surface), which can impact the vehicle's safety and performance.

Learning-based Model Predictive Control (LMPC) is a promising variant of MPC that utilizes iterative control to improve controller performance, optimizing the trajectory based on previous iterations. LMPC has shown great potential in autonomous racing, but its development is very recent and its capabilities has not been fully explored.

Therefore, the aim of this thesis is to investigate the use of LMPC in controlling a mobile robot, more specifically a Duckiebot, improving the handling of track margins in autonomous racing and testing new different implementation of the controller. Specifically, the thesis will develop a novel LMPC formulation that can learn the dynamics of the vehicle and racetrack and adapt to different car models and track conditions. The proposed formulation will also incorporate a safety margin constraint to ensure that the vehicle's trajectory remains within safe boundaries.

The thesis will evaluate the performance of the proposed LMPC formulation using a Duckietown simulation environment and a physical track. Using the physical track we will demonstrate how the original formulation of the LMPC is not capable of handling the specific model of a Duckiebot without some rework. The results of this study can provide insights into the applicability of LMPC in autonomous racing and improve the development of high-speed racing algorithms. Ultimately, the proposed approach can enhance the safety and performance of autonomous racing vehicles and pave the way for their deployment in real-world scenarios.

3 | Experimental Setup

In this chapter the experimental setup based on Duckietown will be presented.

3.1. The equipment

Duckietown consists of a modular track, autonomous vehicles known as Duckiebots, and various elements to create a miniature city environment. These elements include traffic lights, road signs, and a large number of rubber ducks, which serve as obstacles and landmarks for the Duckiebots to navigate around. The modular nature of the Duckietown track allows for the creation of a wide range of different environments and scenarios, providing a rich testing ground for the development and evaluation of autonomous vehicle algorithms and systems.

In our study, we chose to focus on a specific set of tasks and scenarios for the evaluation of our autonomous vehicle algorithms and systems. As a result, we did not need to utilize all of the elements that Duckietown provides, such as traffic lights and road signs, in order to achieve our research objectives. While these elements can be useful for creating more complex and realistic environments, they were not necessary for the specific tasks and scenarios we were addressing in our study. Follows our hardware list.

1. 15 black foam "road" interlocking tiles of size 0.6x0.6m (2x2ft).
2. Duckietown compliant duck tape to delimit road borders.
3. Two duckiebots.
4. A watchtower.

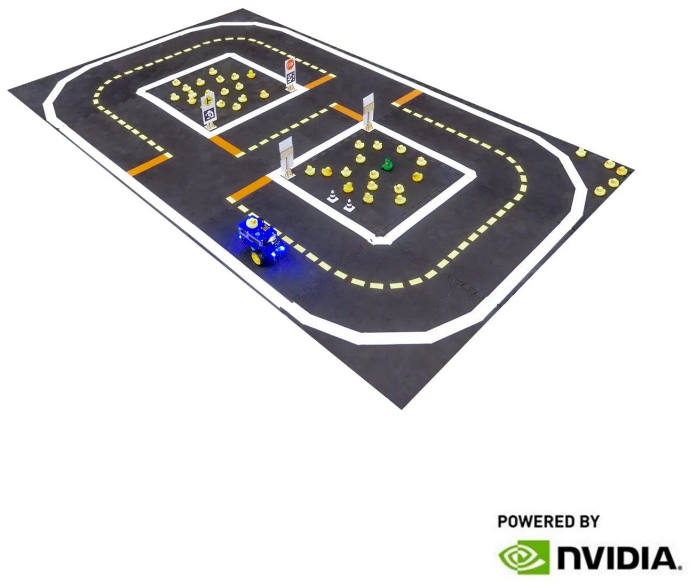


Figure 3.1: The small Duckietown track. Source: duckietown.org

3.2. Duckiebots

Duckiebots are the autonomous vehicles used in the Duckietown platform for the development and testing of autonomous vehicle technologies. These vehicles are equipped with a range of sensors and actuators that allow them to perceive and interact with their environment.

The key sensors used in Duckiebots are a camera, an ultrasonic sensor and an IMU. Cameras provide visual information about the surrounding environment, allowing the Duckiebot to detect and identify objects and features such as traffic lights, road signs, and other vehicles. Ultrasonic sensors use sound waves to measure the distance to objects, providing a range of sensing capabilities for the Duckiebot such as understanding the distance from a possible preceding vehicle.

In terms of hardware specifications, Duckiebots are powered by a Nvidia Jetson Nano single-board computer, which serves as the central processing unit for the vehicle. The Jetson Nano is equipped with a quad-core ARM Cortex-A57 CPU and a 128-core NVIDIA Maxwell GPU, which provide powerful computing capabilities for machine learning and high computing tasks. It also has 2 GB of LPDDR4 memory and a microSD card slot for storage. In terms of connectivity, the Jetson Nano has Gigabit Ethernet, HDMI, and USB 3.0 ports, allowing it to be easily integrated into a wide range of systems and devices. It also has a 40-pin expansion header that provides access to a range of GPIO (general-purpose input/output) and other interfaces, making it highly flexible and customizable. They are also equipped with a motor controller and motors, which allow the Duckiebot to move and control its speed and direction. In addition, Duckiebots are equipped with a power management system to ensure reliable and efficient operation.

Duckiebots are differential wheeled robots: differential robots are a type of mobile robot that are characterized by their use of differential drive, which is a type of drive mechanism that allows the robot to move and change direction by independently controlling the speed of each of its wheels. They are highly maneuverable, as they can change direction by adjusting the speed and direction of their wheels, allowing them to navigate around obstacles and move in any direction.

However, differential robots also have some limitations, such as their inability to move in a straight line without drift and their sensitivity to slippage and uneven terrain, in the Duckiebot this requires various different calibrations.

Duckiebots are designed to be controlled by giving the left and right wheel speed to a ROS node. The control of the motors and the internal ROS connectivity is handled entirely by the robot out of the box.

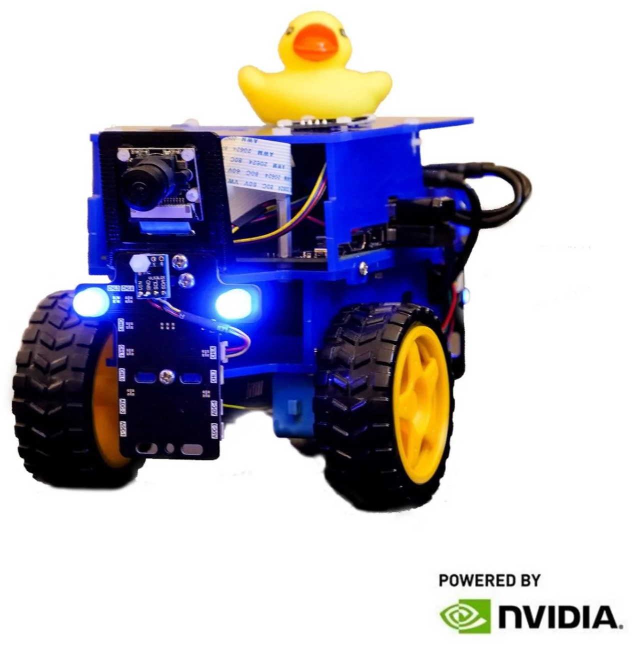


Figure 3.2: A Duckiebot. Source: duckietown.org

3.2.1. Duckiebot calibration

Calibration is an important aspect of differential robot operation, as it ensures that the robot's sensors and actuators are properly aligned and functioning correctly. During our experiments calibration has been performed periodically to ensure that the robot's performance remains consistent over time.

Duckiebot typically requires calibration of both the wheels and the camera in order to ensure optimal performance, but for our experiments the use of the onboard camera was not needed, thus making the calibration non-required for the scenario and the tasks we are addressing.

Wheels calibration

The calibration of the wheels consists in finding the optimal trim and gain parameters. The trim depends on the difference between the dimensions of the two wheels, the gain governs the speed of the wheels. To calibrate the wheels the robot runs along a straight line of known lengths, such as 2 meters. The trim and gain parameters are then adjusted until the robot is able to run straight along the entire length of the line, without drifting or veering off course. This requires careful tuning and measurement to ensure that the optimal parameters are determined.

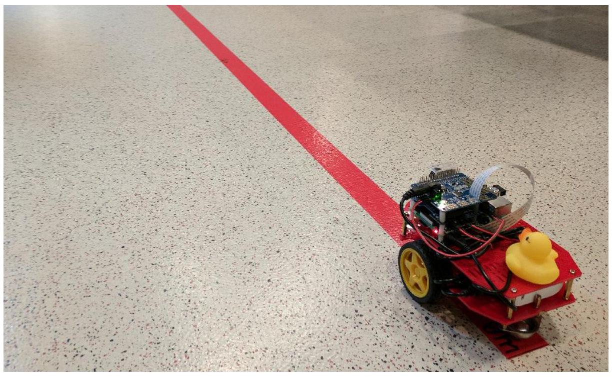


Figure 3.3: Duckiebot wheels calibration. Image from the Duckietown documentation.

3.2.2. Duckiebot mathematical model

The mathematical modeling of the Duckiebot is a crucial aspect of its development and operation, as it allows for the analysis and prediction of its behavior and performance, and for the identification of opportunities for improvement. For our task, the model is necessary to develop an effective control algorithm. The modeling of the Duckiebot has been addressed by Selcuk Ercan and its main passages are here reported. [4]

Force-Balance equations

The force-balance equations are derived under certain simplifying assumptions:

1. Pure rolling. The pure velocity at the contact point between the wheels and the ground is zero.
2. Planar workspace. Out-of-plane dynamics of a Duckiebot is negligible.
3. Longitudinal-symmetry. A duckiebot has a symmetrical mass distribution along its longitudinal axis.
4. Equal wheel radii. The length difference between the right and the left wheel radii is negligible.

Under these assumptions, the equations of motion of a duckiebot can be written as:

$$\sum F_{x'} = m(\dot{u} - v\omega) = F_{rx'} + F_{lx'} + F_{ox'} \tag{3.1}$$

$$\sum F_{y'} = m(\dot{u} + v\omega) = F_{ry'} + F_{ly'} + F_{oy'} \tag{3.2}$$

$$\sum M_z = I_z\dot{\omega} = (F_{rx'}L_1 - F_{lx'}L_2) - a(F_{ry'} + F_{ly'}) + (c-a)F_{oy'} \tag{3.3}$$

Where m is the mass, I the inertia, F_r the force on the right wheel, F_l the force on the left wheel, F_o the force on the omniwheel, u the longitudinal velocity, v the lateral velocity, ω the angular velocity. For the force orientation please see figure 3.4.

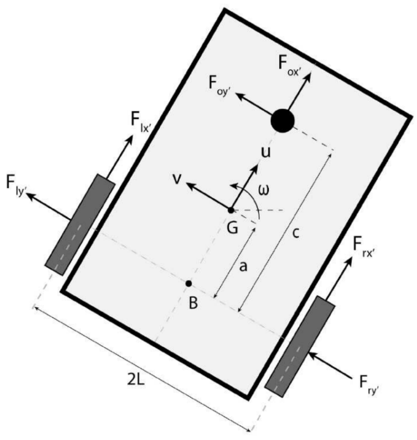


Figure 3.4: Duckiebot model schema.

Rearranging 3.2 and 3.3:

$$F_{ry'} + F_{ly'} = m(\dot{u} + v\omega) - F_{oy'} \tag{3.4}$$

$$I_z\dot{\omega} = (F_{rx'}L_1 - F_{lx'}L_2) - a(m(\dot{u} + v\omega) - F_{oy'}) + (c-a)F_{oy'} \tag{3.5}$$

We can then get an expression describing the rotational acceleration in terms of the traction forces:

$$\dot{\omega} = \frac{L_1}{I_z}F_{rx'} - \frac{L_2}{I_z} - \frac{am}{I_z}(\dot{v} + u\omega) + \frac{a}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} \tag{3.6}$$

The wheel radius r translates the rotational velocity ϕ to linear velocity in the point of contact. Now, given r nominal radius of the right and left wheels, ϕ_r and ϕ_l the right and left wheel velocities, u_r^s and u_l^s slip velocities of right and left wheel along the longitudinal axis and v^s the lateral slip velocity of both wheels:

$$u = \frac{1}{2}[r(\phi_r + \phi_l) + (u_r^s)] \tag{3.7}$$

$$\omega = \frac{1}{L_1 + L_2}[r(\phi_r + \phi_l) + (u_r^s - u_l^s)] \tag{3.8}$$

$$v = \frac{a}{L_1 + L_2}[r(\phi_r - \phi_l) + (u_r^s - u_l^s)] + v^s \tag{3.9}$$

Rearraning u and v we get a form of the extended kinematic formulation expressed as linear combination of the motor speeds:

$$\phi_r + \phi_l = \frac{2}{r}u - \frac{u_r^s + u_l^s}{r} \tag{3.10}$$

$$\phi_r - \phi_l = \frac{L_1 + L_2}{r}\omega - \frac{u_r^s - u_l^s}{r} \tag{3.11}$$

Lateral speed

From 3.10 and 3.11 we can compute the lateral speed as a relation between ω, v, longitudinal velocity, rotational speed and lateral slip.

$$v = a\omega + v^s \tag{3.12}$$

Assumption 5: The motors are considered to operate in steady-state mode, hence the effect of motor inductance is zero.

Generic DC motor model

$$\tau_r = \frac{k_a(V_r - k_b\phi_r)}{R_a} \tag{3.13}$$

$$\tau_l = \frac{k_a(V_l - k_b\phi_l)}{R_a} \tag{3.14}$$

With τ_r and τ_l being the right and left wheel motor torques, k_a motor constant, k_b voltage constant, R_a armature resistance of the motors, V_r and V_l are voltages applied to the right and left motors.

Let us write force-balance equations at the contact point for each wheel. Note that the two forces are acting at the contact point: (i) motor force, (ii) traction force. The traction force is the force exerted by the ground on the wheels and can also be thought of as grip. Then one can write the equations of motion for each wheel as:

$$I_e\dot{\phi}_r + B_e\phi_r = \tau_r - F_{rx'}r \tag{3.15}$$
$$I_e\dot{\phi}_l + B_e\phi_l = \tau_l - F_{lx'}r \tag{3.16}$$

Rearranging the equations:

$$F_{rx'}r = \tau_r - I_e\dot{\phi}_r - B_e\phi_r \tag{3.17}$$
$$F_{lx'}r = \tau_l - I_e\dot{\phi}_l - B_e\phi_l \tag{3.18}$$

Substituting in DC motor model 3.13 and 3.14 we get:

$$F_{rx'} = \frac{k_a(V_r - k_b\phi_r)}{R_a} - \frac{I_e}{r}\dot{\phi}_r - \frac{B_e}{r}\phi_r \tag{3.19}$$
$$F_{lx'} = \frac{k_a(V_l - k_b\phi_l)}{R_a} - \frac{I_e}{r}\dot{\phi}_l - \frac{B_e}{r}\phi_l \tag{3.20}$$

Rotational dynamics

It is demonstrated that the rotational dynamics can be expressed as:

$$\dot{\omega} = f(u,\omega,\chi) + B\begin{bmatrix} V_r \\ V_l \end{bmatrix}^T + \Delta_\omega \tag{3.21}$$

Where χ is a set of model parameters, B is a 2x2 input matric and Δ_ω is the part of the dynamics that will not be modeled and may be interpreted as addittive disturbance.

Substituting 3.19 and 3.20 in 3.9 we get:

$$\dot{\omega} = \frac{L_1}{I_z}(\frac{k_a(V_r - k_b\phi_r)}{R_a r} - \frac{I_e}{r}\dot{\phi}_r - \frac{B_e}{r}\dot{\phi}_r) - \frac{L_2}{I_z}(\frac{k_a(V_l - k_b\phi_l)}{R_a r} - \frac{I_e}{r}\dot{\phi}_l - \frac{B_e}{r}\dot{\phi}_l + \tag{3.22}$$
$$-\frac{am}{I_z}(\dot{v} + u\omega) + \frac{a}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} \tag{3.23}$$

Rearranging the terms to be more explicit:

$$\dot{\omega} = (\frac{L_1 k_a}{I_z R_a r})V_r - (\frac{L_1 k_a k_b}{I_z R_a r} + \frac{L_1 B_e}{I_z r})\phi_r - (\frac{L_1 I_e}{I_z R_a r})\dot{\phi_r} + \tag{3.24}$$

$$-(\frac{L_2 k_a}{I_z R_a r})\dot{V_l} + (\frac{L_2 k_a k_b}{I_z R_a r} + \frac{L_2 B_e}{I_z r})\phi_l - (\frac{L_2 I_e}{I_z r})\dot{\phi_l} + \tag{3.25}$$

$$-\frac{am}{I_z}(\dot{v} + u\omega) + \frac{a}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} \tag{3.26}$$

For clearer notation let's define:

$$a_r^\omega \triangleq \frac{L_1 k_a}{I_z R_a r} \tag{3.27}$$

$$a_l^\omega \triangleq \frac{L_2 k_a k_b}{I_z R_a r} \tag{3.28}$$

$$\beta_r^\omega \triangleq \frac{L_1 k_a k_b}{I_z R_a r} + \frac{L_1 B_e}{I_z r} \tag{3.29}$$

$$\beta_l^\omega \triangleq \frac{L_2 k_a k_b}{I_z R_a r} + \frac{L_2 B_e}{I_z r} \tag{3.30}$$

We can then rewrite the first equation as:

$$\dot{\omega} = a_r^\omega V_r - \beta_r^\omega \phi_r - a_l^\omega \dot{V_l} + \beta_l^\omega \phi_l - (\frac{L_2 I_e}{I_z r})\dot{\phi_l} + -\frac{am}{I_z}(\dot{v} + u\omega) + \frac{a}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} \tag{3.31}$$

Then let Δ_ω be defined as

$$\Delta_\omega \triangleq -\frac{L_1 I_e}{I_z r}\dot{\phi_r} + \frac{L_2 I_e}{I_z r}\dot{\phi_l} - \frac{am}{I_z}\dot{v} + \frac{am}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} \tag{3.32}$$

Now 3.31 rewritten with Δ_ω:

$$\dot{\Delta_\omega} = \alpha_r^\omega V_r - \alpha_l^\omega V_l - (\beta_n^\omega \frac{L_1 + L_2}{r})\omega - (\beta_\sigma^\omega \frac{2}{r})u - \frac{am}{I_z}u\omega + \Delta_\omega \tag{3.33}$$

Parameters β_r and β_l can alternatively be represented as:

$$\beta_r^\omega \triangleq \beta_n^\omega + \beta_\sigma^\omega \tag{3.34}$$

$$\beta_l^\omega \triangleq \beta_n^\omega - \beta_\sigma^\omega \tag{3.35}$$

We can then rewrite:

$$\dot{\Delta_\omega} = \alpha_r^\omega V_r - \alpha_l^\omega V_l - \beta_n^\omega(\phi_r - \phi_l) - \beta_\sigma^\omega(\phi_r + \phi_l) - (\frac{am}{I_z})u\omega + \Delta_\omega \tag{3.36}$$

Now given 3.10 and 3.11:

$$\dot{\Delta}_\omega = \alpha_r^\omega V_r - \alpha_l^\omega V_l - \beta_n^\omega(\frac{L_1 + L_2}{r}\omega - \frac{u_r^s - u_l^s}{r}) - \beta_\sigma^\omega(\frac{2}{r}u - \frac{u_r^s + u_l^s}{r}) - (\frac{am}{I_z})u\omega + \Delta_\omega \tag{3.37}$$

Interpreting slip velocities u_r^s and u_l^s and the lateral accelerations as disturbances, and updating the definition of Δ_ω yields:

Rotational disturbace:

$$\Delta_\omega \triangleq -\frac{I_e}{r}\dot{\phi}_r + \frac{I_e}{r}\dot{\phi}_l - \frac{am}{I_z}\dot{v} + \frac{a}{I_z}F_{oy'} + \frac{c-a}{I_z}F_{oy'} + \beta_n^\omega(\frac{u_r^s - u_l^s}{r}) + \beta_\sigma^\omega(\frac{u_r^s - u_l^s}{r}) - \frac{am}{I_z}\dot{v} \tag{3.38}$$

Rotational acceleration:

$$\dot{\omega} = \alpha_r^\omega V_r - \alpha_l^\omega V_l - \beta_n^\omega(\frac{L_1 + L_2}{r}\omega) - \beta_\sigma^\omega(\frac{2}{r}u) - (\frac{am}{I_z})u\omega + \Delta_\omega \tag{3.39}$$

Longitudinal dynamics

$$\dot{u} = f(u, \omega, \chi) + B \begin{bmatrix} V_r \\ V_l \end{bmatrix}^T + \Delta_u \tag{3.40}$$

Considering the equation of motion:

$$\sum F_{x'} = m(\dot{u} - v\omega) = F_{rx'} + F_{lx'} + F_{ox'} \tag{3.41}$$

Rearraning:

$$\dot{u} = \frac{F_{rx'}}{m} + \frac{F_{lx'}}{m} + \frac{F_{ox'}}{m} + v\omega \tag{3.42}$$

Substituting 3.42 in 3.19 and 3.20:

$$\dot{u} = \frac{1}{m}(\frac{k_a(V_r - k_b\phi_r)}{R_a r} - \frac{I_e}{r}\dot{\phi}_r - \frac{B_e}{r}\phi_r) + \frac{1}{m}(\frac{k_a(V_l - k_b\phi_l)}{R_a r} - \frac{I_e}{r}\dot{\phi}_l - \frac{B_e}{r}\phi_l) + \frac{F_{ox'}}{m} + v\omega \tag{3.43}$$

Rearranging in terms of input voltages:

$$\dot{u} \triangleq (\frac{k_a}{mR_ar})V_r - (\frac{K_aK_b - R_aB_e}{mR_ar})\phi_r - (\frac{I_e}{mr})\dot{\phi_r} + \tag{3.44}$$

$$+(\frac{k_a}{mR_ar})V_l - (\frac{k_ak_b - R_aB_e}{mR_ar}) - (\frac{I_e}{mr})\dot{\phi_l} + \frac{F_{ox'}}{m} + v\omega \tag{3.45}$$

Let Δ_u be considered as disturbance:

$$\dot{u} \triangleq -(\frac{I_e}{mr})\dot{\phi_r} - (\frac{I_e}{mr})\dot{\phi_l} + \frac{F_{ox'}}{m} \tag{3.46}$$

For clearer notation let's define:

$$a_r^u \triangleq \frac{k_a}{mR_ar} \tag{3.47}$$

$$a_l^u \triangleq \frac{k_a}{mR_ar} \tag{3.48}$$

$$\beta_r^u \triangleq \frac{k_ak_b - R_aB_e}{mR_ar} \tag{3.49}$$

$$\beta_l^u \triangleq \frac{k_ak_b - R_aB_e}{mR_ar} \tag{3.50}$$

Thus:

$$\dot{u} = a_r^uV_r - \beta_r^u\phi_r + a_l^uV_l - B_l^u\phi_l + v\omega + \Delta_u \tag{3.51}$$

Then defining:

$$\beta_n^u \triangleq \beta_r^u + \beta_l^u \tag{3.52}$$

$$\beta_\sigma^u \triangleq \beta_r^u - \beta_l^u \tag{3.53}$$

We get:

$$\dot{u} = a_r^uV_r - a_l^uV_l - \beta_n^u(\phi_r + \phi_l) - \beta_\sigma^u(\phi_r - \phi_l) + v\omega + \Delta_u \tag{3.54}$$

Substituting 3.11, 3.12 and 3.8 in 3.54:

$$\dot{u} = a_r^uV_r - a_l^uV_l - \beta_n^u(\frac{2}{r}u + \frac{u_r^s + u_l^s}{r}\omega) - \beta_\sigma^u(\frac{L_1 + L_2}{r}\omega - \frac{u_r^s - u_l^s}{r}) + (a\omega + v^s)\omega + \Delta_u \tag{3.55}$$

Longitudinal disturbance

$$\Delta_u = -(\frac{I_e}{mr})\dot{\phi_r} - (\frac{I_e}{mr})\dot{\phi_l} + \frac{F_{ox'}}{m} + \beta_n^u(\frac{u_r^s + u_l^s}{r}) + \beta_\sigma^u(\frac{u_r^s - u_l^s}{r}) + v^s\omega \tag{3.56}$$

Longitudinal acceleration

$$\dot{u} = a_r^u V_r + a_l^u V_l - (\beta_n^u \frac{2}{r})u - (\beta_\sigma^u \frac{L_1 + L_2}{r})\omega + a\omega^2 + \Delta_u \tag{3.57}$$

Final compact model:

$$\begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -(\beta_n^u \frac{2}{r})u - (\beta_\sigma^u \frac{L_1+L_2}{r})\omega + a\omega^2 \\ -(\beta_n^\omega \frac{L_1+L_2}{r})\omega - (\beta_\sigma^\omega \frac{2}{r})u - (\frac{am}{I_z})u\omega \end{bmatrix} + \begin{bmatrix} a_r^u & a_l^u \\ a_r^\omega & a_l^\omega \end{bmatrix} \begin{bmatrix} V_r \\ V_l \end{bmatrix} + \begin{bmatrix} \Delta_u \\ \Delta_\omega \end{bmatrix} \tag{3.58}$$

Discretization:

$$u_{k+1} = u_k + \dot{u}(u_k, \omega_k, V_r.V_l) \tag{3.59}$$

$$\omega_{k+1} = \omega_k + \dot{\omega}(u_k, \omega_k, V_r.V_l) \tag{3.60}$$

Assuming constant linear velocity v_k and constant angular velocity ω_k in $[t_k, t_{k+1}]$ we can use 2nd order Runge-Kutta integration method to compute the robot odometry.

$$x(t) = \frac{1}{2}\int_0^t (V_R(t') + V_L(t'))\cos(\theta(t'))\,dt' \tag{3.61}$$

$$y(t) = \frac{1}{2}\int_0^t (V_R(t') + V_L(t'))\sin(\theta(t'))\,dt' \tag{3.62}$$

$$\theta(t) = \frac{1}{L}\int_0^t (V_R(t') - V_L(t'))\,dt' \tag{3.63}$$

becomes:

$$x(k+1) = x_k + v_k T_s \cos(\theta_k + \frac{\omega_k T_s}{2} \tag{3.64}$$

$$y(k+1) = y_k + v_k T_s \sin(\theta_k + \frac{\omega_k T_s}{2} \tag{3.65}$$

$$\theta(k+1) = \theta_k + \omega_k T_s \tag{3.66}$$

$$T_s = t_{k+1} - t_k \tag{3.67}$$

Thus, assuming to know the parameters, the discretized state of the Duckiebot is:

$$u_{k+1} = u_k + \dot{u}(u_k, \omega_k, V_r.V_l) \tag{3.68}$$

$$\omega_{k+1} = \omega_k + \dot{\omega}(u_k, \omega_k, V_r.V_l) \tag{3.69}$$

$$x(k+1) = x_k + v_k T_s \cos(\theta_k + \frac{\omega_k T_s}{2} \tag{3.70}$$

$$y(k+1) = y_k + v_k T_s \sin(\theta_k + \frac{\omega_k T_s}{2} \tag{3.71}$$

$$\theta(k+1) = \theta_k + \omega_k T_s \tag{3.72}$$

$$T_s = t_{k+1} - t_k \tag{3.73}$$

3.3. Duckietown connectivity

Duckietown is designed to be easily integrated with ROS. ROS (Robot Operating System) is an open-source software platform for building and operating robotic systems. It is designed to be modular and flexible, allowing developers to easily create and manage the various components of a robotic system, such as hardware drivers, sensors, actuators, and control algorithms. One of the key features of ROS is its support for distributed computing, which allows different parts of a robotic system to be distributed across multiple computers and devices, and to communicate and coordinate with each other over a network. This enables the creation of complex and scalable robotic systems that can take advantage of the processing power and resources of multiple devices. In the specific case of Duckietown ROS enables communication between the robots and a workstation. Unfortunately due to the limitations of ROS (addressed by ROS2) it is not possible to have a decentralized system, as the network requires every time a single master node that orchestrates the messaging services. In the scenario of Duckietown each robot requires to have its own master node to guarantee the speed of execution and messaging across the actuators and the sensors, thus preventing easy communication with the other devices. To address this issue, and at the same time decrease the messaging delay, we built a UDP multicast on top of ROS messaging standards, allowing nodes in different masters to communicate with each other, while keeping the advantages of using the standardized libraries of ROS.

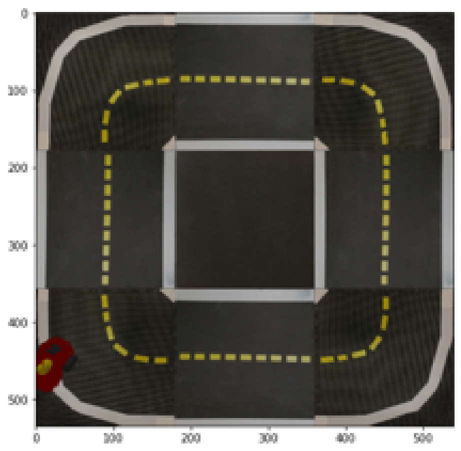
3.4. Duckietown gym setup

As previously mentioned Duckietown provides a python package for the Reinforcement Learning environment Gym, built by OpenAI and the de facto standard for reward-based simulations. The package "duckietown-gym" is open source and is available on pypi. [3] Our experiments does not require for the model to know the reward so the environment has been simplified by using directly the underlying Duckietown Simulator, disposing of the gym wrapper. Between the different configurations available in the environment we decided to focus on the most important for our experiments.

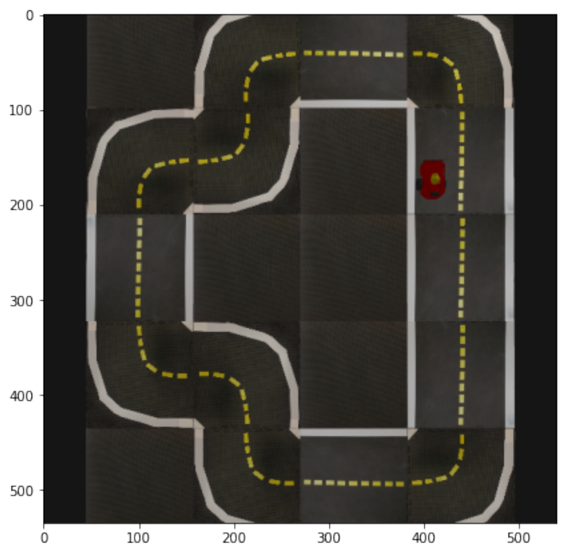
Listing 3.1: Duckietown gym setup

```
env = Simulator("Small-loop",
      full_transparency=True,
      domain_rand=False ,
      user_tile_start=[1,0],
      max_steps=float("inf")
      )
```

The first parameter is the map. There are several maps available, and there is the possibility to assemble one. We decided to focus on "Small-loop" and "ETH_large_loop", the first one being one of the simplest, the second is a short but challenging track for our experiments. Furthermore the second track is the one we used for the experiments in the real circuit. For this reason our physical track has been renamed Milano Duckar.



(a) Small-loop.



(b) ETH_large_loop/ Milano Duckar.

Figure 3.5: The two tracks of our experiments.

The parameter *full_transparency* we get access to the full information available in the environment that would otherwise be hidden to prevent the model from having complete

knowledge of the world. Our experiments do not focus on the comprehension of the environment so this parameter can be set to True.
domain_rand creates random color changes around the environment making it harder to extract the features using simple color filters. This was not a focus of our work so we set the parameter to False.

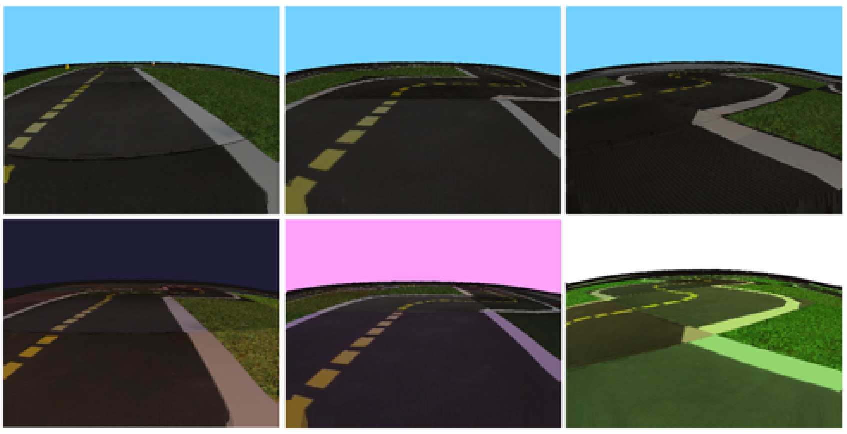


Figure 3.6: Environment with domain randomization from car point of view.

With fixed start_tile we guarantee consistency between starting poses in different trials.

Listing 3.2: Start tile definition

```
user_tile_start = [1,0]
env.unwrapped_start_pose = [[0.11699, 0, 0.41029], 0]
```

The *start_pose* matrix to find the best starting position has been defined by trial and error. The matrix is defined as $[[x, z, y], \theta]$, and those are the coordinates with respect to the tile chosen in *user_tile_start*. For example, tile [1, 0] is the second tile from the left, first from the top.
The last parameter, *max_step*, is the number of steps in a trial before it finishes. In our case, a trial should not have a defined end to it was set to infinite. A step is defined as a discretized moment during the simulation.

3.5. Watchtower

To accurately determine the position and orientation of the Duckiebot in the track during our experiments, we were faced with the challenge of obtaining reliable and precise location information. In a real-world scenario, this could potentially be achieved using GPS with RTK (real-time kinematic) technology, which is capable of providing high-accuracy positioning information. However, due to the small size of our model, it was not feasible to use GPS, and we had to find an alternative solution.

To address this challenge, we implemented a color extraction-based localization algorithm, which was based on the use of a Raspberry Pi computer equipped with a fisheye camera mounted to the ceiling of the track. We placed colored paper on the Duckiebot, and used the camera and the algorithm to track the Duckiebot's position and orientation based on the colors of the paper. This allowed us to obtain accurate and reliable location information, despite the lack of GPS.

We called this camera "watchtower", as it follows the original watchtower idea proposed by Duckietown, but following a very different implementation. In Duckietown several watchtowers connected to a high speed switch are placed around the track, thus distributing the load and enabling an accurate location of several duckiebots at the same time based on a QR placed on their top. In our scenario the number of robots to be localized is limited to two and the process had been simplified as previously described.

3.5.1. Watchtower camera calibration

To accurately extract the localization information for the Duckiebot from the fisheye camera, it was necessary to first calibrate and rectify the camera's image. This process, known as intrinsic calibration, involves the determination of the camera's intrinsic parameters, such as its focal length and principal point, in order to correct for distortion and other effects that can affect the accuracy of the image.

To calibrate the fisheye camera, we used a checkboard pattern and the calibration tools provided by Duckietown. This process, known as the "calibration dance," involves capturing a series of images of the checkboard pattern from different angles, and using these images to compute the intrinsic parameters of the camera.

Once the camera was calibrated and rectified, we were able to extract the localization information for the Duckiebot with a good degree of accuracy and reliability. As the robots move in a 2D surface passing from the points in the image to coordinate in a defined coordinate system was just a matter of a proportion between the dimension in pixel of the image and the measured length in meters of the track from point to point. We were able to automate this process knowing the distance between the two furthest points along x and along y in the track and extracting the central yellow line of the truck. A better overview of this process in the next section.

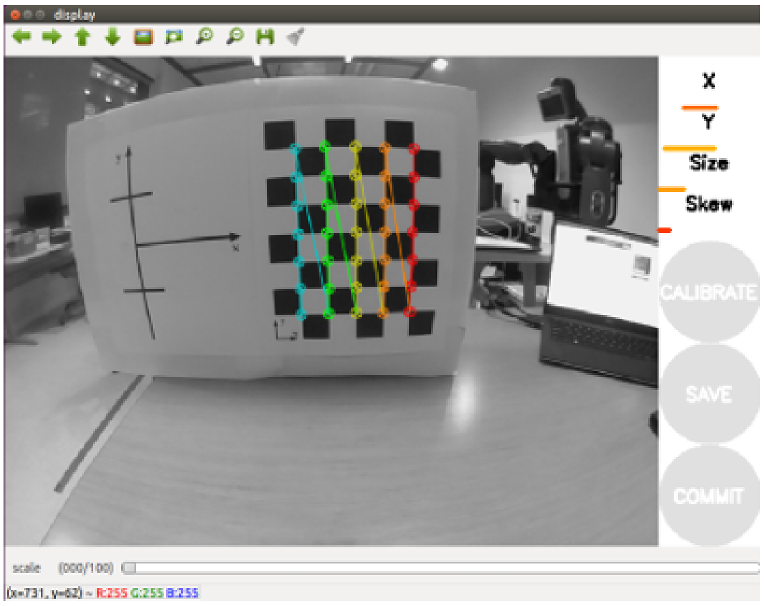


Figure 3.7: Calibration process

3.5.2. Central line extraction

The position of the central yellow lane is not known, I need to extract it. To extract the line this are the steps that have been followed, using as input the top view of the track:

1. Filter the image by color to extract the yellow.
2. Extract the points using Hough Lines point extraction.
3. Sort the points by angle or by distance. The first is easier but less reliable in complex tracks where a single angle can refer to multiple points along the track. To sort by angle the mean of the x and y coordinates have been computed as track center, then we considered the angle formed by each point to the center point.
4. Interpolation of the points by angle or distance. Another advantage of using distance is that the interpolation can be more precise as the sampled points can have the same distance between one and the other. On the other hand, it is very difficult to translate the distance from a starting point to a position on the track. The best interpolation is quadratic, with some smoothing.

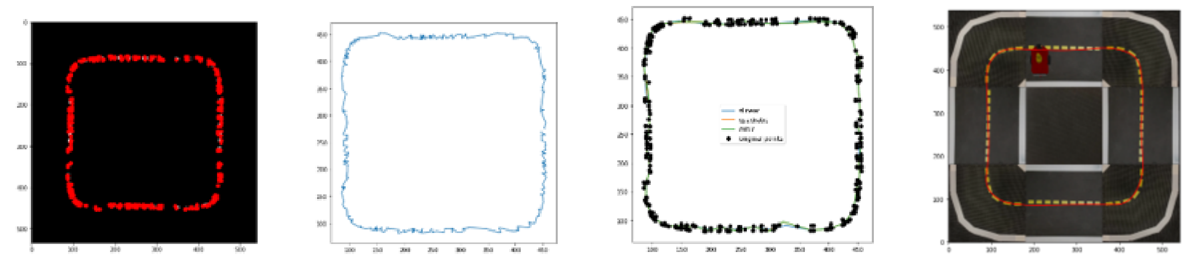

(a) Hough points extraction. (b) Sorted points. (c) Interpolated points. (d) Interpolated points on top of original image.

Figure 3.8: Steps followed in central line extraction.

3.5.3. Localization strategy

The localization in the real track is performed by a ROS node in the watchtower in almost real-time. To be able to extract the Duckiebot location we followed a pipeline of image preprocessing and final localization:

1. Image rectification.
2. Translation in the coordinate system.
3. Set resolution
4. Automatic white balancing
5. Map extraction
6. Car localization

Image rectification

The watchtower is around 2 meters far from the track and its camera is a fisheye, this means that there is a lot of distortion to take into account. To compute the correct parameters I used a Ros function provided by duckietown and a chessboard as previously described. Given the parameters the rectification is handled by OpenCV in the localization service.

Origin definition

To keep eveything easy the origin is defined where it usually is in the image on top right. Measuring the number of pixels in the image and knowing the dimension of the track a coordinate system was defined and gets computed by the localization process.

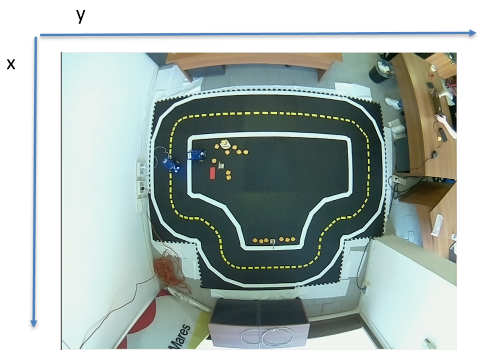


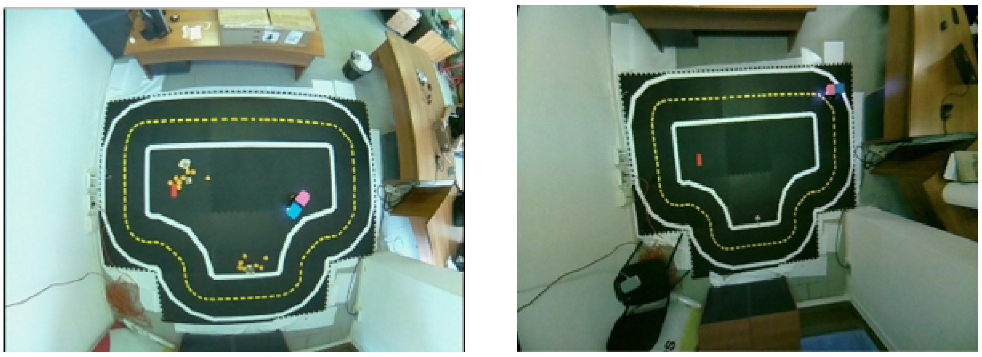
Figure 3.9: Coordinate system.

Image resolution

The original resolution of the image is too high to be handled in real time, the Raspberry was taking around 2 seconds to process the original 1296x972 pixel image. The localization process downscales the image to 324x243, enough to keep all the necessary features and be able to run the localization at 20Hz.

White balancing

The whole procedure assumes that the color stays constant but that is not true across the day. To guarantee robustness from the early lights of the day to the neon leds of the deep night an automatic white balancing algorithm based on the gray world assumption was implemented. Gray world assumption states that on average an image should be always gray, knowing how much it deviates from gray we are able to correct it.



(a) Original watchtower image. (b) Image rectified and white balanced.

Figure 3.10: Watchtower image processing.

Map extraction and car localization

Finally using the previously described procedure (3.5.2) the central line can be extracted and used as trajectory reference.
Given the preprocessed image and the duckiebot with a pink and blue card on top the car localization can be extracted in 5 steps:

1. Pink pixels extraction and computation of their mean with respect to x and y axis, point P is found. The pink color is very well defined and can be used as reference.
2. Draw a bounding box around the pink pixels.
3. Look for the blue pixels inside the bounding box and compute their mean, point B is found.
4. The mean between the two points P and B will be the car position.
5. Use P and B to compute the car orientation.

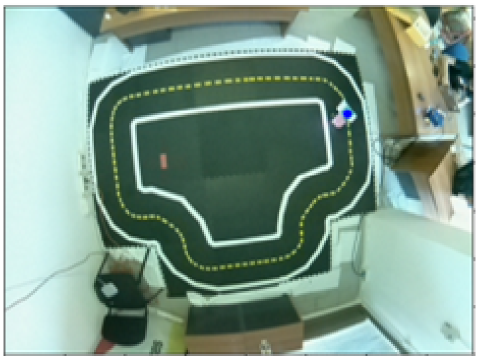


Figure 3.11: Extraction of color blue

3.5.4. What happens in the duckiebot?

The Duckiebot:

- Runs the controller at 10Hz.
- In the unlikely scenario where the new location message has not arrived from the watchtower in 0.1s between one iteration and the other it computes the new position in open loop using the model.
- Computes its linear and angular speed using the model having as input its old speed and position.

- The orientation from the watchtower sometimes can be very inaccurate, to address this issue the Duckiebot does not consider values that are too different from the last received orientation. In particular if the new orientation is more than 120° different from the old value the Duckiebot uses the model to compute the new angle.
- The map is computed by the watchtower only if needed and is stored in the Duckiebot for quick retrieval.

4 | Control Algorithm

In this chapter the two controllers will be presented and the steps from the MPC (Model Predictive Controller) to the LMPC (Learning Model Predictive Control) will be presented.

4.1. Trajectory following with MPC

The Model Predictive Control (MPC) is a widely utilized technique for advanced control that utilizes a mathematical model of the system being controlled and an optimization algorithm to determine the optimal control sequence. In our research, we employed the Casadi framework[2], an open-source collection of tools for numerical optimization that utilizes the technique of algorithmic differentiation (AD) to improve accuracy and speed. Additionally, we utilized the IPOPT[14] library, a set of primal-dual interior point optimization methods specifically designed for continuous systems, which proved to be the most efficient and stable option among the various line search methods we tested.
The idea, as presented in figure 4.1, is:

1. Use the central yellow line as trajectory, extracting it using the method described in chapter 3.5.2. From the line coordinates also the ideal orientation is extracted.
2. Use the MPC to compute the optimal input.
3. Give the input to the duckiebot.

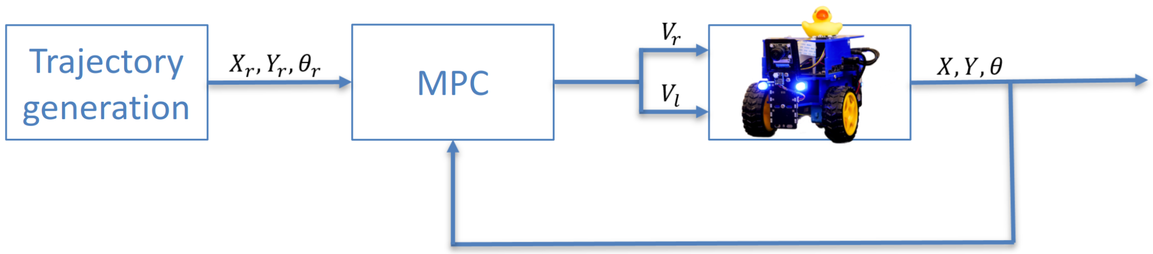


Figure 4.1: Control schema with MPC.

The general MPC definition is the following:

$$\min \quad J \tag{4.1}$$
$$\text{s.t.} \quad X_{k+1} = F(X_k, U_k) \tag{4.2}$$
$$-1 \leq U_k \leq 1 \tag{4.3}$$
$$X_0 = [x_0, y_0, \theta_0, v_0, \omega_0]^T \tag{4.4}$$

Where $U = [V_l, V_R]$, $X_k = [x_k, y_k, \theta_k, v_k, \omega_k]$. We will use p_{kr} and θ_{kr} for the reference point and angles.
In this sections all the experiments have been conducted on the simulator.

4.1.1. MPC without preview

In MPC without preview the reference is a constant point, thus $p_{kr} = p_r \forall k$. To maximize speed another parameter u_{max} is introduced. This is the formulation:

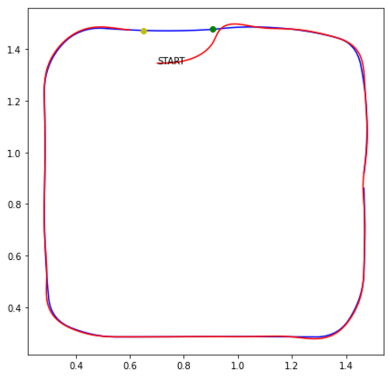
$$J = \sum_{k=0}^{N+1} \|p_k - p_r\|_{Q_1}^2 + \|\theta_k - \theta_r\|_{Q_2}^2 + \|u_k - u_{max}\|_{Q_3}^2 + \|U\|_R^2 \tag{4.5}$$

The formulation takes into account the reference for the orientation of the car. To compute the orientation the following approach was followed:

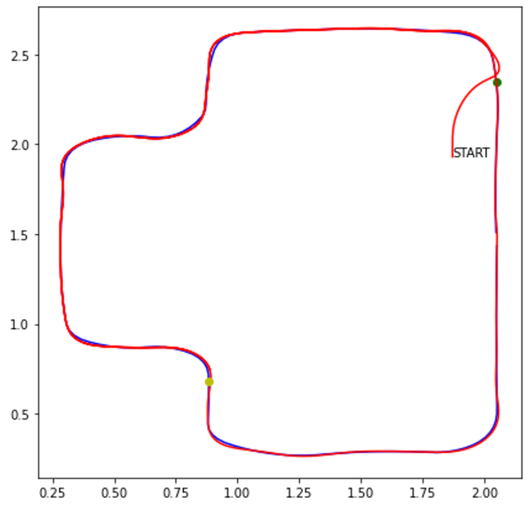
$$\theta_r = \arctan(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}) \quad \forall k = 0, ..., N-1 \tag{4.6}$$

$$\theta_N = \arctan(\frac{y_1 - y_N}{x_0 - x_N}) \tag{4.7}$$

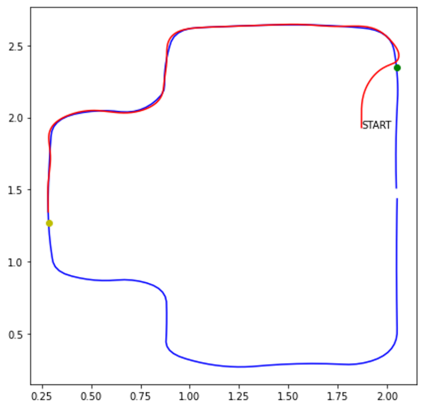
From an implementation prospective the next p_{kr} is taken when the position of the car is close enough to the reference point. This distance is a tunable parameter. As noticeable from Figure 4.6b the MPC without preview is very good at following straight trajectories but it is late to the curves. Also in figure e the distance from the reference is a sawtooth. This was easy to predict as the reference changes every time the car distance to the reference point is smaller than 2.5cm.



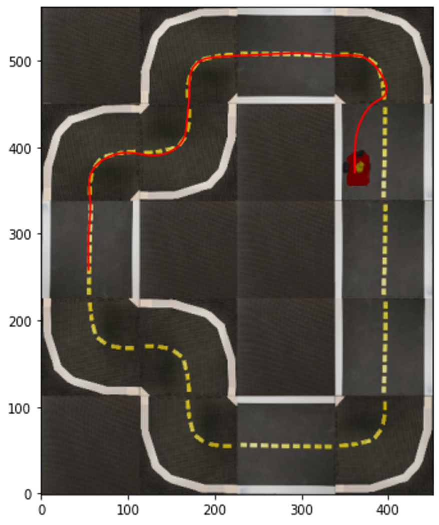
(a) Run on small circle.



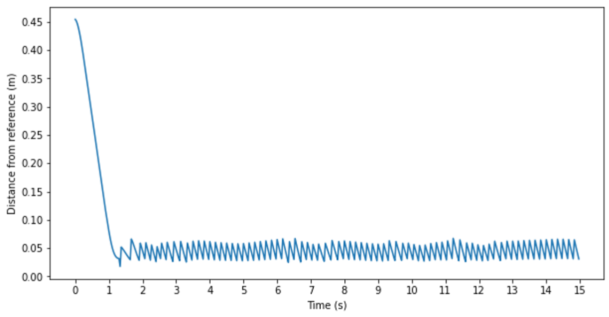
(b) Run on Milano Duckar.



(c) Milano Duckar, only 450 steps.



(d) Milano Duckar, bird view.



(e) Milano Duckar, distance from reference.

Figure 4.2: Analysis of MPC without preview.

4.1.2. MPC with preview

To overcome the limitations of the previous approach the MPC with preview is introduced. In this approach the reference for each point along the trajectory is computed. Finding the optimal number of references, given the frame rate and the duckiebot max speed is easy: $N_{references} = \frac{track\_length}{max\_speed * frame\_rate}$, where the frame rate is the speed of the MPC in Hertz. The frame rate has been constant at 10Hz for all the experiments. The MPC with preview formulation is then:

$$J = \sum_{k=0}^{N+1} \|p_k - p_{kr}\|^2_{Q_1} + \|\theta_k - \theta_{kr}\|^2_{Q_2} + \|u_k - u_{max}\|^2_{Q_3} + \|U\|^2_R \tag{4.8}$$

From a practical perspective at each step the closest point in the trajectory is found and its next N point are provided as reference for the MPC. To find the closest point quickly Voronoi regions are computed at the start of the process.

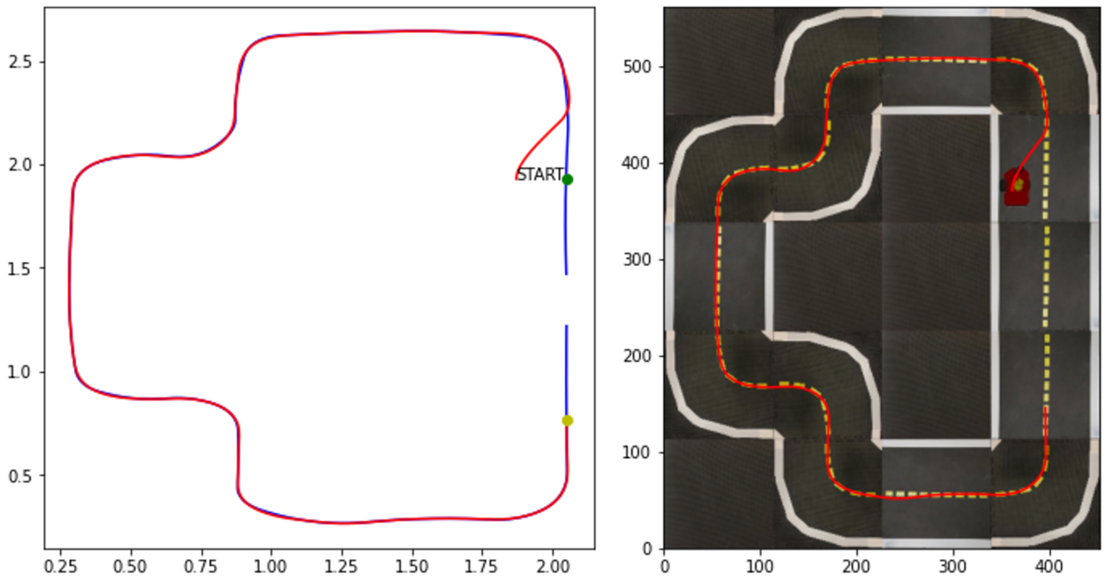


Figure 4.3: MPC with preview, Milano Duckar.

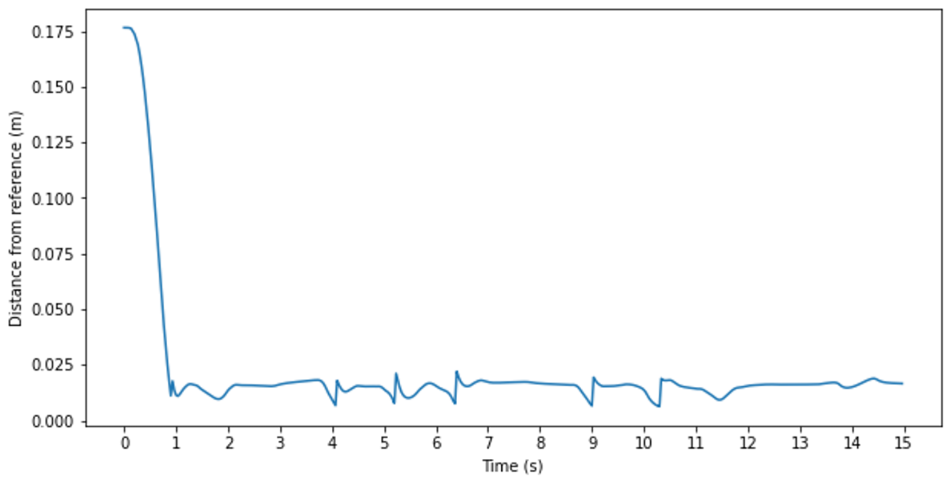


Figure 4.4: MPC with preview, distance from reference.

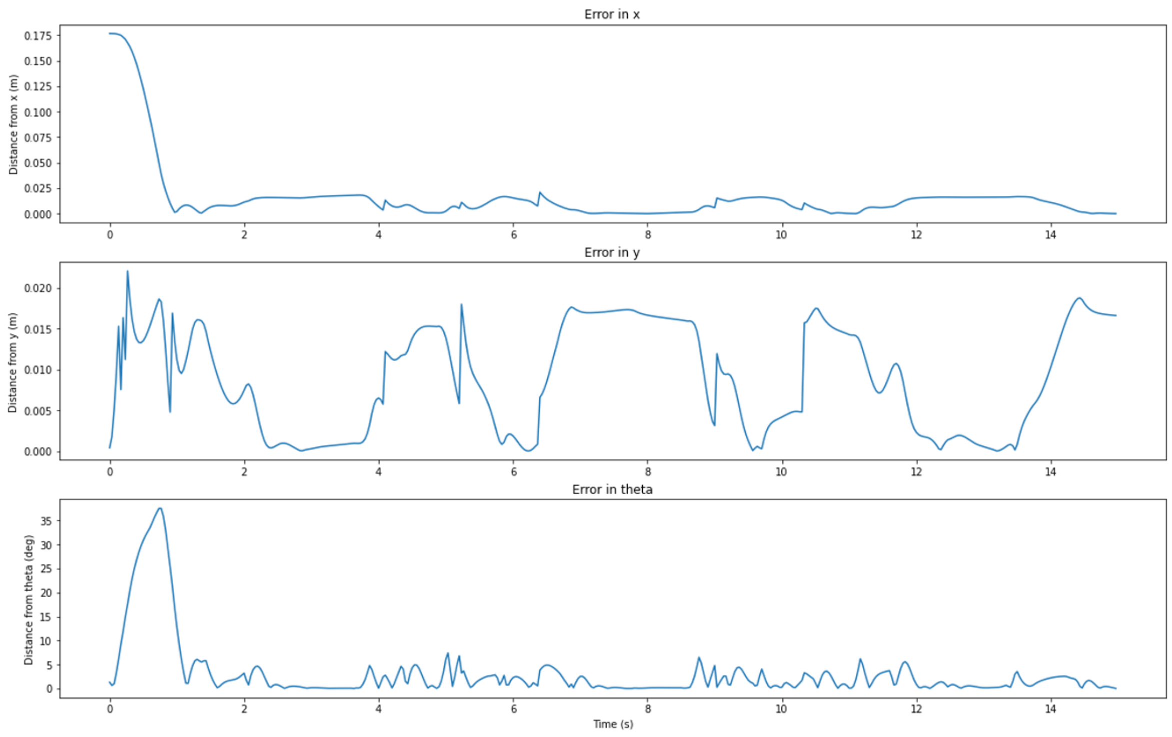
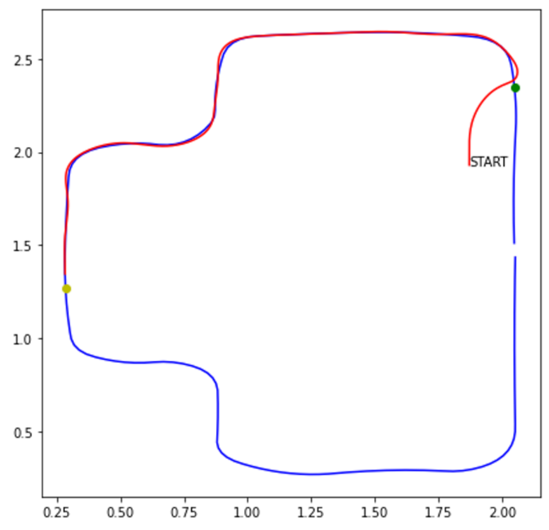
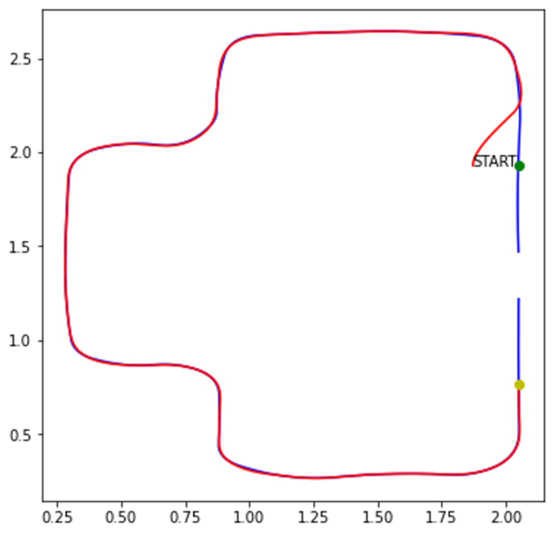


Figure 4.5: MPC with preview, errors across references. Note that road width = 0.47m.

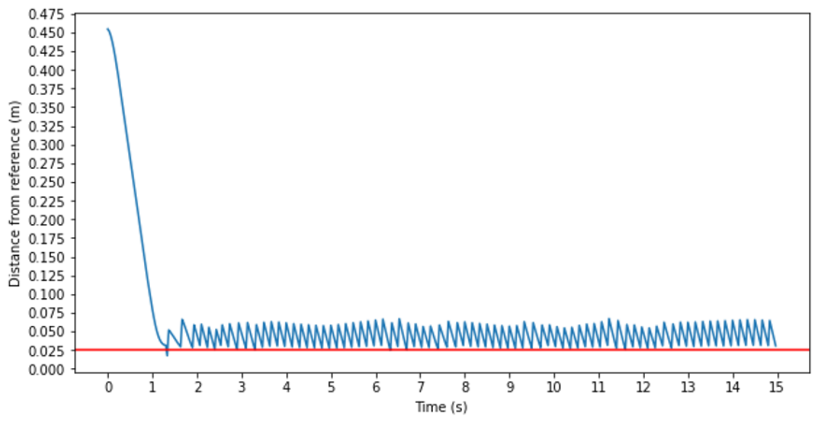
4.1.3. Comparison between without and with previews



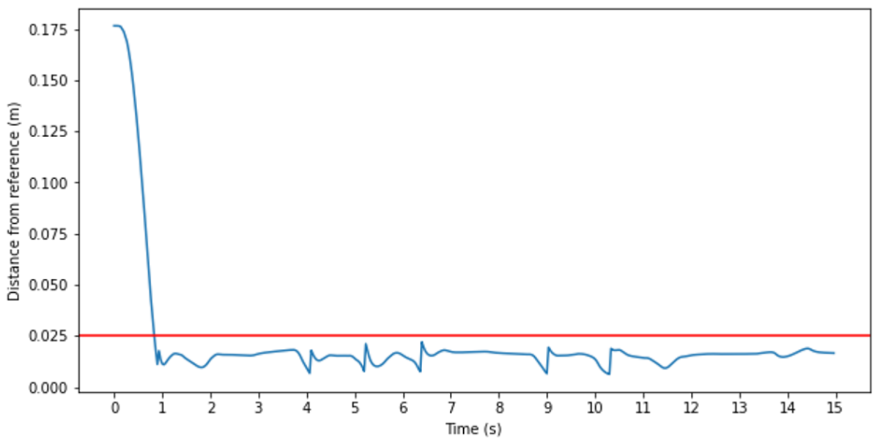
(a) No preview in 450 steps.



(b) Preview in 450 steps.



(c) No preview, distance from reference.



(d) Preview, distance from reference.

Figure 4.6: Comparison, note how in the same amount of time the mpc with preview covers almost double the distance of the same mpc without preview, achieving in the meantime a consistently better precision.

4.1.4. MPC with preview sensitivity analysis

Recalling that the MPC formulation is:

$$J = \sum_{k=0}^{N+1} \|p_k - p_{kr}\|_{Q_1}^2 + \|\theta_k - \theta_{kr}\|_{Q_2}^2 + \|u_k - u_{max}\|_{Q_3}^2 + \|U\|_R^2 \tag{4.9}$$

The tunable parameters are:

- N: prediction horizon. Higher means predictions of longer term but more computational time. N=5 is the best, higher does not provides significant improvements.
- Q_1: the weight on the distance from the reference in meters.
- Q_2: the weight on the distance from the angular reference in radians.
- Q_3: the weight on the difference between the current speed and the max speed.
- R: the weight on inputs.

To find the best combination of parameters R has been kept fixed to 1 and the ratio between the other parameters has been adjusted.

Q_1 vs Q_2

When Q_2 is big with respect to Q_1 the car stands in place or moves just to find the best orientation without caring about its position. But Q_2 should never be at 0 or in some starting condition the Duckiebot prefers to engage the rear gear and run the track backwards.

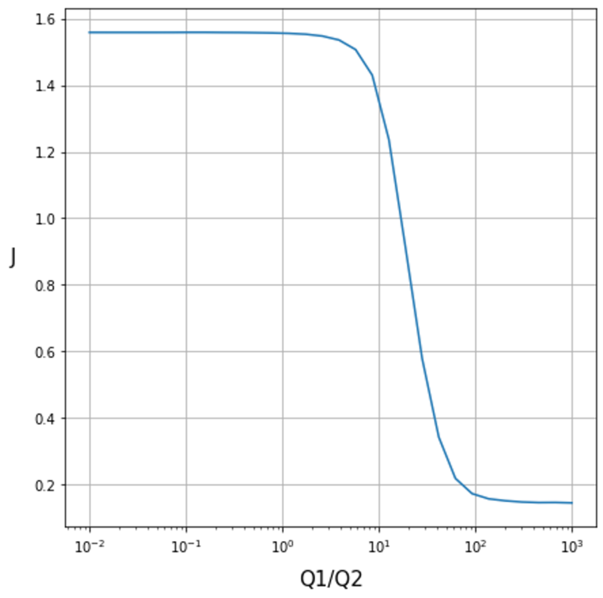


Figure 4.7: Sensitivity Q_1 vs Q_2

Q_1 vs Q_3

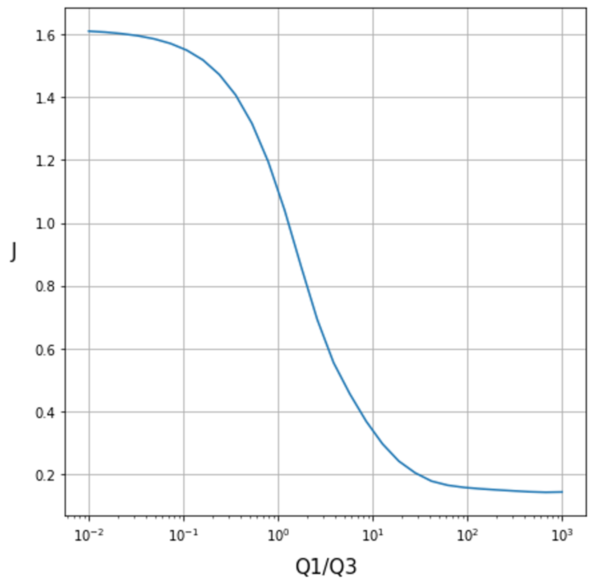


Figure 4.8: Sensitivity Q_1 vs Q_3

Q_2 vs Q_3

Q_3 should still be the smallest value as having too much weight on the speed means only focusing on going as fast as possible without caring about the reference.

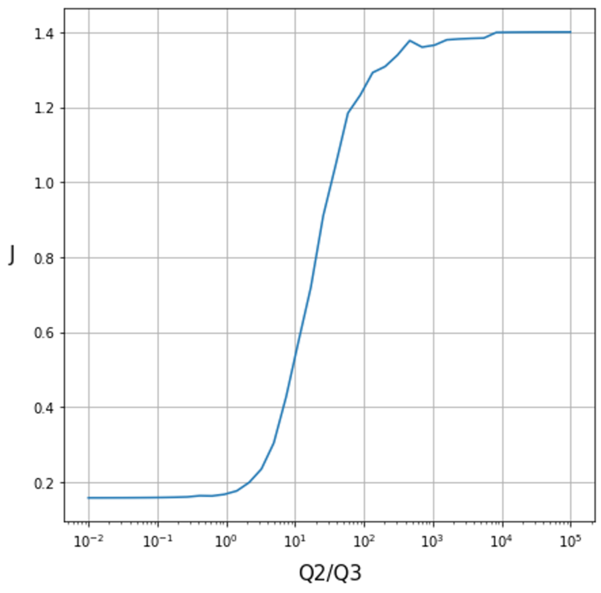


Figure 4.9: Sensitivity Q_2 vs Q_3

4.2. Trajectory planning and following with LMPC

The Learning Model Predictive Controller is a recently developed technique [12] based on iterative learning control. The idea is similar to the MPC but this time the reference is not given but computed in real time based on the latest iteration. The aim of the LMPC is to minimize a parameter learning the optimal trajectory, in our case it needs to minimize the time per loop.

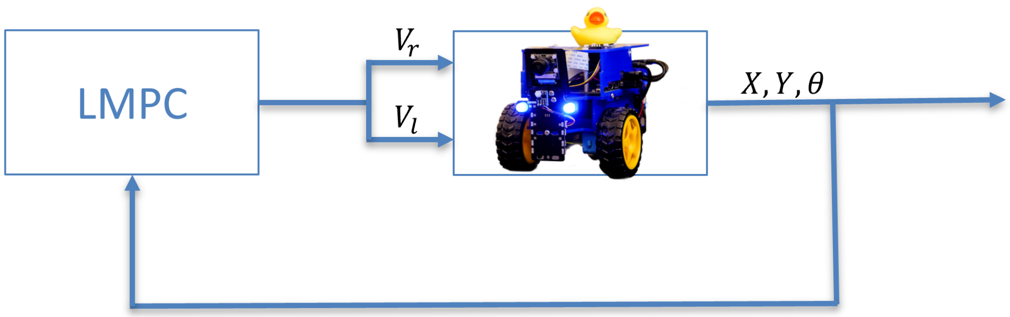


Figure 4.10: LMPC Schema

4.2.1. How the LMPC works

Follows a simple example to explain the principles behind the LMPC.

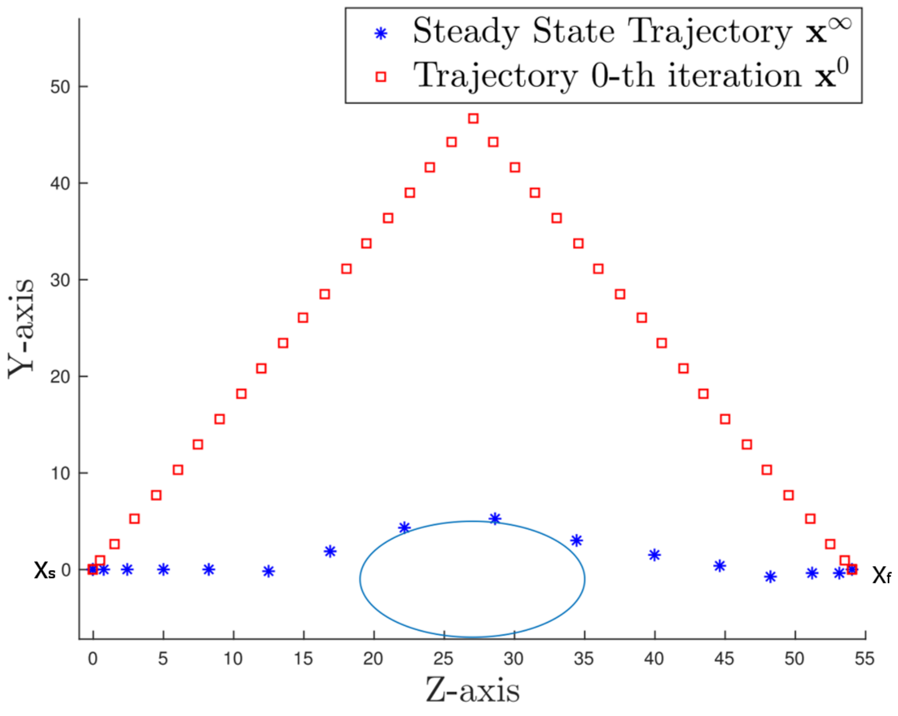


Figure 4.11: LMPC Example

Figure 4.12 shows an example of LMPC. The system needs to go from point X_s to X_f. The first step consists in finding a feasible trajectory using a simple method, like a brute force computation, in the image the trajectory x^0 is in red. The aim is to find the optimal trajectory to go to X_f avoiding the obstacle in the middle and minimizing the total time. Let's suppose that we want to find the whole trajectory in one shot, our horizon N will be ∞:

$$\min_u \quad \sum_{k=0}^{\infty} h(x_k) \tag{4.10}$$

$$s.t \quad x_{k+1} = f(x_k, u_k) \tag{4.11}$$

$$x_0 = x_s \tag{4.12}$$

$$h(x_k) = \begin{cases} 1, & x_k \neq x_F \\ 0, & x_k = x_F \end{cases} \tag{4.13}$$

$$\frac{(z_k - z_{obs})^2}{a_e^2} + \frac{(y_k - y_{obs})^2}{b_e^2} \leq 1 \tag{4.14}$$

The constraints 4.11 and 4.12 are the same that already encountered in the MPC. 4.13 becomes 0 when the system reaches X_f, minimizing $h(x)$ we are effectively telling the system to reach the final destination in the shortest amount of steps. 4.14 defines the

obstacle, the positions observed during the evolution of the system need to be always outside of the blue oval.
Now let's assume that we can't compute all the steps from X_s to X_f in one shot. We want to minimize one part at the time, from X_s to X_N, the constraints will be the same as before but the function to minimize will be:

$$\min_{u} \quad \sum_{k=0}^{N} h(x_k) \tag{4.15}$$

The challenge is choosing X_N to minimize not only the local travel time but also the global. To reach this goal several iterations are performed, and the total travel time is saved at each iteration. Once a trajectory total time is known each reached point in the newly explored trajectory is mapped to a time to arrive to X_F and saved in a set, the "safe set". In the iterations that follow the point X_N will be chosen between the previously explored points that are reachable form the current position and that have smaller time to arrive, de facto minimizing the overall cost.

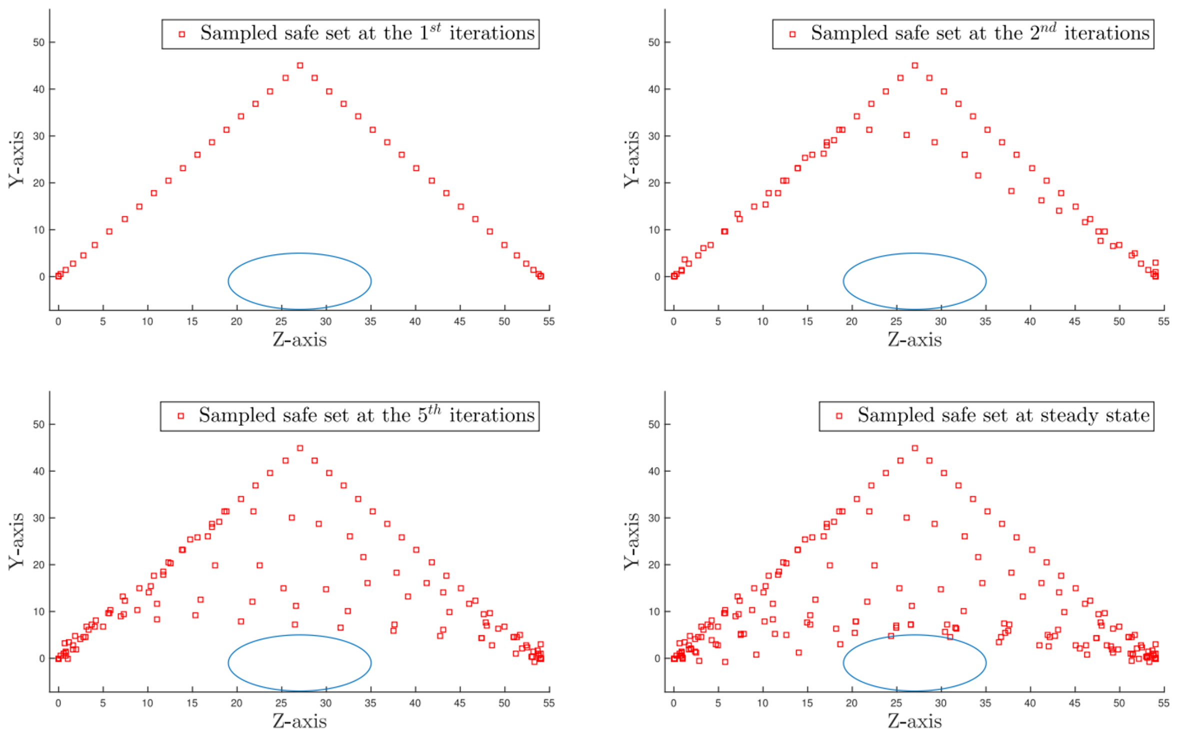


Figure 4.12: LMPC iterations, at every step it aims at a point N points in the future $X_{t+N|t}$ considering the previous fastest trajectory.

We can define a new equation that considers also the time to go in the form of a Q value, an action value function: the cost of taking an action in a state when following a policy is:

$$\min_{u} \quad \sum_{k=0}^{\infty} h(x_k) + Q^{j-1}(x_N) \tag{4.16}$$

$$s.t \quad x_{k+1} = f(x_k, u_k) \tag{4.17}$$

$$x_0 = x_s \tag{4.18}$$

$$h(x_k) = \begin{cases} 1, & x_k \neq x_F \\ 0, & x_k = x_F \end{cases} \tag{4.19}$$

$$\frac{(z_k - z_{obs})^2}{a_e^2} + \frac{(y_k - y_{obs})^2}{b_e^2} \leq 1 \tag{4.20}$$

$$x_N \in SS^{j-1} \tag{4.21}$$

The loss function 4.16 is very similar to the previous but takes into account also the action value function. 4.21 is the safe set of the iteration $j-1$, it contains points where the agent went in the previous iterations that are close to a point in time $t+N$. The definition of safe set is: "A non empty subset S of the vertices of a connected graph $G = (V(G), E(G))$ is a safe set if, for every connected component C of $G[S]$ and every connected component of D of $G-S$, we have $|C| \leq |D|$ whenever there exists an edge of G between C and D".

4.2.2. From LMPC to Local LMPC

In our scenario it can be difficult to find a reachable point in the cloud of collected trajectories, to overcome this limitation we followed an approach proposed by Rosolia and Borrelli in 2020[13]. The idea consists in creating a convex hull using the K nearest neighbours of x in the previous iteration, and using the LMPC to find the best point inside the hull.

Convex safe set

A *convex safe set* is defined as:

$$CS^j = Conv(\bigcup_{i=0}^{j}\bigcup_{t=0}^{\infty} x_t^i) \tag{4.22}$$

Remembering that a *convex combination* is defined as any $P \subseteq \mathbb{R}$ where:

$$Conv(P) = \{\sum_{i=1}^{n} \lambda_i p_i | n \subseteq \mathbb{N} \wedge \forall \mathrm{i} \subseteq \{\mathrm{i}, ..., \mathrm{N}\} : \lambda_\mathrm{i} \leq 0 \wedge \mathrm{p_i} \subseteq \mathrm{P}\} \tag{4.23}$$

Local convex Q function

Let's now define a *local convex Q function* around x as the convex combination of the cost associated with the stored trajectories:

$$Q_l^j(\overline{x}, x) = \min_{\lambda} J_l^i(x)\lambda \tag{4.24}$$

$$s.t \quad \lambda \leq 0, \mathbb{1}\lambda = 1, \mathrm{D}_\mathrm{l}^\mathrm{j}(\mathrm{x})\lambda = \overline{\mathrm{x}} \tag{4.25}$$

Where J_l^i collects the cost-to-go associated with the K-nearest neighbours to x from the l^{th} to the j^{th} iteration. The cost to go $J_{t\to T^j}^j(x_t^j) = T^j - t$ represents the time to drive the vehicle from x_t^j to the finish line along the j^{th} trajectory, computed at the end of the lap.

Local LMPC

Given this definition we can proceed to define the design of the *Local LMPC*:

$$J_{t\to t+N}^{LMPC,j}(x_t^j, z_t^j) = \min_{U_t^j, \lambda_t^j} \sum_{k=t}^{t+N-1} h(x_{k|t}^j) + J_l^{j-1}(z_t^j)\lambda_t^j \tag{4.26}$$

$$s.t. \quad x_{t|t}^j = x_t^j \tag{4.27}$$

$$\lambda \leq 0, \mathbb{1}\lambda = 1, \mathrm{D}_\mathrm{l}^{\mathrm{j}-1}(\mathrm{z}_\mathrm{t}^\mathrm{j})\lambda_\mathrm{t}^\mathrm{j} = \mathrm{x}_{\mathrm{t+N|t}}^\mathrm{j} \tag{4.28}$$

$$x_{k+1|t}^j = A_{k|t}^j x_{k|t}^j + B_{k|t}^j u_{k|t}^j + C_{k|t}^j \tag{4.29}$$

$$x_{k|t}^j \subseteq X, u_{k|t}^j \subseteq U \tag{4.30}$$

$$\forall k = t, ..., t+N-1 \tag{4.31}$$

With

$$h(x) = \begin{cases} 0 & \text{if } x \in x_f. \\ 1 & \text{else.} \end{cases} \tag{4.32}$$

Direct your attention on the convex constraint 4.28:

$$\lambda \leq 0, \mathbb{1}\lambda = 1, \mathrm{D}_\mathrm{l}^{\mathrm{j}-1}(\mathrm{z}_\mathrm{t}^\mathrm{j})\lambda_\mathrm{t}^\mathrm{j} = \mathrm{x}_{\mathrm{t+N|t}}^\mathrm{j} \tag{4.33}$$

$$z_t^j = \begin{cases} x_N^{j-1} & \text{if } t = 0. \\ S_l^j(z_{t-1}^j)\lambda_{t-1}^{j,*} & \text{else.} \end{cases} \tag{4.34}$$

The vector z_t^j represents a candidate terminal state for the planned trajectory of the LMPC at time t. S is the matrix which collects the evolution of the states stored in the columns of the matrix D:

$$S_l^j(x) = [x^l_{t_1^{l,*}+1}, ..., x^l_{t_k^{l,*}+1}, ..., x^j_{t_1^{j,*}+1}, ..., x^j_{t_k^{j,*}+1}] \tag{4.35}$$

4.2.3. LMPC implementation

In the original implementation of the author the vehicle has in its state the distance from the track border as a way to know its position and a simple constraint that the distance from the border should be bigger than 0. Achieving the same with Duckietown is not trivial as it requires a deep change in the simulator. To address this issue a new term 4.42 has been introduced in the constraints to take into account the track boundaries:

$$J^{LMPC,j}_{t\to t+N}(x_t^j, z_t^j) = \min_{U_t^j,\lambda_t^j} \sum_{k=t}^{t+N-1} h(x^j_{k|t}) + J_l^{j-1}(z_t^j)\lambda_t^j \tag{4.36}$$

$$s.t. \quad x^j_{t|t} = x^j_t \tag{4.37}$$

$$\lambda \leq 0, \mathbb{1}\lambda = 1, \mathrm{D}_\mathrm{l}^{\mathrm{j}-1}(\mathrm{z}_\mathrm{t}^\mathrm{j})\lambda_\mathrm{t}^\mathrm{j} = \mathrm{x}^\mathrm{j}_{\mathrm{t+N|t}} \tag{4.38}$$

$$x^j_{k+1|t} = A^j_{k|t}x^j_{k|t} + B^j_{k|t}u^j_{k|t} + C^j_{k|t} \tag{4.39}$$

$$x^j_{k|t} \subseteq X, u^j_{k|t} \subseteq U \tag{4.40}$$

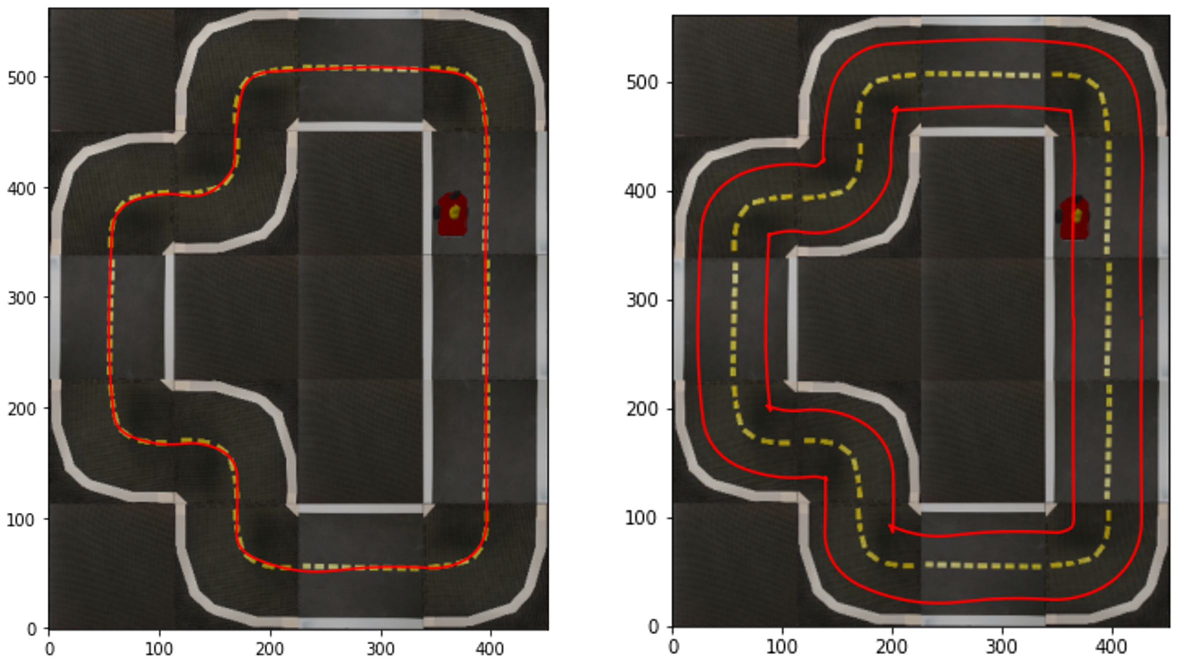
$$\forall k = t, ..., t+N-1 \tag{4.41}$$

$$\lambda_{SS} \leq 0, \mathbb{1}\lambda_\mathrm{SS} = 1, \mathrm{T}(\mathrm{z}_\mathrm{t}^\mathrm{j})\lambda_\mathrm{SS} = \mathrm{x} \tag{4.42}$$

Where $T(z_t^j)$ are a given number of points in the track border that are closer to the Duckiebot. This new convex hull forces the vehicle to stay inside the borders for the whole trajectory.

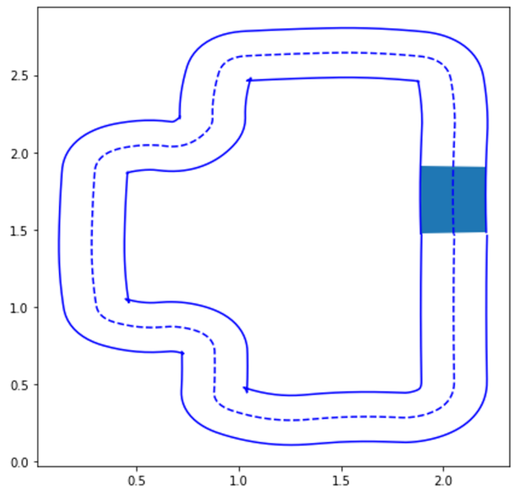
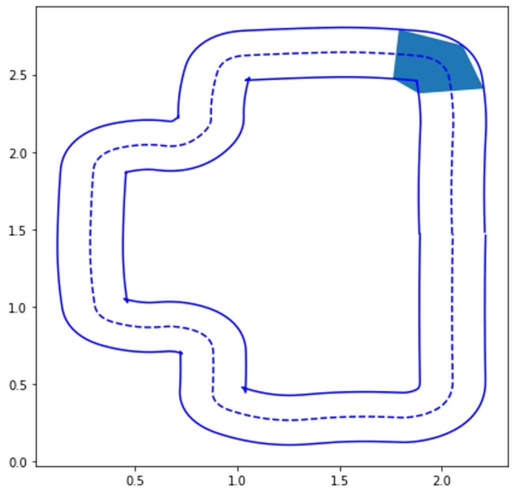
Border extraction

To extract the border we can proceed as seen in chapter 3.5.2 to find the central lane. Finding the borders will be only a matter of finding two points equally distant to each point along the central trajectory.



(a) Central lane in Milano Duckar. (b) Borders extracted considering a couple of cm for the car width.

Figure 4.13: From the central lane to the borders.

(a) Convex hull visualization on a straight road. (b) Convex hull visualization on a curve road.

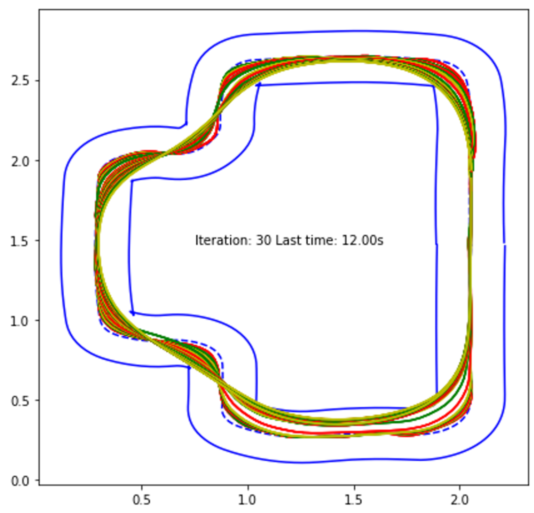
Figure 4.14: Convex hull visualization, note that 5 points are necessary as in the first curve the car tends to stay a bit wide in the first laps.

4.2.4. LMPC sensitivity analysis

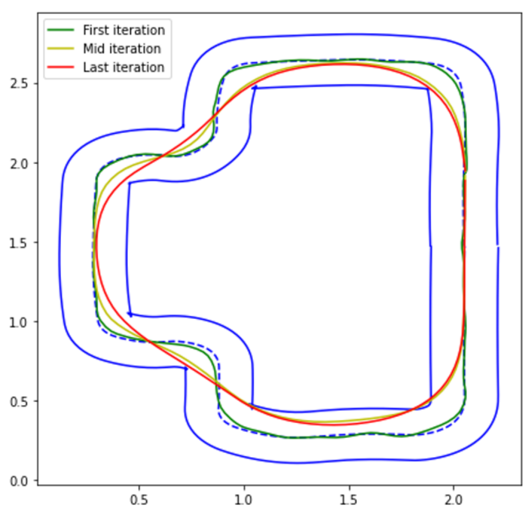
The following hyperparameters need to be tuned:

1. N (integer): The horizon, the bigger the N the faster the convergence, but computationally it gets very heavy already with N=10;
2. K (integer): The number of nearest neighbours to consider for the convex hull. Bigger K means more exploration, sometimes it leads to cutting too much the curves;
3. i_j (integer): The number of past iterations to consider in the nearest neighbour. Using more iterations means not losing memory of what happened in certain trajectories but also a much more complex convex hull to compute;
4. i_j_all (boolean): wheter or not to use all the past iterations for the nearest neighbour. This prevents memory loss but also widens the search for optimal points as the number of points to consider for nearest neighbor needs to be computed as $K * i\_j$ or the convex hull would become much smaller and focused on a fixed area in just a few iterations, losing all exploratory capabilities;
5. $Frame\_rate$ (integer): Default for Duckietown is 30Hz, 10Hz have been used in our experiments to guarantee a good accuracy of the model but also faster computational times.

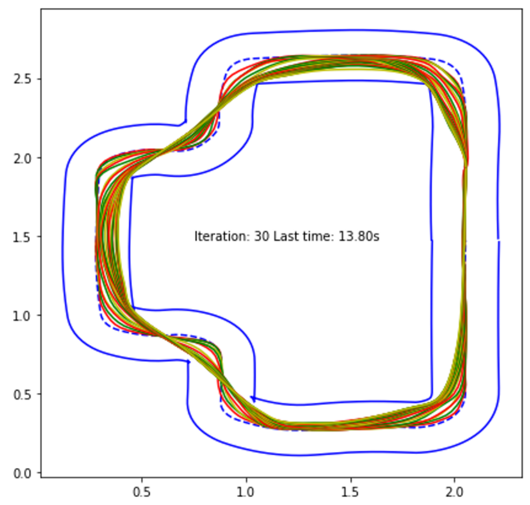
Sensitivity for N



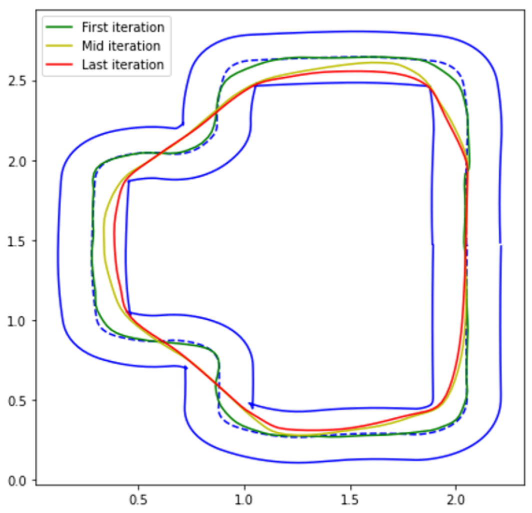
(a) With N=2 the average computation time per lap is 7.41s.



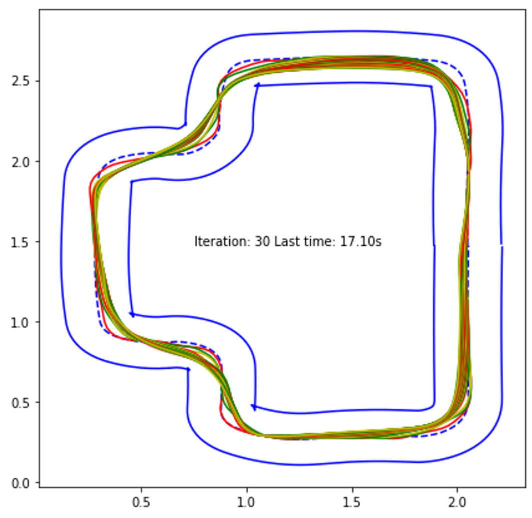
(b) Laps 1, 15 and 30 with N=2.



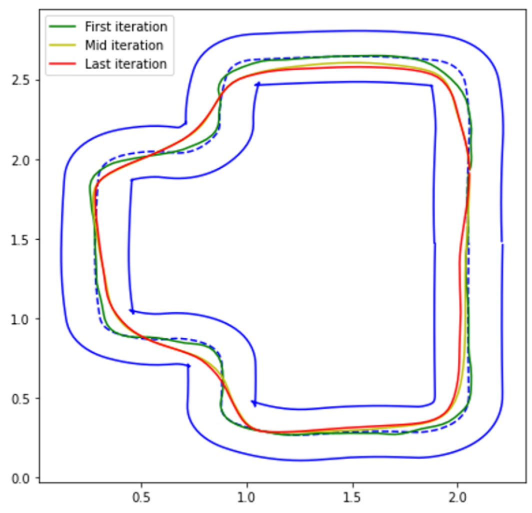
(c) With N=5 it is 12.71s.



(d) Laps 1, 15 and 30 with N=5.



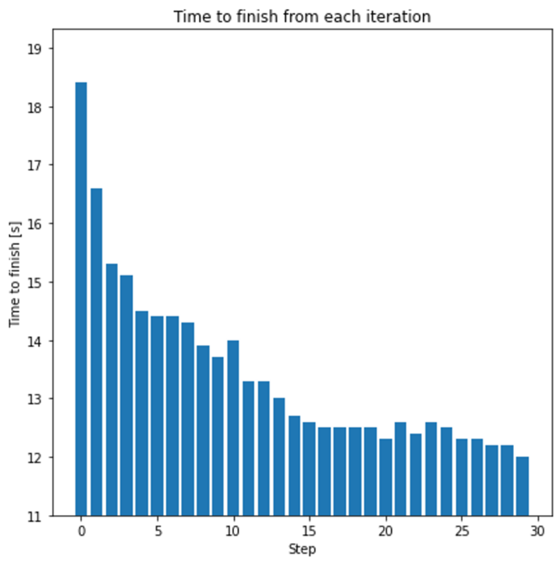
(e) With N=8 it is 19.23s.



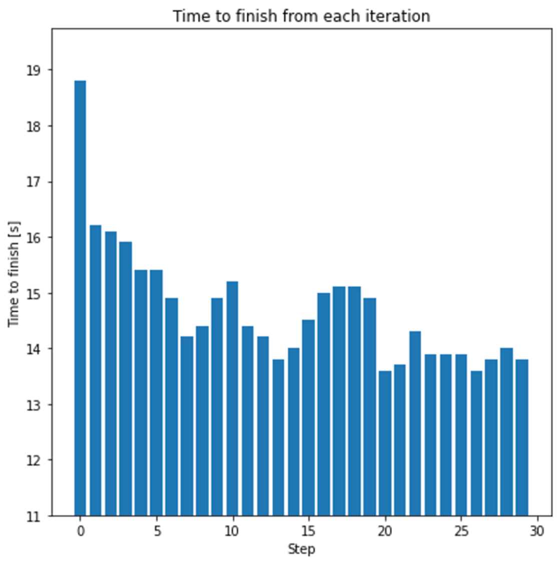
(f) Laps 1, 15 and 30 with N=8.

Figure 4.15: Experiments with N = 2, 5, 8. The best lap time is with N=2.

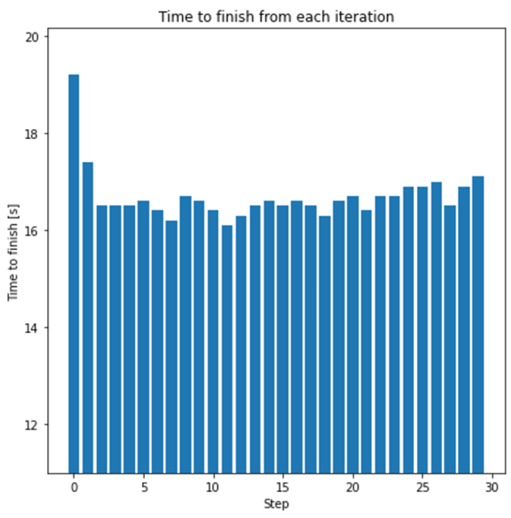
Why higher N leads to poor performance? Higher N means longer predictions that in most of the cases brings the car out of the constraints. To avoid this issue the controller becomes very conservative. This is visible also in the next plots with the times per lap, even with smaller N the car tends to close the trajectory too much and crashes into the boundaries, resulting in last moment maneuvers, smaller speed and longer lap times. This happened also in the original implementation by Rosolia.



(a) N=2.



(b) N=5.



(c) N=8.

Figure 4.16: Evolution of times per lap with different N.

As visible in Figure 4.15 and 4.16 with N=2 the convergence is steadier and more regular, reaches a better lap time and requires a smaller computational burden. Results with N=2 are very promising but not easy to reproduce, as visible in Figure 4.17. A slight overlap of the trajectory with the border is acceptable and was taken into account when computing the track border. This results when the LMPC computed reference is after a 90 degrees curve or when the discretization bring the LMPC to believe that all the points are inside the track. Furthermore it can happen that the computed trajectory leads to a position where the car has no alternatives other than going outside the track. This can be solved by allowing the car to go backwards but given that the objective is minimizing the lap time this approach is counter efficient. Solutions for this issue will be further explored in the analyses of the other parameters.

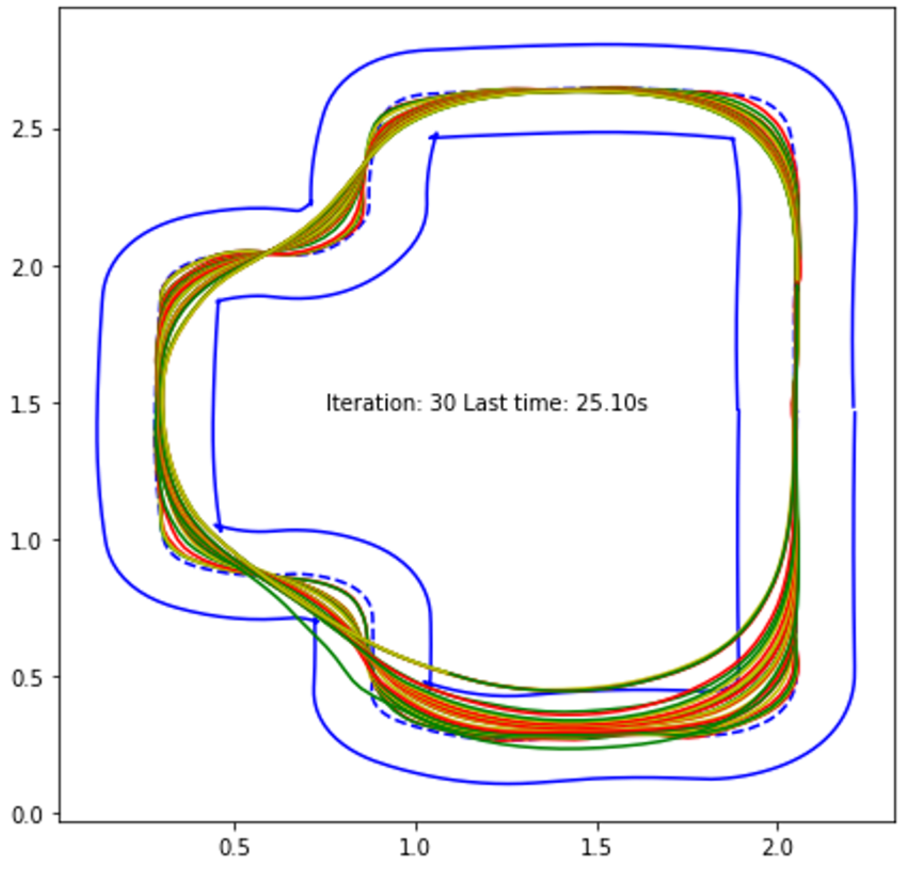
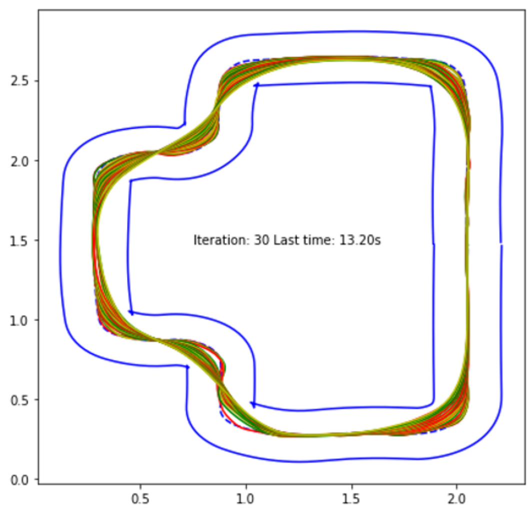
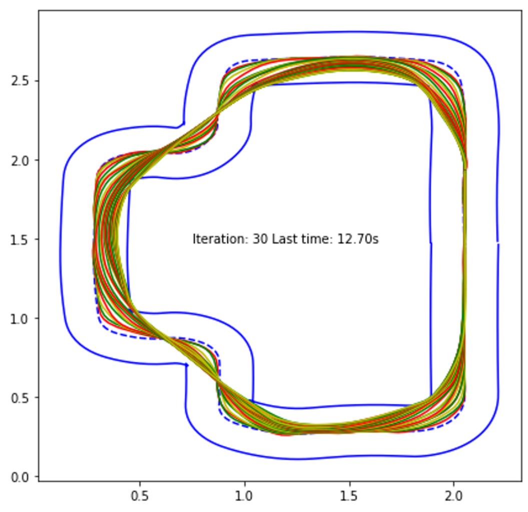


Figure 4.17: N=2 and K=8, note the lower part of the trajectory

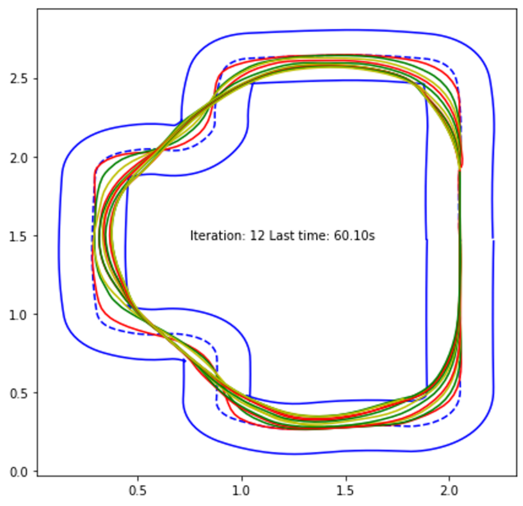
Sensitivity for K



(a) With K=5 the average computational time per lap is 10.85s.



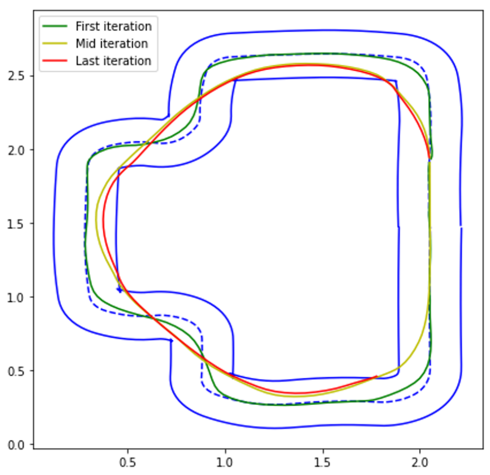
(b) With K=8 the average computational time per lap is 9.361s.



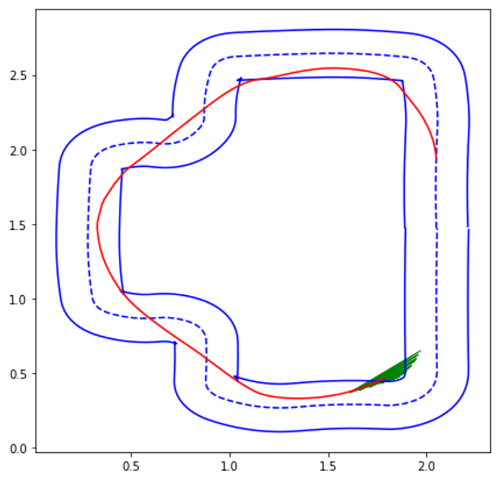
(c) With K=10 it is 9.38s but it gets stuck after only 12 laps.

Figure 4.18: Experiments with K = 5, 8, 10.

As seen in figure 4.18 with higher K the exploration is more rapid but it gets stuck faster.



(a) First, mid and last iteration with K=10.



(b) The safe set when it can't find a solution.

Figure 4.19: More on K = 10.

Higher K means that the car can see a solution that is much more distant than before, resulting in curves getting cut at the point where it can not find any feasible solution because the safe set does not include any point inside the track. A simple workaround for this issue is to use boundaries instead of constraints, allowing the car to go outside the borders, but this results in the car consistently breaking the boundaries.

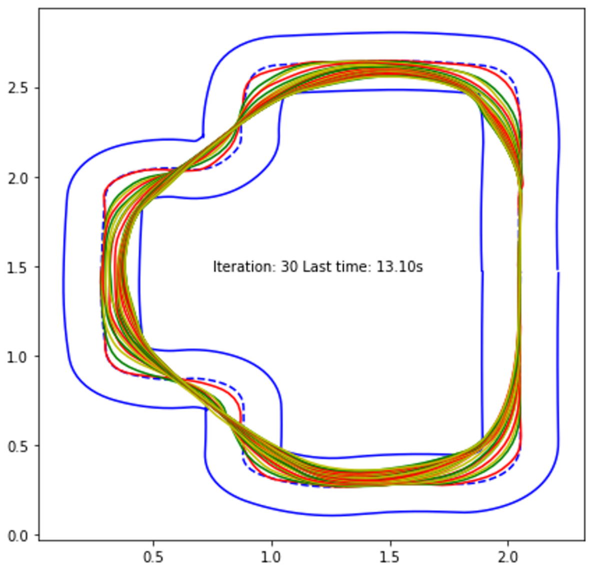
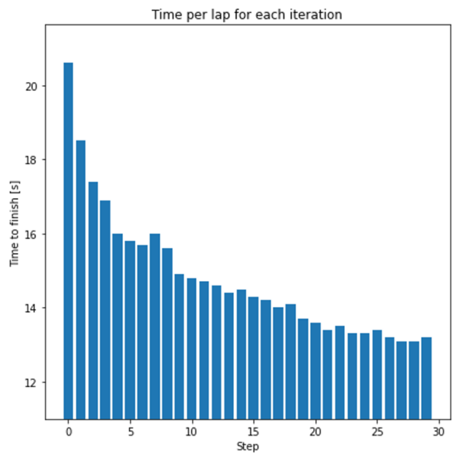
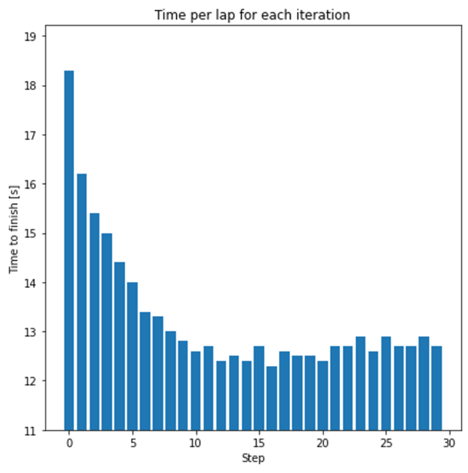


Figure 4.20: K=10 and i_j=3

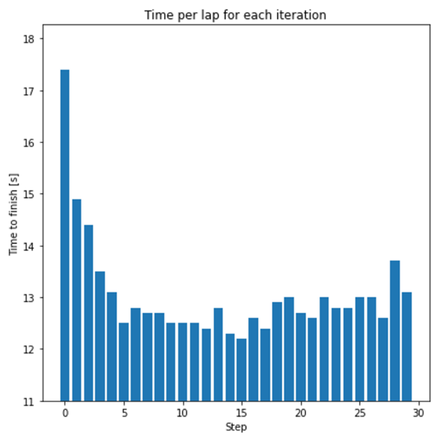
Predictably, as highlighted in Figure 4.20, widening the safe set and considering the last 3 iterations instead of the last 2 for the nearest neighbour (i_j = 3 instead of 2) the car no longer gets stuck. Its computational time is also a bit reduced (8.935s). More on i_j later.



(a) K = 5.



(b) K = 8.

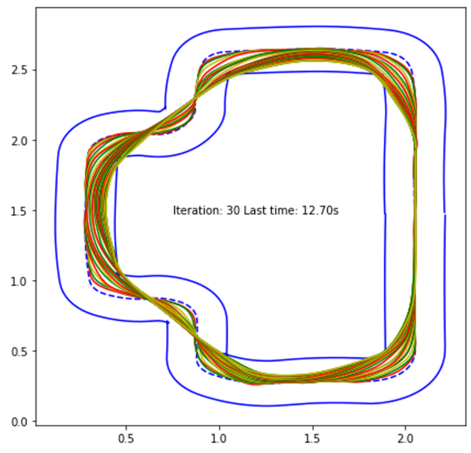


(c) K = 10, i_j = 3.

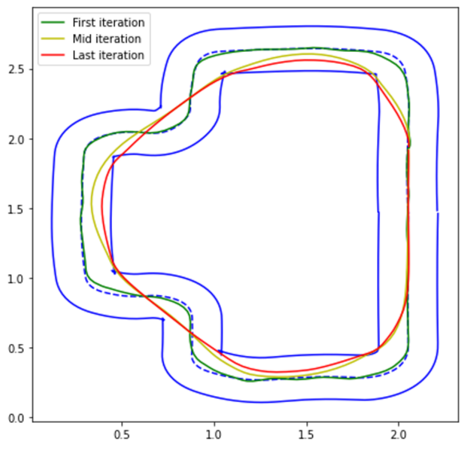
Figure 4.21: Time per lap with different K.

Smaller K means smaller exploration and steadier decrease but it takes longer to reach smaller lap times. Its computational time is also noticeably bigger. This makes sense considering that the optimizer needs to find a solution across a bigger safe set.

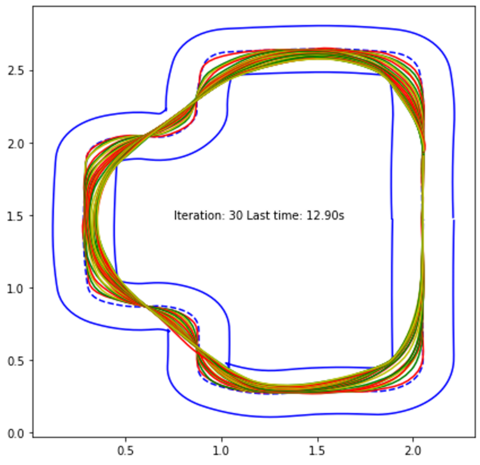
Sensitivity for i_j



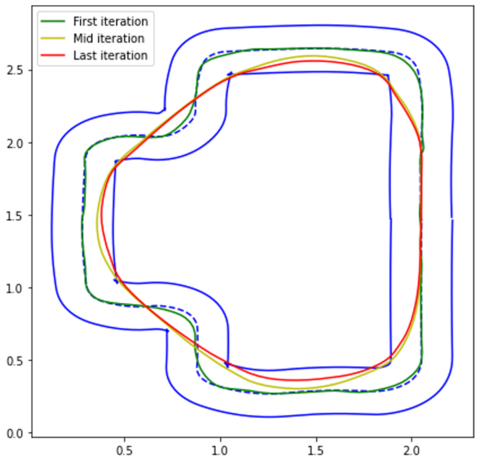
(a) i_j = 2, avg comp time: 9.361s.



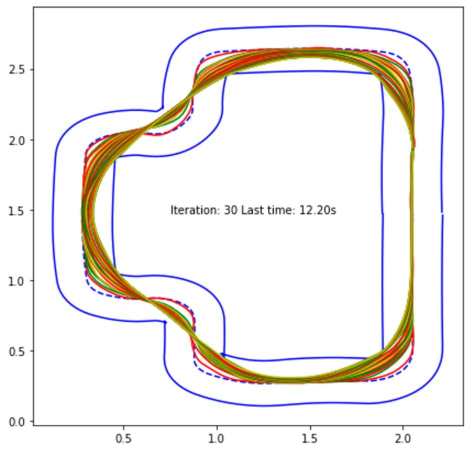
(b) i_j = 2, first mid and final iteration.



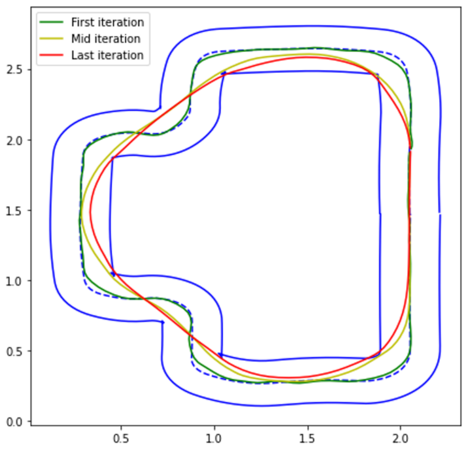
(c) i_j = 3, avg comp time: 9.43s.



(d) i_j = 3, first mid and final iteration.



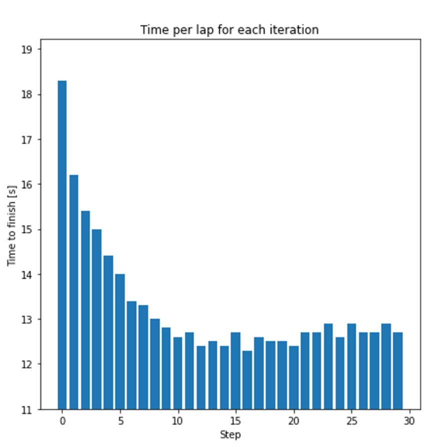
(e) i_j = 4, avg comp time: 10.46s.



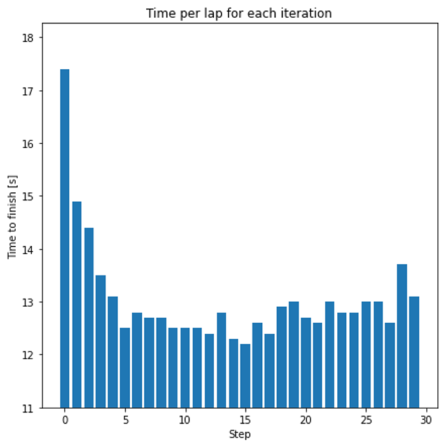
(f) i_j = 4, first mid and final iteration.

Figure 4.22: Laps with different i_j given N=3 and K=8.

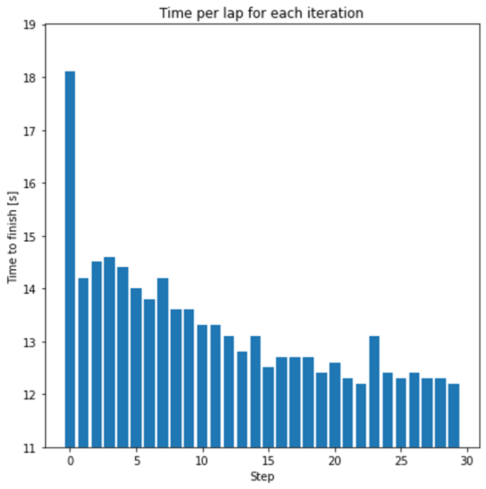
Predictably, as explained in the previous section, bigger i_j means bigger safe set and longer computational time. With i_j = 1 it gets stuck pretty quickly but its computational time is a noteworthy 8.87s per lap.



(a) i_j = 2, best: 12.70s.



(b) i_j = 3, best: 12.90s.



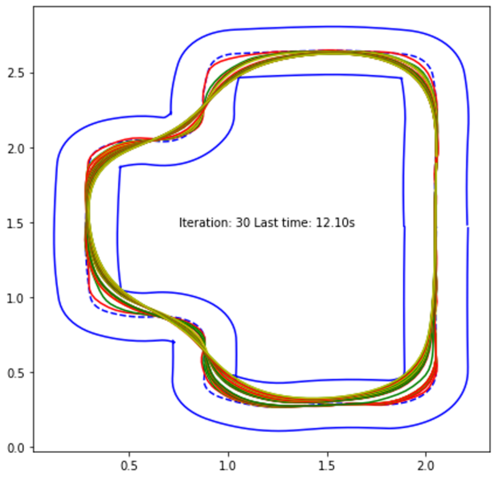
(c) i_j = 4, best: 12.20.

Figure 4.23: Laps time evolution with different i_j

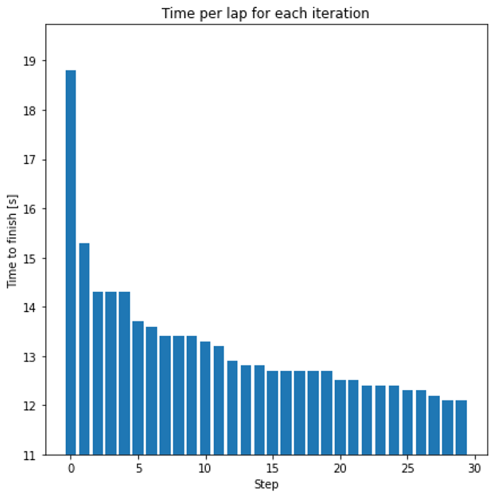
As visible in Figure 4.23 bigger i_j means a slower but steadier decrease in lap time. It also corresponds to an higher computational burden.

Conclusions

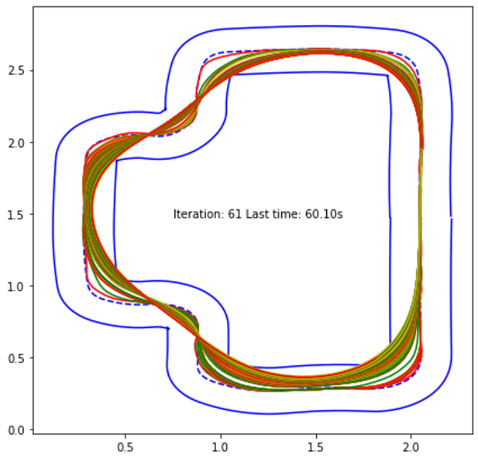
With N=2, K=8 and i_j=4 and an average computational time per lap of 7.43s this MPC breaks all the records for speed and reliability. It gets stuck after 60 laps after stabilizing at 11.8s at around the 45th lap.



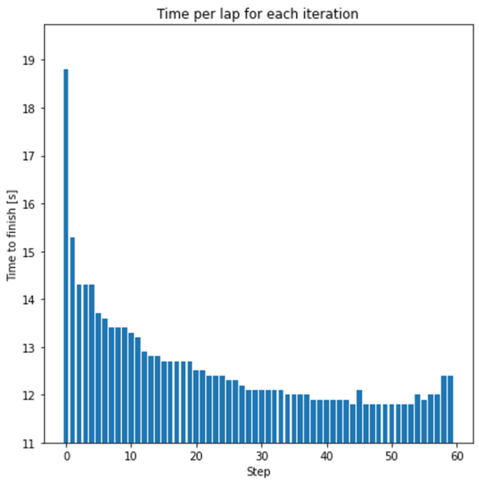
(a) 30 laps with chosen parameters.



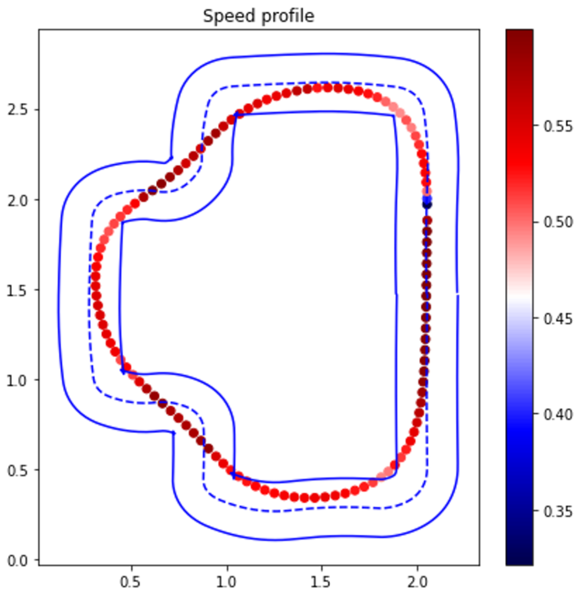
(b) Laps time evolution with chosen parameters.



(c) Keep running until gets stuck.



(d) Keep running until gets stuck lap time evolution. Best time is 11.8s



(e) Best lap speed profile

Figure 4.24: N=2, K=8, i j=4

4.3. Small improvements

4.3.1. A simplification

From the original implementation I removed h(x), but when the car is close to the finish lane I set the cost of the first steps of the next iteration to 0, de-facto pushing the car to position itself in the original position and orientation. This resulted in a faster execution while the final result is the same.

$$J^{LMPC,j}_{t\rightarrow t+N}(x^j_t, z^j_t) = \min_{U^j_t,\lambda^j_t} \sum_{k=t}^{t+N-1} h(x^j_{k|t}) + J^{j-1}_l(z^j_t)\lambda^j_t \tag{4.43}$$

$$h(x) = \begin{cases} 0 & \text{if } x \in x_f. \\ 1 & \text{else.} \end{cases} \tag{4.44}$$

With N=2, K=8 and i_j=4 the average computational time per lap is 7.43s without h(x), 8.47s with h(x). Making this new approach noticeably faster while obtaining identical performances in term of speed per lap.

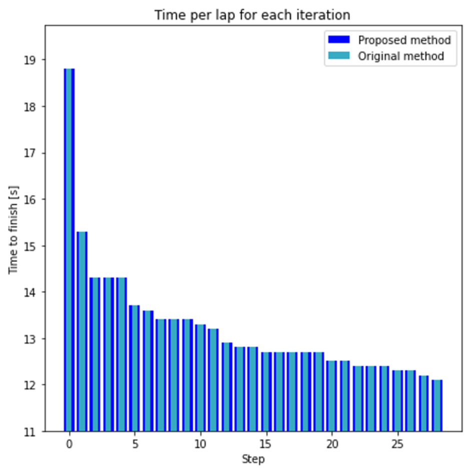


Figure 4.25: Compared time evolution with and without h(x)

4.3.2. Flying start

An issue of the LMPC is that at each lap it will target speed 0 at the end of the lap, lowering both its final speed and its initial speed. It is acceptable if the car needs to stop at the finish line, but having multiple laps this is not the best approach. To solve this issue we implemented a "flying start". This consists in saving the initial speed of the car at 0.6 (the maximum speed) instead of 0 for the first lap, tricking the LMPC into thinking that the car never started at speed 0. This approach led to a new best time of 11.7s, against the previous 11.8s.

4.4. Comparison with baseline

As a possible baseline consists in finding a trajectory that minimizes the the trajectory length and the curvature, trying to find the smallest path with the highest average speed. To optimization variable α is defined as:

$$\alpha : \begin{cases} x_i = x_i^{in} + \alpha_i(x_i^{out} - x_i^{in}) \\ y_i = y_i^{in} + \alpha_i(y_i^{out} - y_i^{in}) \end{cases}, \alpha_i \in [0,1] \tag{4.45}$$

With $[x^{in}, y^{in}]$, $[x^{out}, y^{out}]$ the inner and outer track border. The length minimization will be:

$$J_l(\alpha_0, ..., \alpha_N) = \sum_{i=0}^{N-1} (\Delta s_i)^2 \tag{4.46}$$

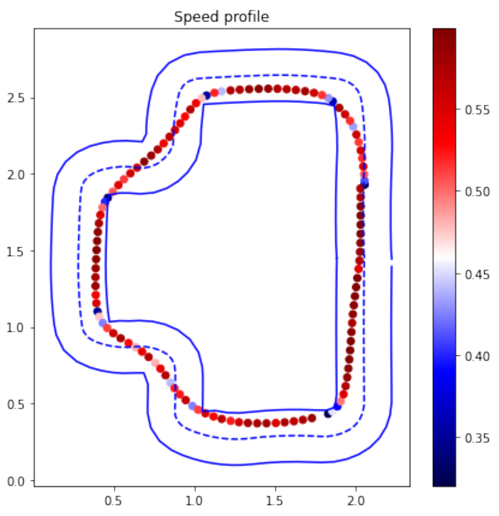
With $(\Delta s_i)^2 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$.
The curvature instead:

$$J_\rho(\alpha_0, ..., \alpha_N) = \sum_{i=0}^{N-1} (\Delta \rho_i)^2 \tag{4.47}$$

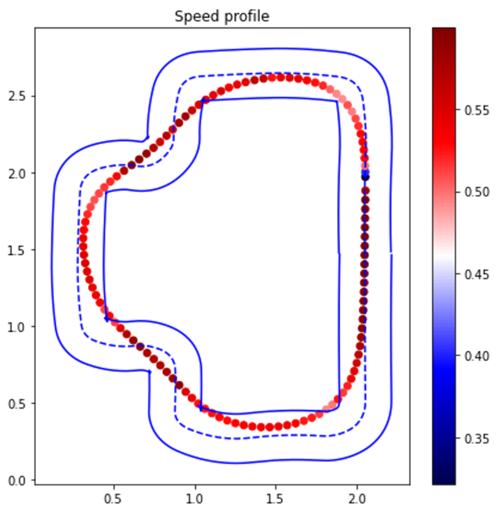
With $\rho_i = \frac{\Delta\theta_i}{\Delta s_i}, \Delta\theta_i = \arctan(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}) - \arctan(\frac{y_i - y_{i-1}}{x_i - x_{i-1}})$.
The function to be minimized will be:

$$\min_{\alpha_0,\dots,\alpha_N} \ k_l J_l + k_\rho J_\rho \tag{4.48}$$

With J_l and J_ρ being two parameters to find. the next step would be to find the speed profile, but we don't need it having a MPC that maximizes speed. To find the optimal parameters the Bayesian hyperparameter finder Optuna [1] has been used. The lap time of the car using this approach is of 12s, against the 11.7s obtained by the LMPC with flying start.



(a) Baseline speed profile.



(b) LMPC speed profile.

Figure 4.26: Speed profile (in m/s) of baseline against LMPC

As visible in Figure 4.26 while the baseline is able to saturate the car speed more, it lacks the ability to understand the car dynamics thus resulting in a less consistent speed and in sudden decelerations.

5 | Experimental results

5.1. Model identification

The model's parameters are dependent on several physical factors, which require estimation on the actual model. The parameters to be estimated, in 5.2, are γ_1, γ_2, γ_3, ξ_1, ξ_2, ξ_3, a_r^u, a_l^u, a_r^ω, a_l^ω.

$$\begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\gamma_1 u - \gamma_2 \omega + \gamma_3 \omega^2 \\ -\xi_1 \omega - \xi_2 u - \xi_3 u \omega \end{bmatrix} + \begin{bmatrix} a_r^u & a_l^u \\ a_r^\omega & a_l^\omega \end{bmatrix} \begin{bmatrix} V_r \\ V_l \end{bmatrix} + \begin{bmatrix} \Delta_u \\ \Delta_\omega \end{bmatrix} \tag{5.1}$$

Equation 5.2 represents the model, where the parameters' physical significance is not immediately apparent and necessitates their estimation. To estimate these parameters, various experiments have been conducted to explore different scenarios, and the collected values have been normalized between 0 and 1. To estimation formulation is:

$$\min_{\gamma_1,\gamma_2,\gamma_3,\xi_1,\xi_2,\xi_3,a_r^u,a_l^u,a_r^\omega,a_l^\omega} \sum_{N=1}^{10} ((F_x(\gamma_1, ..., a_l^\omega, x_{i+N-1}) - x_{i+N})^2 + (F_y(\gamma_1, ..., a_l^\omega, y_{i+N-1}) - y_{i+N})^2)\forall_i \tag{5.2}$$

The final calibrated values for the specific Duckiebot we used are those in table 5.1

Parameters

	Ideal parameters	**Estimated parameters**
γ_1	5	2.14148837
γ_2	0	0.12200042
γ_3	0	-0.28237442
ξ_1	4	1.3380637
ξ_2	0	0.40072379
ξ_3	0	1.30781483
a_r^u	1.5	1.30781483
a_l^u	1.5	1.03762896
a_r^ω	15	2.9650673
a_l^ω	15	2.89169198

Table 5.1: Estimated parameters.

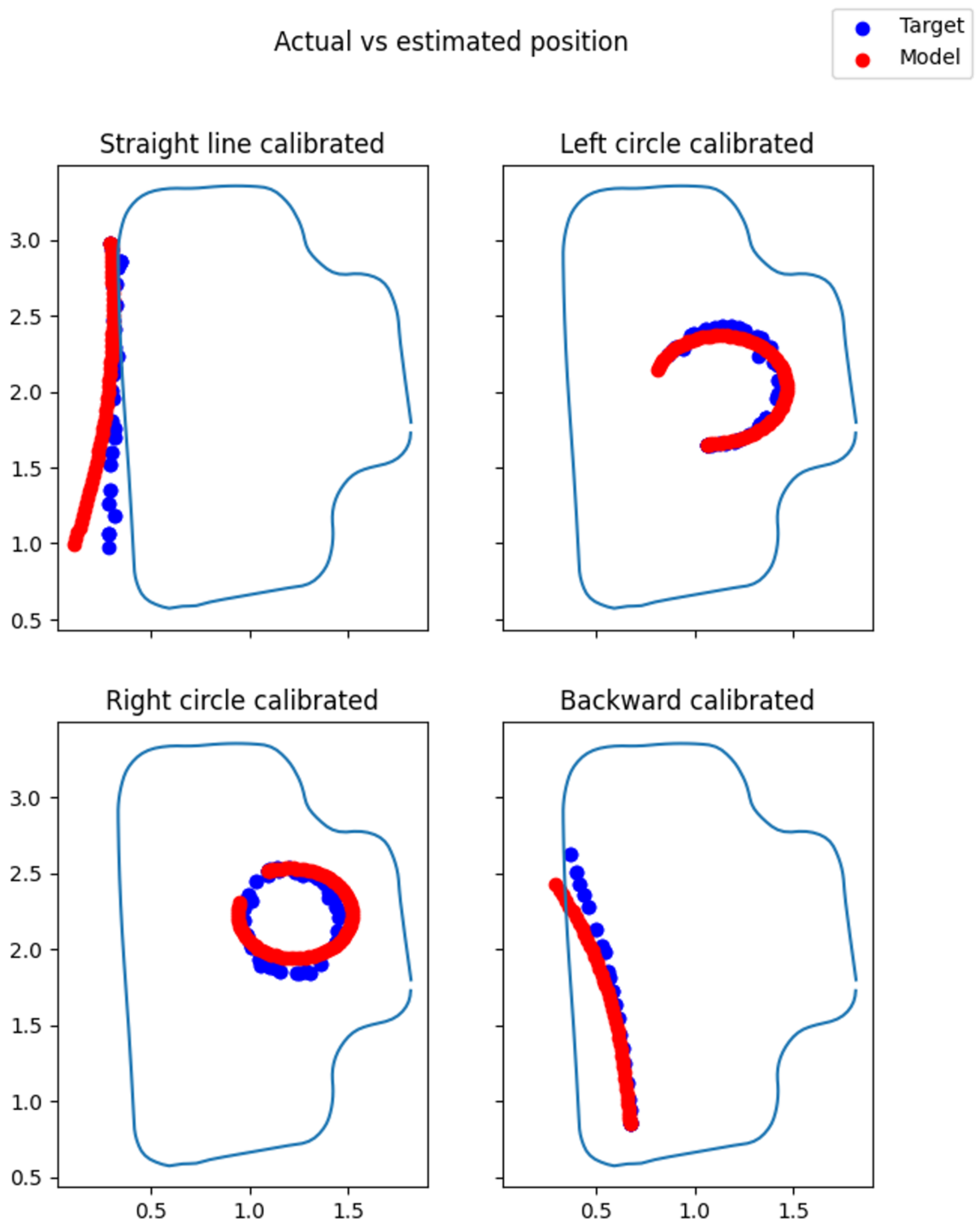
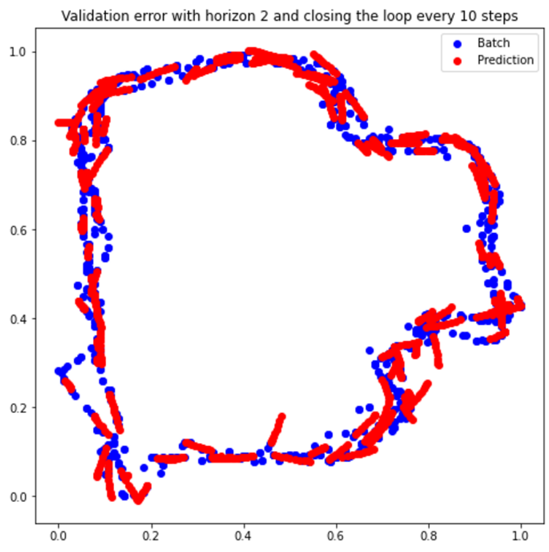
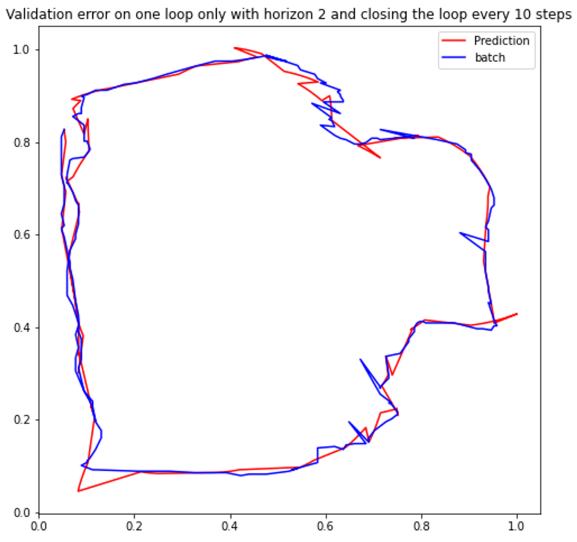


Figure 5.1: Some of the training data we used. Other than this some MPC loops around the track have been used.



(a) Validation example 1.



(b) Validation example 2.

Figure 5.2: Two MPC loops as validation. 10 steps open loop before a correction. The MSE is 0.04, using default parameters it was 0.06. Mean error on $x : 0.00191m^2, y : 0.00186m^2$. This is 4cm, that is around the estimated localization error from the watch-tower.

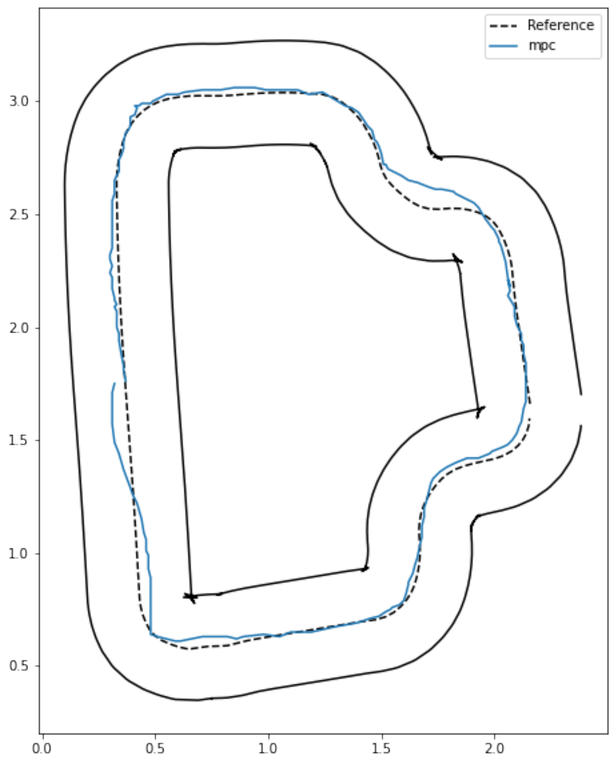
5.2. MPC

The application of MPC without preview using real parameters proved to be ineffective both in simulation and on the actual track. Therefore, MPC with preview was utilized. In this study, the formulation used for MPC with preview is the one already defined in section 4.1.2, with the following parameters: $N = 10$, $Q_1 = 10^3$, $Q_2 = 10^{-2}$, $Q_3 = 0$ and $R = 10$.

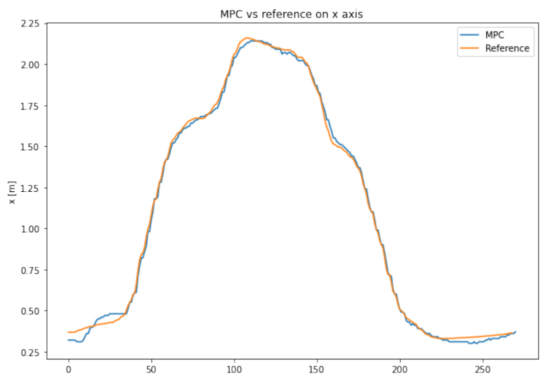
Given that the track is surrounded by walls, precautions have been taken to ensure the safety of the car by limiting its speed to half of its maximum capability, thereby preventing any potential damages.

To further evaluate the performance of MPC with preview, several simulations were conducted to test its effectiveness in different scenarios. The results of these simulations were compared to those of the previous MPC without preview. See figure 5.1

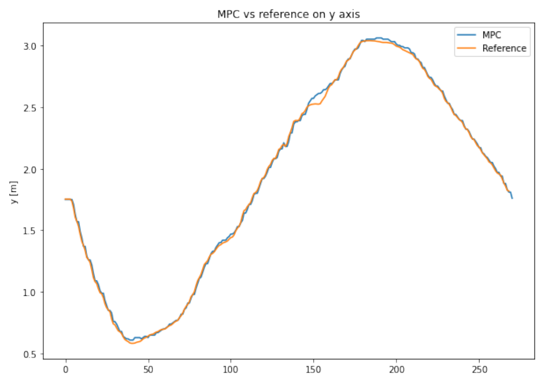
Overall, the results of this study highlight the importance of utilizing MPC with preview when dealing with complex control problems, particularly in scenarios where real-world physical constraints must be taken into account.



(a) MPC run in Milano Duckar.



(b) MPC error on x.



(c) MPC error on y.

Figure 5.3: MPC run in Milano Duckar.

5.3. LMPC

The LMPC formulation we originally proposed is a MPC without preview and as explained in the section before it can not work. To overcome this limitation a new LMPC with preview has been designed. The idea is:

1. Compute the reference as a line between the current position and the target, where the target is the nearest point in the previous iteration at distance d, where d is a

tunable parameter. For d we used $d = N * 0.03$, being 3cm the distance that the duckiebot covers in 0.1s;

2. Run the MPC with preview and a slack on the reference, to be able to generate a trajectory that stays inside the margins.

The formulation is in 5.10

$$\min \sum_{k=0}^{N+1} \|p_k - p_r\|_{Q_1}^2 + \|\theta_k - \theta_r\|_{Q_2}^2 + \|u_k - u_{max}\|_{Q_3}^2 + \|U\|_R^2 \tag{5.3}$$

$$s.t. \tag{5.4}$$

$$X_{k+1} = F(X_k, U_k) \tag{5.5}$$

$$-0.1 \leq u \leq 0.5 \tag{5.6}$$

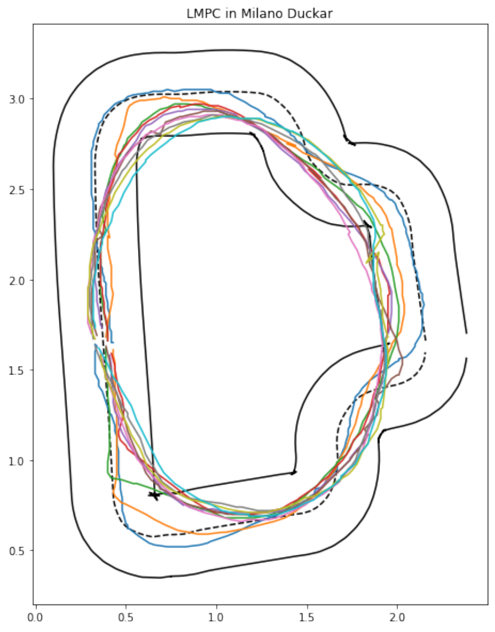
$$X_0 = [p_0, \theta_0, u_0, v_0, \omega_0]^T \tag{5.7}$$

$$r \leq p + s \tag{5.8}$$

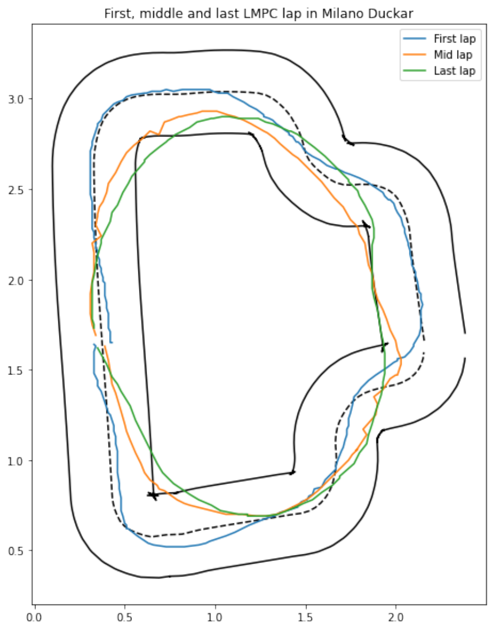
$$rp - s \tag{5.9}$$

$$\lambda_{ss} \leq 0, I1\lambda_{ss}, T(X_k^j)\lambda_s s = r \tag{5.10}$$

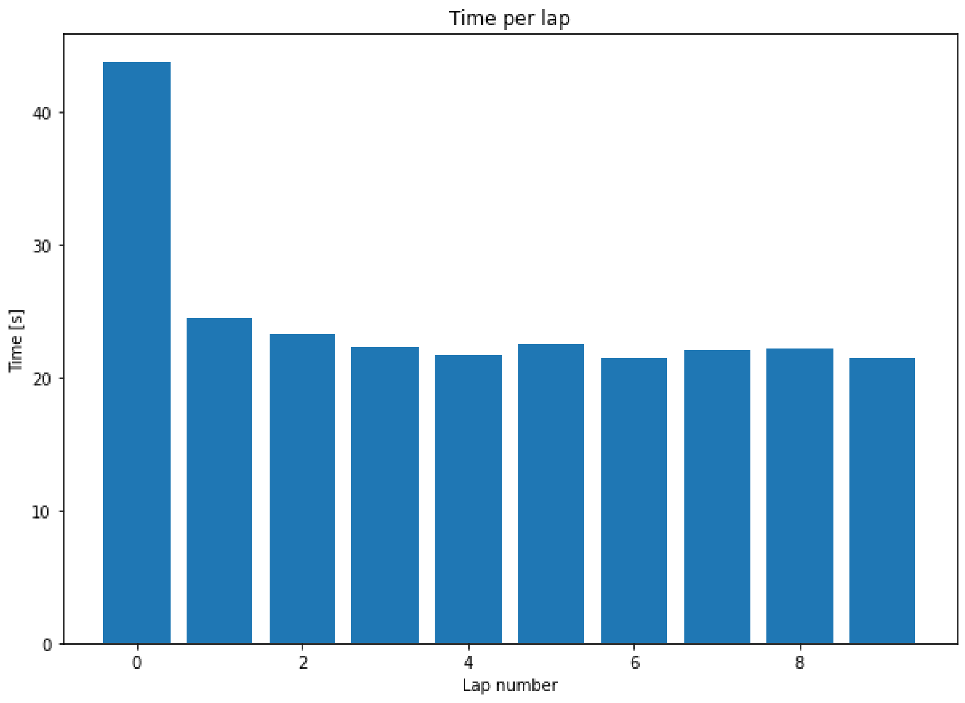
With $U = [V_l, V_r]^T$, $X_k = [x_k, y_k, \theta_k, u_k, \omega_k]^T$, p vector of reference point, s vector of slacks, $T(X_k^j)$ track margins close to the point.



(a) LMPC run in Milano Duckar.



(b) Significative LMPC laps.

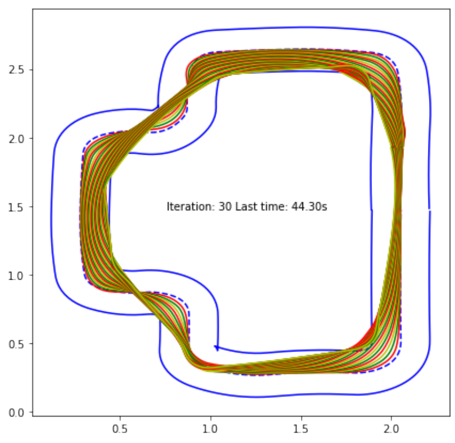


(c) LMPC lap times.

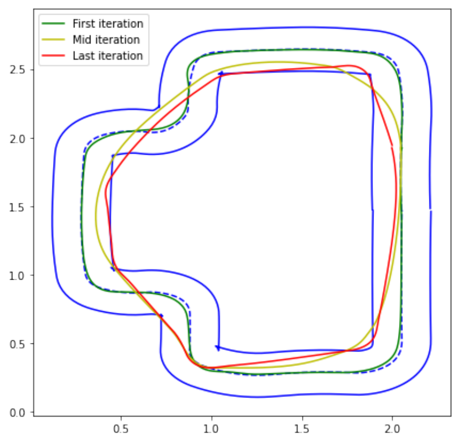
Figure 5.4: LMPC in Milano Duckar.

Remarking that lap 0 is the MPC, as visible in 5.4c already in the first lap of the LMPC the time per lap improves from 43.7s to 24.4s, for then to stabilize at around 22s when the Duckiebot starts getting very close to the margins.

In simulation we can further explore the behaviour of the LMPC with preview in more laps. As in Figure 5.5 we can see that the LMPC learns a trajectory close to the margin only sometimes partially cutting the corners. This behaviour is expected and has been taken into account when designing the margins. The margins are defined as only 5 points along the border and it can happen that the close to the corner the two references for the internal margin are before and after the corner, de facto changing the shape of the track. To prevent this from happen we can use more points in the margin convex hull, but this slows down the computation significantly, with small improvements in terms of control.



(a) LMPC with preview.



(b) Significative LMPC with preview laps.

Figure 5.5: LMPC with preview in simulation.

It is noteworthy that this new formulation of the LMPC runs online on board the Duckiebot Jetson Nano, as there is only one simple convex hull involved.

6 | Conclusions and future directions

In conclusion, the results presented in this thesis provide evidence that a Learning Model Predictive Control (LMPC) approach can be successfully applied to the task of driving a Duckiebot in a real-world environment, with improved performance compared to a traditional MPC. Through exploration and careful reference picking, the learning MPC was able to adapt and improve its control strategy based on real-time feedback from the environment. This resulted in smoother and more efficient trajectories, with faster completion times.

The learning MPC also demonstrated its ability to handle complex scenarios, such as navigating with close constraints and adjusting to changes in the environment, which can be challenging for traditional control methods. Additionally, this thesis proved the learning MPC to be able to generalize well to new environments, indicating its potential for real-world applications beyond the specific setting in which it was originally developed.

Overall, this thesis provides compelling evidence that a learning MPC approach can be a valuable tool for autonomous navigation tasks and can potentially lead to improved performance and robustness in real-world applications. Further research is needed to explore the full potential of this approach and its integration with other control strategies and sensors.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[3] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull. Duckietown environments for openai gym. `https://github.com/duckietown/gym-duckietown`, 2018.

[4] S. Ercan. Modelling strategies for a mobile robot. Available at `https://drive.google.com/file/d/19U1DUo3GtqHxncEKLn2d6RRdLTLgD0Bv/view` (29/04/2019).

[5] A. F. K. K. Konstantin Chaika, Anton Filatov. Automatic wheels and camera calibration for monocular and differential mobile robots. *Applied Sciences 11*, 2021. URL `https://www.duckietown.org/archives/79889`.

[6] A. K. I. A. Konstantin Danilov, Ruslan Rezin. Towards blockchain-based robonomics: autonomous agents behavior validation. *Arxiv*, 2018. URL `https://arxiv.org/abs/1805.03241`.

[7] S. O. C. A. J. P. H. Miao Liu, Kavinayan Sivakumar. Learning for multi robot cooperation in partially observable stochastic environment with macro-actions. *IROS 2017*, 2017. URL `https://arxiv.org/abs/1707.07399`.

[8] D. J. X. N. Z. R. A. E. M. r. Michael Yuhas, Yeli Feng. Embedded out-of-distribution detection on an autonomous robot platform. *Design Automation for CPS and IoT (DESTION 2021) Workshop*, 2021. URL `https://dl.acm.org/doi/abs/10.1145/3445034.3460509`.

[9] K. K. A. S. Mikita Sazanovich, Konstantin Chaika. Imitation learning approach for ai driving olympics trained on real-world and simulation data simultaneously.

Workshop on AI for Autonomous Driving (AIAD), the 37th International Conference on Machine Learning, Vienna, Austria, 2020, 2020. URL `https://arxiv.org/pdf/2007.03514.pdf`.

[10] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *CoRR*, abs/1604.07446, 2016. URL `http://arxiv.org/abs/1604.07446`.

[11] B. G.-T. Péter Almási, Róbert Moni. Robust reinforcement learning-based autonomous driving agent for simulation and real world. *2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207497*, 2020. URL `https://ieeexplore.ieee.org/document/9207497`.

[12] F. B. Ugo Rosolia. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 2017. URL `https://www.sciencedirect.com/science/article/pii/S2405896317306523`.

[13] F. B. Ugo Rosolia. Learning how to autonomously race a car: A predictive control approach. *IEEE Transactions on Control Systems Technology, vol. 28, no. 6, pp. 2713-2719*, 2020. URL `https://ieeexplore.ieee.org/abstract/document/8896988`.

[14] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming 106(1), pp. 25-5*, 2006.

List of Figures

List of Tables

List of Symbols

Variable	**Description**	**SI unit**
$\boldsymbol{x}$	x coordinate	m
$\boldsymbol{y}$	y coordinate	m
$\boldsymbol{\theta}$	orientation	rad
$\boldsymbol{v}$	longitudinal speed	m/s
$\boldsymbol{\omega}$	angular speed	rad/s
$\boldsymbol{Q_1}$	weight on distance from coordinate reference	
$\boldsymbol{Q_2}$	weight on distance from angular reference	
$\boldsymbol{Q_3}$	weight on difference between input and max input	
$\boldsymbol{R}$	weight on control action	

Acknowledgements

Grazie alla mia famiglia e ai miei amici.