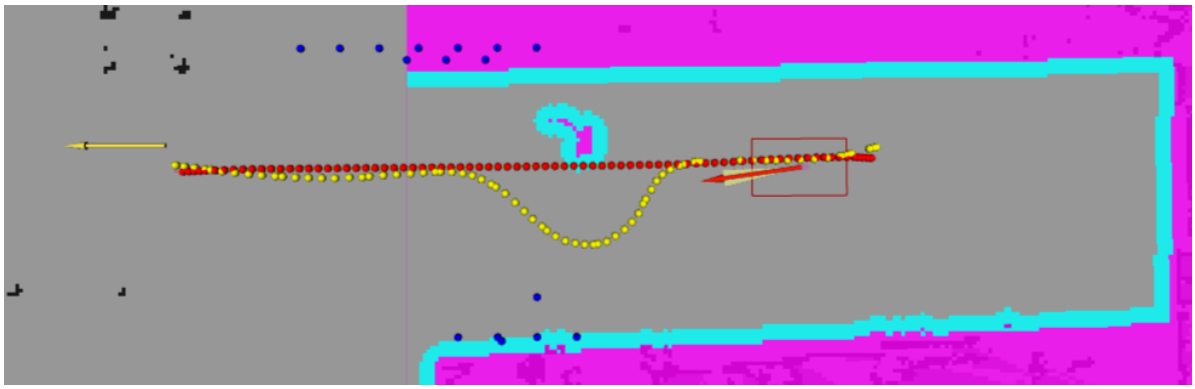
TÉCNICO LISBOA



Nonlinear Model Predictive Control for trajectory tracking and obstacle avoidance of industrial mobile robots

Francisco Maria de Lima Raposo Madeira

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Alberto Manuel Martinho Vale
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. Pedro Tiago Martins Batista
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Maria Isabel Lobato de Faria Ribeiro

December 2023

“If I have seen further it is by standing on the shoulders of Giants.”

- Sir Isaac Newton

To all the Giants whose shoulders I have stood on.

Thank you.

DECLARATION

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, a word of appreciation must go out to both my supervisors Professor Rodrigo Ventura and Professor Alberto Vale, for giving me an opportunity to develop such a project. Without their trust, knowledge, guidance and patience this work would have never been possible.

A sincere thank you also goes out to João Tavares, João Mendes and Inês Silva, with whom I had the pleasure of working with. Through and through, they have always shown incredible support to the development of this work, willingly volunteering their time and constantly sharing insights and meaningful expertise.

This thesis also marks the end of a long academic journey. For this and because of this, it would be wrong not to acknowledge so many people who have shaped my path and who have given me so much.

To the beautiful city of Torino, a note of appreciation for a life full of novelty, of experiences and adventure. To Diogo Guerreiro, Diogo Silva, João Silva and Vasileios Lokovitis, thank you for everything you have taught me there and ever since.

To Instituto Superior Técnico, thank you for a group of friends whom I consider to be unwavering standards of excellence. To Daniel Farinha, Henrique Figueiredo, Hugo Pereira, João Leitão, José Reis and Luís Azevedo know that you have made all the work, the all-nighters and the sacrifice bearable and, might I risk, enjoyable.

To my greatest friends and to everything we have lived. To Francisco Cortez, Francisco Seabra, João Faria, João Mangerona, José Albuquerque, José Martins, Miguel Neves and Miguel Ressurreição, there is not much to say that has not already been said or lived through. In everything and for everything, thank you.

Finally, a note of profound gratitude must go out to my closest family. To my father, Rogério, and my mother, Marta, who appear to keep finding new ways to understand and support me. For the love, for leading the way, for the service and for the values, thank you. To my sister, Carminho, and my brother, Vicente, who have made me want more than ever to be the example I see in others. Thank you.

To all of you that were mentioned here, and to all of you that were not, know that this work is as much yours as it is mine. I am but a reflection of your example. These are the shoulders upon which I stand.

Resumo

A procura por maior eficiência em processos industriais levou à necessidade de desenvolvimento de tecnologias de automação, particularmente em intralogística. Neste campo, veículos autónomos acabam por oferecer maior robustez e flexibilidade, quando comparados com soluções tradicionais. Este trabalho aborda o desafio de dotar robôs autónomos de capacidades de navegar através de ambientes industriais, sendo capazes de se desviar de obstáculos estáticos e dinâmicos, enquanto fazem o seguimento de uma trajetória de referência de forma correta.

Um controlador baseado em *Nonlinear Model Predictive Control* (NMPC) é desenvolvido como solução para o seguimento de uma trajetória de referência, aspeto crucial para a implementação de uma frota de robôs autónomos. O controlador baseia-se numa função de custo que penaliza a distância à referência e otimiza o controlo. Como garantia de segurança, são impostas restrições de modo a garantir uma distância mínima de potenciais obstáculos ao longo do horizonte preditivo. A abordagem deste trabalho aumenta a eficiência computacional do problema através da representação do mapa de ocupação dentro de uma janela de visualização, sobreposta com uma grelha de células *voxel*. Adicionalmente, restrições específicas devido ao comportamento real do robô são incorporadas no controlador.

A validação da eficácia do controlador desenvolvido é feita em chão de fábrica, com condições reais, e revela precisão no seguimento da trajetória de referência e capacidade de desvio de obstáculos estáticos e dinâmicos. A comparação com o *Timed-Elastic-Band* (TEB) revela que o controlador desempenha melhor seguimento de trajetória e tem uma capacidade de desvio de obstáculos comparável à deste método.

Palavras-chave: *Nonlinear Model Predictive Control*, seguimento de trajetória, desvio de obstáculos estáticos, desvio de obstáculos dinâmicos, condições reais.

Abstract

The growing demand for efficiency in industrial processes has prompted a need for automation technologies, particularly in intralogistics. In this field, autonomous vehicles offer enhanced robustness and flexibility when compared to traditional solutions. This work addresses the challenge of enabling autonomous robots to navigate through factory environments, avoiding both static and dynamic obstacles, while accurately tracking a reference trajectory.

A Nonlinear Model Predictive Control (NMPC) controller is introduced as a solution and designed to track a time-parameterized reference, a crucial feature for deploying a fleet of autonomous robots. The controller formulates a cost function that minimizes the distance to the reference trajectory and optimizes control inputs. To ensure safety, hard constraints are imposed to maintain a minimum distance from potential obstacles along the prediction horizon. The approach taken enhances computational efficiency by representing the occupancy map data within a view-window using an overlaid voxel grid. Additionally, behaviour-specific constraints derived from real-world robot experimentation are incorporated.

Real-world testing on factory floor validates the effectiveness of the developed controller, demonstrating precise trajectory tracking and the ability to navigate around both stationary and moving obstacles. A comparison with the Timed-Elastic-Band approach used as a baseline reveals that the controller has superior tracking performance while maintaining comparable obstacle avoidance capabilities.

Keywords: Nonlinear Model Predictive Control, trajectory tracking, static obstacle avoidance, dynamic obstacle avoidance, real-world conditions.

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

α	Angle between vehicle's heading and vector that connects target point to its rear axle
β	Angular range
Δ	Error
δ	Steering angle
λ	Weighting coefficient
ω	Angular velocity
θ	Heading angle

Roman symbols

(q_0, q_1, q_2, q_3)	Quaternions
(x, y, z)	Cartesian coordinates
$\boldsymbol{u}$	Control inputs
$\boldsymbol{x}$	State
a	Acceleration
c	Height
d	Distance
f	Frequency
h	Time step
k	Optimization time step
L	Track width
l	Wheel-base
N	Prediction horizon length

p	Position
t	Time
v	Linear velocity
w	Width

Subscripts

0	Initial
max	Maximum
min	Minimum
nom	Nominal
obs	Obstacle
range	Range
ref	Reference condition
replan	Replanning
safe	Safety
size	Dimensions
c	Controller
L	Robot's left-wheel
l	Look-ahead
R	Robot's right-wheel
r	Robot
s	Step

Superscripts

opt	Optimal
T	Transpose

Glossary

2-D Two-dimensional. 25, 26

AGV Automated Guided Vehicle. xix, 2, 3

AMCL Adaptive Monte Carlo Localization. xvii, xix, xx, 33, 44, 45, 48, 49, 55–57, 61, 62, 64, 78, 79, 91

AMR Autonomous Mobile Robot. 2, 3

APF Artificial Potential Fields. 8, 11, 17

AUV Autonomous Underwater Vehicle. 11

CPU Central Processing Unit. 28

CPU Graphics Processing Unit. 28

DWA Dynamic Window Approach. 12

EB Elastic-Band. 11

GB Gigabyte. 28, 47

IPOPT Interior Point Optimizer. 43, 79

IQR Interquartile Range. 52

IST Instituto Superior Técnico. 2

LIDAR Light Detection and Ranging. 8, 19

LQR Linear-Quadratic Regulator. 6, 7

MPC Model Predictive Control. xix, 6, 7

NA Not Applicable. 48

NLP Nonlinear Programming Problem. 19, 21, 23

NMPC Nonlinear Model Predictive Control. ix, xi, xvii, xix, 4, 13–23, 25, 26, 37, 43, 45, 50–52, 56, 60, 77–79

OCP Optimal Control Problem. 16, 18, 21, 50

PANOC Proximal Averaged Newton-type method for Optimal Control. 19

PID Proportional-Integral-Derivative. 6

RAM Random Access Memory. 28, 47

ROS Robot Operating System. xix, 11, 24, 37, 44, 45, 49

TEB Timed-Elastic-Band. xvii, 11–14, 47, 50, 55, 58, 59, 62, 64, 67–74, 76, 77, 91, 92

UAV Unmanned Aerial Vehicle. 3, 9, 11

VO Velocity obstacle. xix, 9, 10

Chapter 1

Introduction

1.1 Motivation

This thesis addresses trajectory tracking and obstacle avoidance for a differential drive robot in an industrial environment. A complete robotic motion planning system integrates both global and local planner. The global planner is responsible for generating the robot's trajectory, from initial point to target. It requires that a map of the environment is given previously and generates a feasible trajectory, this is, taking into account the robot's physical constraints and any other specific requirements. A local planner, on the other hand, deals with the robot's immediate surroundings. It is responsible for trajectory following and the avoidance of any obstacles that had not been considered by the global planner. It relies on sensor data and guarantees the robot's safe operation.

As part of the AGiLE project, explained below, the developed work determines the optimal control inputs needed for a robot to accurately follow a dynamic trajectory, while being equipped with obstacle avoidance capabilities. In this section, both the industrial context and aerospace motivation behind this dissertation are explained. This thesis develops a local planner, to complement the global planner developed by Silva [1], as part of the same AGiLE project.

1.1.1 Industrial context

Technological advancements have traditionally been regarded as the way to efficiently increase the productive output of an industry. Automation, newer machines, processes and management systems, have allowed for the better use of resources into an end product [2]. These advancements benefit both clients and manufacturers. The client has access to more affordable and higher quality products, while the manufacturer can reduce costs and take advantage of economies of scale [3]. The efficient increase in production that derives from these advancements is a major concern for every company and a key contributor to the development of a nation's economy. It is imperative that there is a constant investment in better and more agile technologies.

Furthermore, it is important to understand logistics, or logistics management, as defined by the Council of Supply Management Professionals as "that part of supply chain management that plans, im-

plements, and controls the efficient, effective forward and reverses flow and storage of goods, services, and related information between the point of origin and the point of consumption to meet customers' requirements" [4]. The most important field of this being the field of intralogistics, understood as the management, optimization, and distribution of products within the same warehouse or manufacturing unit [5].

The AGiLE project derives from a partnership between Instituto Superior Técnico (IST), Imeguisa, and Volkswagen Autoeuropa looking to provide end-to-end automation of intralogistics flow inside the Volkswagen Autoeuropa car manufacturing plant. The overall goal is to positively impact production outcomes by making the processes inside the factory more agile and cost-effective. To do this, the project will integrate a fleet of Autonomous Mobile Robots (AMRs) into the factory environment. This deployment is done through a decentralized architecture, that requires little to no human interaction and allows for a greater logistical autonomy.

The current solution for the flow of material from storage to production line relies on Automated Guided Vehicles (AGVs) to handle transportation. These robots can only move by following a magnetic stripe on the floor, which means they are limited to repetitive tasks. An AGV is unable to accommodate alternative paths other than the one given by following the magnetic stripe. If it comes across an obstacle it will simply stop and, consequently, halt production. Several use problems are also present, namely due to degradation of the magnetic stripes. Maintenance and upkeep of these stripes is both time-consuming and labor-intensive. Finally, this solution is unable to react to production line demand. The robots abide by a simple "push" strategy. This is, the material is always "pushed" from storage to assembly line, whether it is currently needed or not. In Figure 1.1, it is possible to see an example of how AGVs operate on the factory floor. Note how the robot follows the black magnetic stripe on the floor.

The AGiLE project proposes the deployment of a fleet of AMRs through a decentralized architecture as a fix to the shortcomings of the intralogistics solution currently in use. The agent responsible for deciding whether restock is needed - peripheral - will send a message to all the robots on the shop floor. Upon receiving the request, the robots lead an auction among themselves. Based on availability, distance to goal, battery charge, each robot presents a bid. The winner of the auction will be the robot most suited to complete the task. The global planner will generate the trajectory from the robot's initial

Figure 1.1: Operation of AGVs on Volkswagen Autoeuropa's manufacturing plant floor.

position to the target, considering the map of the environment. Finally, the robot should be able to accurately follow the trajectory and avoid any obstacles in its way. The work developed in this thesis is solely concerned with this last part. Upon arrival to the target, the AMR will convert to an AGV-mode, for docking purposes, since it allows for greater precision.

With the deployment of a fleet of robots that are capable of communicating with one another, unconstrained navigation on the factory floor and adaptation to obstacles, it is possible to better manage the logistical flow inside the Volkswagen Autoeuropa factory. The decentralized architecture of this solution allows for a flexible, reliable and scalable solution. Moreover, by taking into account real-time requirements of the assembly line, so that the supply is accurate and uninterrupted, the AGiLE project overcomes the limitations of the "push" strategy, ultimately allowing for a more efficient use of resources and materials.

1.1.2 Aerospace motivation

There is an ever-increasing adoption of unmanned aerial vehicles (UAVs) as a transformative solution across several industries. Both in civil applications, such as precision agriculture, infrastructure inspection, surveillance, delivery of goods, and scientific research, such as feature mapping in remote locations to geology applications, this technology offers a flexible, reliable and affordable option, with great disruptive potential. In addition, and of particular interest to the industrial context of this thesis, UAVs might revolutionize logistical flows inside factories, by adding a degree of freedom in the z-coordinate [6–9].

Given the requirements of the multiple applications of UAV technology, it is key for reliability that the vehicle can accurately follow a given trajectory. Additionally, adaptability to changes in the environment, i. e., equipping a UAV with obstacle avoidance capabilities is vital for further adoption of these vehicles as trustworthy solutions.

This thesis contributes to the aerospace engineering domain as the algorithm is applicable to UAVs whose trajectories are confined in planes defined as $z = k$, where k is a constant, requiring solely a change in the considered vehicle dynamics. In this case, the robots move on the ground plane, defined as $z = 0$. Besides, it addresses all three main functional technology areas of UAVs: guidance, navigation, and control [10]. It deals with the desired changes in velocity, rotation, and acceleration needed to follow a desired path of travel - guidance, requires the determination of the state of the vehicle - navigation, and, finally, communicates the controls needed to execute the guidance commands - control.

1.2 Objectives

The main objective of this dissertation is to develop a controller for an industrial robot. This controller must be able to accurately follow a dynamically feasible trajectory and avoid both static and dynamic, i.e. moving, obstacles. In order to achieve these end objectives, some intermediate ones, below, must

also be met.

- The study of the theoretical concepts behind trajectory following and obstacle avoidance algorithms and methodologies.
- Formulation of trajectory following as an optimal control problem through the use of a Nonlinear Model Predictive Control (NMPC) controller.
- Incorporation of obstacle avoidance capabilities into the controller, through constraints that must be satisfied for feasibility.
- Algorithm validation in real-world scenarios through testing on factory floor for a differential drive vehicle.

1.3 Literature review

In this section, a literature review on several trajectory tracking and obstacle avoidance algorithms is provided. Additionally, a look into the currently implemented trajectory following solution is presented, as well as a comprehensive review of its shortcomings.

1.3.1 Trajectory tracking

Trajectory tracking is the ability of a vehicle to accurately follow a given reference trajectory. Formally, trajectory tracking is the design of a control law $\boldsymbol{u}(t)$ such that

$$\|\boldsymbol{\chi}(t) - \boldsymbol{\chi}_{\mathrm{ref}}(t)\| = 0 \tag{1.1}$$

for a vehicle model given as a set of differential constraints $\dot{\boldsymbol{\chi}} = f(\boldsymbol{\chi}, \boldsymbol{u})$ and a reference trajectory with an associated timing law $\boldsymbol{\chi}_{\mathrm{ref}}(t) : \mathbb{R} \to \mathbb{R}^n$ [11].

Numerous methodologies have been put forth for this purpose, but this review will focus on the most applicable to car-like vehicles, as it is the use case considered. The choice of an algorithm should take into account the specific requirements of the implementation, but the baseline is that the smaller the deviation from the reference trajectory, the better and more suited the algorithm is considered to be. Henceforth, trajectory following and trajectory tracking will be used interchangeably and referring to the same task.

Note also that the classification throughout this section is not mutually exclusive. Several algorithms and methodologies have variants and may be used to complement each other in several control strategies. The requirements of the implementation will dictate how each technique is used and transformed into a control strategy.

The majority of these algorithms face a trade-off between accuracy and required computational resources. A better, more complete model will produce more realistic results, at the cost of increased

computational complexity. For the problem considered in this work, complexity introduced from reference tracking precision should be carefully balanced with the fast computation needed to react nimbly to perturbations in a rapid-paced factory environment. The controller must be robust to disturbances and uncertainties, such as slight modelling errors, environmental and sensor noise but not so complex that it may hinder its real-time implementation.

Trajectory or path tracking

A common misunderstanding in this field deserves mentioning, primarily, regarding the difference between path and trajectory tracking. Though both address the ability of a system to accurately reach and follow a given set of coordinates, trajectory tracking takes time as an additional constraint. If a two-dimensional plane is considered, a path-tracking algorithm would be considered as successful as its ability to get the robot as close as possible to a reference set of (x, y) coordinates at whatever speed. Contrarily, a trajectory tracking algorithm would need to consider speed as a constraint, as it is required to abide by a set of (x, y, t) coordinates. Path tracking is focused exclusively on a reference, whereas trajectory tracking focuses on a time-parameterized reference.

Model-free algorithms

Perhaps the simplest of trajectory-following controllers, the model-free algorithms consider solely the geometric properties of the trajectory. They do not account for the vehicle model and rely on information about the current position, velocity of the vehicle, and desired trajectory to generate the control inputs. This is the case for the pure pursuit algorithm, whose ultimate goal is to find the steering angle input (δ) at every point in the trajectory.

First, the robot identifies a target point at a specific distance, referred to as the look-ahead distance (d_l), from its present position in the trajectory. Then, an arc is fitted between the rear wheel of the vehicle and the chosen target point [12]. Finally, through trigonometric relations, the steering input can be calculated. These relations are derived from the geometry of the problem, shown in Figure 1.2 for a bicycle-model vehicle, and use the look-ahead distance, the vehicle's wheel-base (l), and the angle between the vehicle's heading and the vector that connects the target point to the vehicle's rear axle (α) [13]. This relation is given by

$$\delta = \arctan\left(\frac{2l\sin\alpha}{d_l}\right). \tag{1.2}$$

Its deployment is straightforward and the algorithm is easily tunable, besides requiring very low computational effort. However, by not considering the vehicle's model, the algorithm lacks accuracy and is unable to handle more complex driving scenarios or higher speeds. Though some work has been developed towards adaption to complexity, as is the case of the work developed by Wang et al. [14], the pure pursuit technique continues to show limited control over robot dynamics.

As an extension of the pure pursuit algorithm, the Stanley controller allows for an improved ability to handle high speeds and sharp turns. It uses a more sophisticated approach, by considering both the vehicle's heading and the curvature of the path. It does not require that a look-ahead distance is defined.

Considering a reference point, the required steering angle is a function of the velocity of the vehicle, a pre-determined gain, the distance from the vehicle's front axle to the reference point and the angle from the vehicle's heading direction to the tangent at the reference point [15].

Model-based algorithms

Model-based algorithms make use of mathematical models to describe the vehicle's dynamics and design a controller such that the inputs given to the system translate into a desired output. In this case, the input is a reference trajectory and the output is the set of controls that allow for tracking of said trajectory. These techniques are accurate, robust and stable, since they can be modeled to handle disturbances without exhibiting unstable or oscillatory behavior. However, they are quite complex and use up a lot of computational resources, making them hard to fit for real-time applications. Additionally, they are sensitive to model errors, such as unmodeled vehicle dynamics and parametric uncertainties.

Often, feedback linearization algorithms are implemented to simplify the nonlinear dynamics of the vehicles considered. The idea behind it is that through variable change and suitable control input, it is possible to achieve an equivalent linear system to the one being considered. That is, a system with the same dynamics that effectively behaves as a linear one and that allows for the use of standard linear control techniques, such as a proportional-integral-derivative (PID) controller. Because of this, feedback linearization is a rather popular approach for the design of nonlinear control systems, enabling the use of well-defined linear control techniques. It is possible to see this applied to nonholonomic car-like robots in the work edited by Laumond [16].

Though there are several model-based algorithms, in this literature review, only Model Predictive Control (MPC) and the Linear-Quadratic-Regulator (LQR) cases are addressed. This is mainly due to both being suited to dealing with disturbances, given their closed feedback loop nature. Further, both operate within a state-space representation, which is a natural way to model and control dynamic systems. Finally, these cases have clear guidelines for tuning, which is a great aide in implementation.

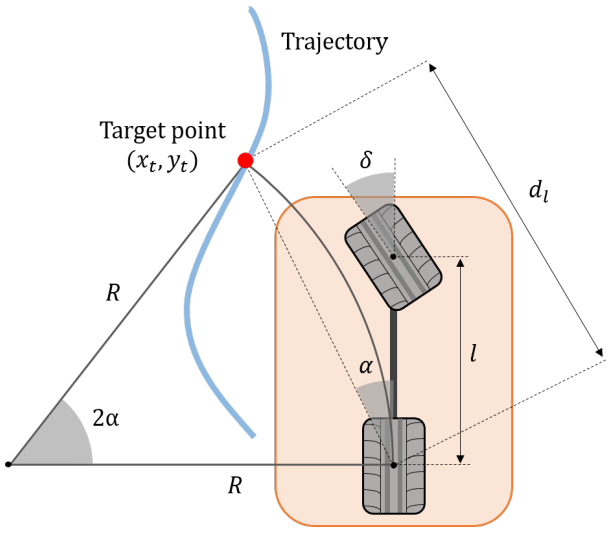


Figure 1.2: Pure pursuit algorithm variable illustration for a bicycle-model vehicle.

As a model-based algorithm, model predictive control (MPC) is a powerful technique that optimizes a function, usually referred to as a cost function, over a prediction horizon to find the optimal controls for the robot. For this, it uses the vehicle's kinematic model to determine its future states. The cost function could, for example, penalize distance to the reference trajectory, which will lead to an optimization of the control variables that minimize this distance and, thus, perform accurate trajectory following. After predicting the states over the horizon, it applies the first instance of the optimal controls and repeats the process of optimization over the horizon, one time step ahead [17]. It considers constraints on both input and output of the system and can deal with complex dynamics, thus making it ideal for industrial applications [18]. The general structure of an MPC controller can be seen in Figure 1.3, adapted from Yakub and Mori's work to the scenario this thesis addresses [19].

Similarly to what happens for the MPC case, a linear quadratic regulator (LQR) also aims at minimizing a cost function that captures the desired performance of the system. However, for this algorithm, the cost function is quadratic and the use of a linear model of the system is required. It is quite a robust and efficient control strategy, as it can handle disturbances and does not require great computational effort [20]. Despite this, the inability to deal with nonlinearities is quite limiting for the use case of this work.

Robust control algorithms

When considering the accuracy needed in certain applications, the use of high-fidelity vehicle models is essential. However, for complex nonlinear systems, a model like this, which considers all disturbances or imprecisions, is computationally very expensive. Robust control techniques appear as a way to tackle this increase in complexity, whilst guaranteeing that no uncertainties, due to the use of a more simplified model, are introduced in the solution.

As one of these techniques, sliding mode control reduces higher-order systems to first-order state variables, resorting to the premise that the system is forced to follow a specific trajectory, or "sliding mode". This trajectory is essentially a cross-section of the system's normal behavior that would guarantee good performance even in the presence of disturbances and uncertainties. The designed switching control law, which leads the system to the sliding mode must also be developed to be robust to these perturbations [21, 22]. This controller is quite suitable to deal with uncertainties and external disturbances, but can suffer from great performance loss due to unmatched disturbances. It also reduces the

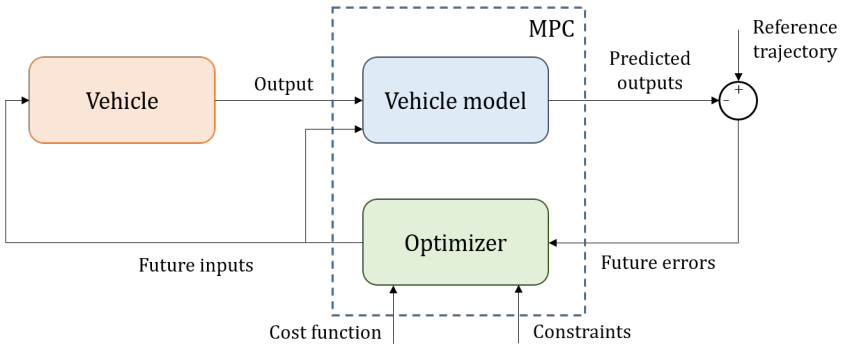


Figure 1.3: General structure of an MPC controller for trajectory tracking.

order of the system dynamics, allowing for a fast response, but has a tendency to excite high-frequency unmodeled dynamics and to suffer from chattering (i.e., oscillation or vibration of the system due to rapidly changing and frequent switching between two or more control values).

1.3.2 Obstacle avoidance

Obstacle avoidance tackles the problem of identifying and avoiding possible obstacles or hazards that may appear in the trajectory of a given vehicle or robot. Identification of said obstacles is typically made using resources such as sonars or LIDAR sensors that scan its surrounding environment. Further, it is common to define a safety distance around the identified obstacles, within which the robot must remain in relation to these impediments. Formally, defining an obstacle as any impediment in the robot's path, obstacle avoidance may be defined as the task of satisfying a control objective subject to abidance by the inequality constraint $|p_r - p_{\text{obs}}| \geq d_{\text{safe}}$, where p_r is defined as the robot's position, p_{obs} the obstacle point closest to the robot's position and d_{safe} as the safety distance. This inequality must be verified for all the obstacles in the environment so that there is always a non-intersection between the robot's coordinates and the inflated obstacles (i.e., the obstacles' boundaries with the added safety distance).

First, it is important to distinguish between deliberative and reactive obstacle avoidance as different concepts. The work developed in this thesis is concerned with the latter and, as such, the algorithms presented in this section solely address this issue. Deliberative obstacle avoidance should be part of the global planner. Trajectory planning takes the map of the environment, considers the obstacles in advance, and plans around them, generating a safe route from the initial position to the target. This is normally done in a static environment, and, thus, pertains to the avoidance of static obstacles. On the other hand, reactive obstacle avoidance deals with dynamic environments, where fast responses are critical to deal with unpredictability. Data from sensors is used to generate immediate responses that guarantee the safe operation of the robot in unforeseeable scenarios.

There are multiple algorithms and methodologies pertaining to obstacle avoidance, but the common concern across all of them is to provide the robot with the ability to perform its tasks in a safe manner. The ability to deal with unpredictability, as well as doing so in a way that is fast and decisive is crucial for building trust in the robot's autonomy.

Potential fields algorithms

Potential fields algorithms, also known as artificial potential fields (APF), are one of the most popular techniques for obstacle avoidance. The idea behind it is the description of the robot's environment as a set of attractive and repulsive forces. The combination of these forces is used to create a "potential field" that is then used to safely guide the robot towards its goal. Though the way to compute the forces vary according to implementation, by definition, the target is represented as an attractive force and the obstacles as a repulsive one. A visual representation of how these forces may be computed and how they may show on a potential field can be seen in Figure 1.4. The algorithm continuously computes the forces in the environment and updates the robot's velocity accordingly. When the robot encounters

an obstacle, the repulsive force will push it away from said obstacle, allowing for smooth and risk-free navigation [23].

This is a computationally efficient method, suitable for real-time applications, and, thus, making it a popular choice for different obstacle avoidance applications, from mobile robots to cooperative UAVs, as per the works of Park et al. [24] and Sun et al. [25]. Nevertheless, not only is it hard to find a balance between the attractive and repulsive forces, so that there is a guarantee that the robot reaches its goal, but also the algorithm may get stuck in local minima. A local minimum is a point in the search space where the objective function has a lower value than all neighboring points, but is not the global minimum of the space considered. As the goal is to minimize an objective function, the solution usually gets trapped in these local points and is unable to advance and search for the globally optimal solution. To overcome this problem, an understanding of the environment as a whole is needed. Typically, this algorithm needs to be paired with a global planner, or use techniques that explore different areas of the search space for a globally optimal solution, such as random restarts or simulated annealing. Simulated annealing works by generating a new solution from a random change in the previous solution, to test if it is a better solution than the previous one. The amount of randomness in this search is controlled by a gradually decreasing temperature parameter [26].

Velocity obstacle-based algorithms

Velocity obstacle (VO) methods deal mostly with avoidance of moving, or dynamic, obstacles. They work by defining a set of unsafe velocities that would result in a collision with the obstacles in the environment, called the velocity obstacle. To achieve this, the algorithm needs the *a priori* knowledge of current positions and velocities of the obstacles in the environment and of the robot's current velocity. By drawing the cone of all possible future positions of both the robot and obstacles, it is possible to calculate

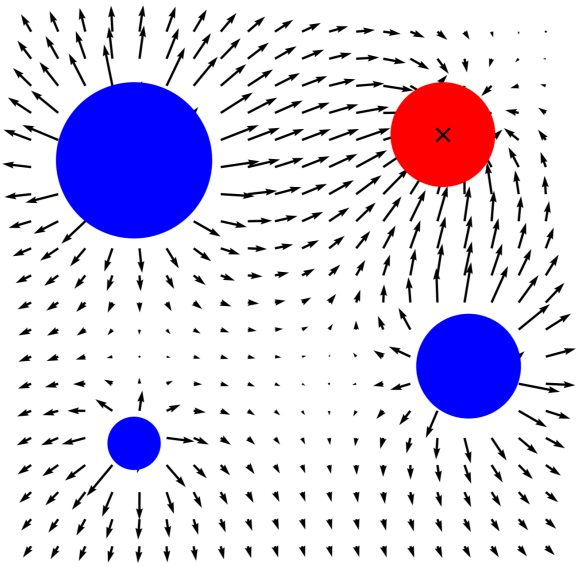


Figure 1.4: Potential field with a set of obstacles (blue) and a target (red with cross).

the intersection between them. This intersection, or velocity obstacle, as previously mentioned, is the set of all relative velocities between robot and obstacles that would result in a collision [27]. From the set of safe velocities, one is chosen and given as an input to the robot. This choice of velocity from the safe set is a problem in and of itself and several other algorithms have been proposed as a solution for this. These algorithms are, however, beyond the scope of this literature review. An illustration of a velocity obstacle based algorithm implementation can be seen in Figure 1.5, as adapted from van den Berg et. al.'s work [28].

As the previous potential fields method, velocity obstacle based avoidance is computationally efficient and allows for real-time implementation. Moreover, it is also very effective in dealing with dynamic obstacles with unpredictable or erratic patterns of movement. As a result, it has found several applications, such as in unmanned surface vehicles, unmanned aerial vehicles and even in naval ships, as per the works of Cho et al. [29], Tan et al. [30] and Shaobo et al. [31], respectively. The main disadvantage of this technique is that an optimal trajectory may not be generated from the safe velocity choice. A possible, yet complex, solution to this is in pairing the technique with another one that is capable of choosing a globally optimal solution from the set of admissible velocities.

Learning-based algorithms

Learning-based algorithms are a type of machine learning technique based on the training of an agent to avoid obstacles in the environment. It is important to distinguish between two main categories of learning-based techniques: supervised and reinforcement learning, which vary according to how the training of the model is done.

On the one hand, supervised learning uses a dataset of labeled input-output pairs. When giving the model these pairs, it can then learn to predict what the output will be, given a certain input [32]. In the context of obstacle avoidance, the model can be trained with labeled data from sensors, considering

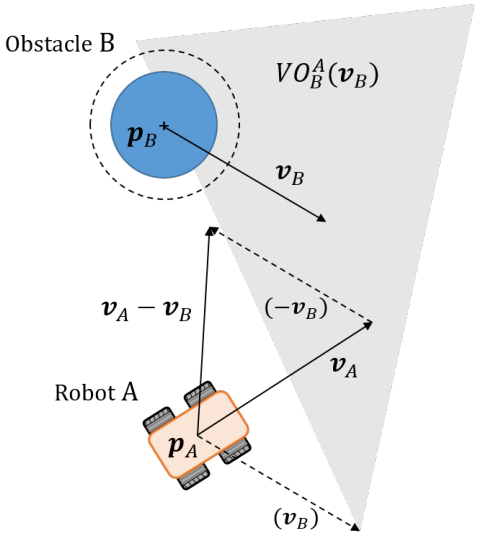


Figure 1.5: Representation of sets of unsafe velocities as part of a velocity obstacle (VO) algorithm for obstacle avoidance.

both the presence and absence of obstacles in the robot's path, with the goal of developing obstacle avoidance capabilities that can be applied to unlabeled data. On the other hand, reinforcement learning deals with trial and error. The agent in training tries different actions and observes their outcome. Feedback is then given in the form of rewards and/or penalties and the agent will learn to maximize reward and minimize penalty through its actions [33]. In this context, the robot could be allowed to try different input controls and where they take it, being penalized for every crash or for breaching the safety distance around the obstacles. It would, then, learn which steering and velocity inputs would minimize this penalty.

With a wide range of applications, especially considering the rise of autonomous vehicles, there has been a constant improvement of learning-based algorithms in recent years. Applications go from unmanned aerial vehicles (UAVs) to autonomous underwater vehicles (AUVs), as per the work of Wu et al. [34] and Bhopale et al. [35], respectively. These techniques are good at handling dynamic, unpredictable and changing environments, and can learn tasks that are hard to program manually. For example, for very complex and hard to describe vehicle models, this would be a useful technique. It would understand the outputs from determinate inputs, without the need to mathematically and accurately describe the vehicle dynamics. However, large amounts of data are needed to train the algorithm. This is both very time-consuming and expensive, even leading to work being developed in the sense of creating and labeling data [36]. Finally, these learning-based techniques, though promising, do not provide any kind of robustness guarantees.

1.3.3 Baseline solution

The currently implemented solution to the local planning problem for the AGiLE project, used as the baseline for this work, is the use of the Timed-Elastic-Band (TEB) approach. TEB is an online optimal local trajectory planner that deals with the navigation and control of mobile robots and is implemented as an extension of the navigation packages of the Robot Operating System (ROS). Considering the classification made in this section, TEB could be considered as a model-based algorithm in terms of trajectory tracking, with obstacle avoidance capabilities based on a modified APF strategy. Moreover, TEB builds on top of the Elastic-Band (EB) algorithm, where virtual "elastic bands" are used to navigate around obstacles. Intuitively, a path that connects an initial to a target position is created, with the properties of a physical elastic band. The obstacles will then exert a repulsive force on this "elastic band" and will deform the path around the blockages, either contracting or extending this path. The TEB algorithm adds temporal aspects to the EB method, by using a time-varying function to modify the shape of the path as the robot progresses towards its goal. The path is segmented and a velocity profile calculated for said segments [37, 38].

One of the main problems of both the EB and TEB algorithms, as is the case for potential field obstacle avoidance techniques in Subsection 1.3.2, is getting stuck in locally optimal solutions, rather than globally optimal ones. To illustrate this, consider the example where a path is drawn as a straight line from an initial point to a target. Then, an obstacle is introduced in the environment, moving from right

to left. As the obstacle approaches the path, it deforms left, away from the obstacle due to the repulsive force it exerts. As the obstacle proceeds, deform further and further. At a certain point, the deformed path is unnecessarily long and the globally optimal path would only be achieved if the path could "jump across" the obstacle, and either return to its original shape or deform right, depending on the size of the obstacle. One way to solve this problem is by using a series of locally optimal trajectories, done by integrating topology concepts into TEB. The algorithm can then switch between these trajectories as needed [39, 40]. Further, the TEB algorithm has also been expanded in order to adapt to car-like robots [41].

In comparison with other popular algorithms, such as the Dynamic Window Approach (DWA), TEB appears to perform slightly better for dynamic obstacle avoidance. Note that the DWA algorithm is a method that evaluates the achievable states of the robot within a dynamic window of possibilities and then chooses the optimal motion from these. TEB is fast in re-shaping its path and does not stop unnecessarily in the presence of obstacles. However, it also carries some deviation from experiment to experiment and does not draw straight paths from current position to goal. It appears to employ different strategies for different experiment iterations, thus getting different results in path planning and tracking, even if under similar circumstances [42].

Moreover, when testing, as part of the AGiLE project, a few problems arise when implementing TEB, regarding its efficient obstacle avoidance and accurate trajectory following. With respect to obstacle avoidance, TEB has trouble recalculating its path when considering dynamic environments with other robots. This is particularly evident when there is an intersection of the robot's paths or when one needs to overtake the other. In these cases, the robots usually both stop and are unable to react or one robot stops while the other one advances. These are both non-optimal solutions to these situations and, therefore, inappropriate for real-life factory implementation. The problem with the accurate trajectory following derives from the piece-wise optimization of the path as a trajectory. This leads to the robot "cutting corners" in order to optimize the path. Additionally, though the robot follows piece-wise trajectories, it globally follows a path - the difference between these two concepts having been explained in Subsection 1.3.1.

In a dynamic factory environment, a robot that deviates from its trajectory is a significant safety hazard. This digression can cause collisions and harm people, destroy equipment and material, as well as cause substantial production delays. In addition to this, following paths and not trajectories will defeat the purpose of deploying autonomy-capable robots. The trajectory optimization algorithm, upstream, guarantees the robot's safe operation by considering the trajectories of other robot's and obstacles in the environment [1]. As a result, it is imperative that the local planner does trajectory, rather than path, following.

Though TEB allows for some changes to its parameters, therefore being able to somewhat mitigate these problems, it is insufficient to guarantee the safe operation of the robots in a real-world scenario. The currently implemented version of TEB has had over a year of work in fine-tuning its parameters, yet is still unable to deal with the problems of the factory, enumerated throughout this text.

1.4 Contributions

The contributions of this dissertation are as listed below.

- Study and summary of different approaches to trajectory or path tracking and obstacle avoidance, as well as their integration.
- Development of a tunable trajectory following controller capable of static and dynamic obstacle avoidance. This tunability regards the vehicle's kinematic model, the controller's operating frequency, the cost function, the voxel cell size for obstacle avoidance constraints, the safety distance from obstacles and the replanning capabilities.
- Novel integration of minimal obstacle description through the use of voxel grids into an NMPC trajectory tracking controller framework.
- Testing of algorithm performance for a differential drive vehicle in a real factory environment for both trajectory tracking and obstacle avoidance, with validation of the vehicle model.
- Testing and comparison of TEB as a baseline solution for a differential drive vehicle in a real factory environment for both trajectory tracking and obstacle avoidance.
- A *GitHub* repository with the developed controller and full documentation on it.[1]

1.5 Dissertation outline

This dissertation is structured in chapters, as presented below. A brief summary of each chapter is also provided.

- **Chapter 2 - Background**

  In this chapter, the underlying fundamentals of the developed algorithm are explained. First, a brief introduction is given on what constitutes an optimization problem. Then, an explanation on the basis of Nonlinear Model Predictive Control (NMPC) theory is given, along with an analysis of the work that most closely relates to the approach taken in this thesis. Afterwards, voxel grids are described. An elucidation on what constitutes a solver, as well as one on the CasADi framework are also provided. Finally, a brief clarification is given on the fundamentals of the development framework used in this work.

- **Chapter 3 - Proposed approach**

  In this chapter, the problem this thesis addresses is formally put forth. Then, the kinematic vehicle model used for development and testing of the work is shown. The NMPC controller is presented, as well as the reasoning behind its design choices and the changes made to the controller after implementation in a real-world scenario. Finally, the implementation of trajectory replanning capabilities are explained.

[1] The repository can be accessed at: `https://github.com/frmadeira/nmpc_controller/`.

- **Chapter 4 - Trajectory tracking results**

  In this chapter, the trajectory tracking capabilities of the controller are studied and compared against the TEB baseline, currently implemented as a solution in the project. First, the environment used for testing shown, as well as the methodology explained. Then, a study on the validity of the used vehicle model is done. For controller tuning, a study is done regarding the influence of varying parameters such as the prediction horizon length and the weighting coefficient of the cost function and how this affects the computational efficiency of the controller. The main experimental results presented pertaining to the trajectory tracking capabilities of the developed controller are shown and compared against the baseline TEB solution. Finally, studies are conducted on both the influence of the maximum velocity given to the robot and how convergence is attained when away from the reference.

- **Chapter 5 - Obstacle avoidance results**

  In this chapter, the obstacle avoidance capabilities of the NMPC controller are studied, and compared against the TEB baseline. First, the effect of voxel cell size on the computational efficiency of the controller is studied. Then, the controller's static obstacle avoidance capabilities are studied, for different geometries of obstacles. Additionally, the dynamic obstacle avoidance capabilities of the controller are shown for scenarios in which there's an obstacle directly crossing the robot's way and for when the robot has to overtake an obstacle going in the same direction as its trajectory. Finally, the trajectory replanning capabilities of the controller are shown.

- **Chapter 6 - Conclusions and future work**

  In this chapter, all the work developed in this thesis is summarized and main conclusions are drawn. The main achievements of this work are also shown. Additionally, opportunities for further development are highlighted.

Chapter 2

Background

In this chapter, the underlying fundamentals of the developed algorithm are explained. First, a brief introduction is given on what constitutes an optimization problem. Then, an explanation on the basis of Nonlinear Model Predictive Control (NMPC) theory is given, along with an analysis of the work that most closely relates to the approach taken in this thesis. Afterwards, voxel grids are described. An elucidation on what constitutes a solver, as well as one on the CasADi framework are also provided. Finally, a brief clarification is given on the fundamentals of the development framework used in this work.

2.1 Optimization problems

An optimization problem consists on finding the best, or optimal, solution out of all of possible ones, while being subject to a given set of constraints. It typically involves the minimization of a cost function, though variations exist where the objective is maximization of a function. The latter would be equivalent to minimizing the symmetric of the cost function.

The standard form of an optimization problem is

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & e_i(\boldsymbol{x}) = 0, \quad i = 1, ..., m, \\ & g_j(\boldsymbol{x}) \geq 0, \quad j = 1, ..., p \end{aligned} \tag{2.1}$$

where

- $\boldsymbol{x} \in \mathbb{R}^n$ is the optimization variable,
- $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is the cost function,
- $e_i(\boldsymbol{x}) = 0$ is an equality constraint, and
- $g_j(\boldsymbol{x}) \geq 0$ is an inequality constraint.

When dealing with optimization problems, it is possible to differentiate between hard and soft constraints. Hard constraints must be satisfied for a solution to be considered feasible, while soft constraints

are desirably satisfied, if the cost of doing so is not too high. Violating a soft constraint leads to a feasible suboptimal solution, but violation of a hard constraint leads to infeasibility. In the particular case of this thesis, a hard inequality constraint could be that the robot maintain a distance greater than safety from an obstacle. A soft constraint, on the other hand, could be a penalty term for distance to the reference trajectory, introduced as a term of the cost function.

2.2 Nonlinear Model Predictive Control

Nonlinear Model Predictive Control (NMPC) is an advanced model-based control technique for the feedback control of a nonlinear system subject to a set of constraints [43]. The typical structure is the same as for the linear case, illustrated in Figure 1.3, with the only difference being that the system model has nonlinear dynamics. The NMPC technique uses this mathematical model to predict and optimize the future behaviour of the system, by taking control actions accordingly. The definition of the model is often obtained theoretically, through knowledge of the physics of the system, or empirically.

The future states of the system are calculated with the system model based on the current vehicle states and the computed control inputs. Using the robot's current state as an input allows to account for the deviation from reference and, thus, derive a closed loop feedback control law. The integration of an NMPC controller is similar to that of conventional controllers, differing solely on the use of a combination of prediction and optimization, rather than precomputed control laws.

The operating principle of a basic NMPC controller is illustrated in Figure 2.1 and is described sequentially below, adapted from the definition given by Grüne and Pannek [43].

At each time step, defined as the discretized time intervals of the problem, t_s, $h = 0, 1, 2, ...$

- Measure the current state of system $\boldsymbol{x}(h) \in \mathbb{X}$, where $\mathbb{X}$ is the set of allowable system states.
- Define the current state as the initial condition $\boldsymbol{x}(h) := \boldsymbol{x}_0$ and solve the optimal control problem (OCP) that pertains to the given scenario. This OCP would appear in the form of a constrained optimization problem, as

$$\begin{aligned} \text{minimize} \quad & J_N(\boldsymbol{x}_0, \boldsymbol{u}(.)) := \sum_{k=0}^{N-1} l(\boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \\ \text{with respect to} \quad & \boldsymbol{u}(.) \in \mathbb{U}^N(\boldsymbol{x}_0) \\ \text{subject to} \quad & \boldsymbol{x}_u(0, \boldsymbol{x}_0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}_u(k+1, \boldsymbol{x}_0) = f(\boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \end{aligned} \tag{2.2}$$

  where function f represents the system dynamics and function $l : \mathbb{X} \times \mathbb{U} \to \mathbb{R}_0^+$ is the cost function. The cost function, which penalizes, for example, distance to reference, is minimized over the finite prediction horizon N with respect to $\boldsymbol{u}(.) \in \mathbb{U}^N(\boldsymbol{x}_0)$, where $\mathbb{U}^N$ is the set of allowable control inputs over the prediction horizon and $\boldsymbol{u}(.)$ is the control inputs vector. The prediction horizon N is defined as the time duration over which the future behaviour of the system is predicted.

- Define the NMPC-feedback value as $\mu(\boldsymbol{x}(h)) := \boldsymbol{u}^*(0)$ and apply the optimal control to the robot. The remaining values of $\boldsymbol{u}^*$ can be discarded, or used to warm-start the optimization problem at the next time step $t_s = h + 1$. This warm-start procedure consists on providing the solver with an initial guess closer to the solution and, as such, increases its computational efficiency, reducing total solver time. This procedure is explained in detail in Subsection 2.5.2.

A note should also be made on the difference between prediction horizon N and control horizon N_c, here assumed to take equal values. The control horizon dictates the number of optimized control actions, while the prediction horizon dictates prediction depth. If the former is smaller than the latter, the last computed input is kept constant for the remaining length of the control horizon [44]. Moreover, note that though it is mathematically correct to consider that optimization is made with respect to the set of control inputs $\boldsymbol{u}(.) \in \mathbb{U}^N(\boldsymbol{x}_0)$, usually direct collocation methods are used, for which the states are also considered as optimization variables. These direct collocation methods are further explained in Section 2.5.

A list of advantages and disadvantages of an NMPC controller over conventional controllers is shown in Table 2.1. For the reasons presented, and considering the characteristics of the problem this thesis addresses, this control technique was chosen for this work.

2.3 Related work

In this subsection, some related work is analyzed regarding both its approaches to the problem at hand and its shortcomings when approaching it as a whole. The work reviewed in this subsection is the one that most closely relates to the work of this thesis, i.e., work that incorporates trajectory tracking and obstacle avoidance into a Nonlinear Model Predictive Control framework. Some clarification is also provided as to how obstacle avoidance is achieved in each of these approaches.

Some authors resort to coupling obstacle avoidance strategies that have been previously explained in this thesis into an NMPC framework. This is the case for the work developed by Hamid et al. [45], in which artificial potential fields (APF) are used as a collision avoidance strategy. These APF feed yaw

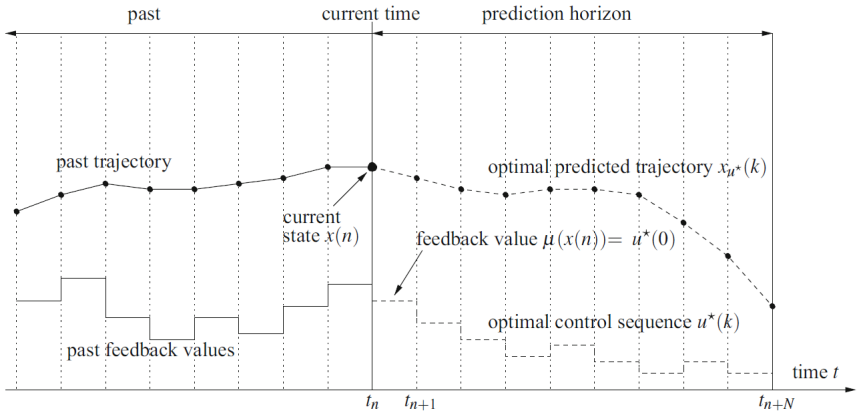


Figure 2.1: Operating principle of an NMPC controller. (Sourced from: Grüne and Pannek [43])

and deceleration rates to the main NMPC controller, responsible for navigation, when in the presence of an obstacle. The work finds that this strategy provides reliable collision avoidance when in hazardous scenarios and further implements a move-blocking strategy, i.e. keeping the optimal solution unchanged for a longer control horizon, in order to reduce the computational burden typical of this control technique.

While solutions that target both path and trajectory tracking problems seem to be somewhat standard when considering Nonlinear Model Predictive Control strategies, most of the work regarding the introduction of obstacle avoidance capabilities into an NMPC seems to rely on introducing either hard or soft constraints into the definition of the central Optimal Control Problem (OCP). The difference between both these constraints is explained in Section 2.1.

When obstacle avoidance is incorporated into an NMPC controller as a soft constraint, it means that a term is added to the cost function. This is the case of the work developed by Abbas et al. [46], that for a vehicle position (x, y) and an obstacle position (x_o, y_o) incorporates the differentiable point-wise repulsive function, that describes the inverse of the distance to the obstacle, into the cost:

$$P_k = \frac{1}{(x - x_o)^2 + (y - y_o)^2 + \epsilon}, \tag{2.3}$$

in which ϵ is a value different than zero, in order to avoid singularities. Though only tested for linear and point obstacles, this work shows promise in steering autonomous vehicles away from collisions, depending on the slope of the repulsive function in 2.3. Further, the work developed by Gaertner et al. [47] has found success in including dynamic obstacles into the receding horizon planning for a scenario of four-legged autonomous robots. Though having to use simplistic cylindrical representations of moving obstacles, this example sets a great computational cost benchmark.

As an expansion of the work done in obstacle avoidance through the introduction of soft constraints into an NMPC controller framework, it is possible to cite both the articles of Park et al. [48] and Kamel et al. [49]. Park et al. introduce the concept of a parallax-based system, through which it is possible to con-

Table 2.1: Advantages and disadvantages of an NMPC controller over conventional controllers.

Advantages	Disadvantages
Ability to handle nonlinear systems. Conventional controllers are usually designed to work only with linear systems, unlike the NMPC controller.	**Sensitive to unmodelled dynamics**. Though naturally robust, the NMPC controller could be sensitive to unmodelled vehicle dynamics.
Optimality. The NMPC controller uses an optimal solution tailored to both the cost function and constraints of the use case considered.	**High computational expense**. The computational cost for an NMPC controller is quite high, especially for long prediction horizons and/or higher number of constraints.
Constraint handling. Ability to handle multiple constraints. Also, these are easily implemented for the specifics of the problem.	**Tuning**. The usually higher number of control parameters due to the model and constraints means that an NMPC controller could be harder to tune.
Multivariable handling. Extension to multivariable case is straightforward and easily handles multiple inputs and outputs.	

sider both the moving direction and dimensions of an obstacle in an explicit manner. This parallax-based method claims to have advantages over distance-based ones. The work developed by Kamel et al. is able to introduce obstacle trajectory and motion prediction into the collision avoidance cost. Moreover, it is able to account for the uncertainty over the prediction horizon due to delay in communication when multiple agents, or robots, are involved.

There have also been some publications on this integration of obstacle avoidance capabilities through the use of hard, rather than soft, constraints. These constraints must be satisfied for a solution to be considered feasible, as is the case of the work developed by Sani et al. [50], in which a bug-type algorithm forces the robot to move tangentially to the surface of the obstacle and then return to the original path after completely dodging the obstacle. Using LIDAR sensors to detect the obstacle's position $(x_{\text{obs}}, y_{\text{obs}})$ and diameter d_{obs}, and given the robot's position (x_r, y_r), the hard constraint is defined as:

$$\sqrt{(x_r - x_{\text{obs}})^2 + (y_r - y_{\text{obs}})^2} \geq (r_{\text{obs}} + r_r), \tag{2.4}$$

where r_{obs} and r_r are, respectively, the radius of the obstacle and of the robot. Note that this constraint does not distinguish avoidance from static or moving obstacles and that the considered obstacle position is not necessarily its center, but rather the closest point detected by the sensor. Further work has also been developed regarding the introduction of the obstacle's trajectory into the obstacle avoidance constraint definition, as is the case of the work developed by Lindqvist et al. [51]. In this, spherical obstacles are considered to have either a linear or projectile motion or are considered to be static. This trajectory is fed into the NMPC as an extra input, done from an initial condition. Obstacle avoidance is ultimately achieved as an equality constraint that considers the parameterized trajectory of the spheres. Both of the works discussed in this paragraph have found success in dealing with dynamic obstacles.

Finally, it is important to refer to other work, in which the constraints are relaxed. In this case, though the obstacle avoidance capabilities of the controller are set into the NMPC framework as hard constraints, they are ultimately cast as soft constraints in order not to lead to infeasibility, i.e., to maintain convexity of the state and output spaces. In the case of the work developed by Sanchez et al. [52], though only considering fixed obstacles, an innovative introduction of an auxiliary trajectory is done so that it maintains feasibility of the successive optimization problems. Thus, avoiding issues even when the reference curve is infeasible. Further work is developed by Sathya et al. [53], in which a novel modelling framework is introduced to deal with obstacles that are mathematically described as intersections of a finite number of strict nonlinear inequalities. This is, the obstacles are either defined as quadratic constraints or either polyhedral or polynomial ones. Further, this work uses PANOC, a novel way to solve NLP problems first introduced by Stella et al. [54], and that is said to outperform common solvers.

Though all the work cited in this subsection is innovative in introducing obstacle avoidance capabilities into a path or trajectory tracking NMPC controller, most work is not able to do so in a way that is agnostic to the obstacle's format, position and trajectory. Abbas et al.'s work is only tested for obstacles described as either linear or points and fails to be able to meet the minimum computational requirements for online implementation. Both Gaertner et al.'s and Lindqvist et al.'s work must consider simplistic rep-

resentations of obstacles as, respectively, cylindrical or spherical shapes. Furthermore, the latter must characterize the trajectory into some pre-defined motion in order to have it as an input to the controller, as is the case for Kamel et. al's work. In addition, both Sani et al.'s and Sathya et al.'s work require information regarding, respectively, the dimensions of the obstacle or its mathematical description. Finally, most work lacks experimental tests, as is the case for Park et al.'s, Sanchez et. al's and Hamid et al's articles, which are only tested for simulated environments and never in a real-world scenario.

In conclusion, none of the related work is able to develop obstacle avoidance capabilities on top of an NMPC tracking controller framework in a way that is both independent of the shape, size and movement of the obstacles in the environment and that is tested in a real-world scenario. There is a gap for development of a controller to be tested in real-world conditions and that has no need for previous information on the obstacle from which it takes the needed avoiding actions.

2.4 Voxel grids

Voxel grids, short for volumetric pixel grids, are three-dimensional structures that organize volumetric data. Essentially, they can be thought of an extension of an image's representation through pixels. Depending on the required application, each cell, or voxel, stores information about a property or attribute and thus a voxel grid is able to discretize and represent a continuous property.

The voxel grid is defined by its resolution, which determines both the size and number of cells along each dimension. A higher resolution allows for a finer representation of the considered property or attribute, but also requires greater computational resources. The voxels can be uniform throughout the grid or vary, depending on the requirements of the application. A trade-off must be considered between how coarse a grid can be, for computational efficiency, while still presenting a true representation of the property considered.

A particular use of voxel grids is the representation of obstacles in space, through occupancy mapping. If a voxel cell contains any part of an obstacle, it is marked as occupied. If it does not, it is marked as free. The group of occupied voxels defines the obstacle in space. This can also be simplified and used in a two-dimensional space, by disregarding the height coordinate. In Figure 2.2, based on Zhou et al.'s work [55], the voxelization of a three-dimensional rabbit model is shown for different resolutions.

2.5 Numerical solvers

In the context of Nonlinear Model Predictive Control, a numerical solver is defined as a method or algorithm needed to solve the Optimal Control Problem formulated as part of the NMPC control scheme. Solvers use numerical methods to search for the optimal control inputs, this is, the ones that minimize a given cost function and satisfy both the dynamics of the system and defined set of constraints. According to Kelly [56], there are three techniques for solving an optimal control problem: dynamic programming, indirect and direct methods.

Dynamic programming solves the Hamilton-Jacobi-Bellmann equations over the entire state space. These equations give necessary and sufficient conditions for optimality in control with respect to a cost function. It is effective if for unconstrained low-dimensional systems, however, not for higher-dimensional ones.

Indirect methods, also know as “optimize-then-discretize” methods, construct the necessary and sufficient conditions for optimality in an analytical manner. These are, then, discretized and optimized. It is possible to achieve more accurate solutions using this method, at the expense of increased complexity for construction and solution.

Finally, direct methods, also know as “discretize-then-optimize” methods, discretize the continuous-time optimal control problem, converting it into a Nonlinear Programming Problem (NLP). An NLP is an optimization problem, as shown in Section 2.1, with a nonlinear objective function and/or a feasible region determined by nonlinear constraints. These methods, usually do not require an analytical analysis of the problem and are the best suited for the use case considered in this thesis. This is the method used for this thesis.

2.5.1 Discretization

Implementing the NMPC controller requires the formulation of an Optimal Control Problem (OCP) into an NLP. The fundamental difference between the two being that an OCP deals with systems described by differential equations and involves finding the optimal control inputs, while an NLP deals with finding the optimal values of decision variables subject to nonlinear constraints. To do this, the continuous-time kinematic model of the system, in this case, vehicle, must be discretized into a finite set of time intervals. Then, a finite-dimensional optimization problem is constructed and solved.

The single shooting method, a numerical technique used to solve optimal control problems and applicable in both direct and indirect formulations, approximates the trajectory the robot will follow using a simulation. A set of parameters are chosen and then simulated to check if the goal is reached. The robot’s route is represented as a single segment, with a single constraint, in which the trajectory’s fi-

(a) Original rabbit model. (b) Model for resolution of $0.4\,\mathrm{mm}$. (c) Model for resolution of $1\,\mathrm{mm}$.

Figure 2.2: Voxelization of a rabbit model for different resolutions. (Sourced from: Zhou et al. [55])

nal configuration must coincide with the goal configuration. This simplification is useful when control is simple and there are few path constraints. However, the relation between decision variables, objective function and constraints is usually highly non-linear and the approximation made by this model is insufficient in guaranteeing the needed accuracy.

Moreover, multiple shooting is an advanced technique that enhances the single shooting method by breaking down a trajectory into smaller segments and applying the single shooting approach to each of these segments. This approach results in a more reliable discretization, making it particularly effective for solving intricate problems. As these segments become shorter, the connections between decision variables, the objective function, and constraints become more linear, leading to greater algorithm accuracy [57].

Additionally, collocation methods approach discretization by approximation through use of polynomial splines (i.e., a function that is defined piecewise by polynomials). The denomination of these methods derives from the act of strategically choosing the points within the considered domain to solve for the problem. Two common collocation techniques are the trapezoidal collocation and Hermite-Simpson collocation. The former approximates the objective function as piecewise linear functions, using trapezoidal quadrature and the latter as piecewise quadratic functions. Further, orthogonal collocation uses higher order polynomials, with the added advantage that each segment might be represented by a different order polynomial [58].

Considering the need for online implementation, as part of the NMPC controller, the choice of discretization method should favour options that guarantee greater computational efficiency. For a problem with a great number of constraints, as is the case with the problem this thesis addresses, it usually proves more efficient to solve a larger sparse non-linear problem than a smaller, denser program. This is, collocation methods are chosen over shooting methods.

Trapezoidal collocation

Considering both the closed feedback nature of the controller and its operating frequency, the benefit in reduced computational cost from the choice of a simpler method outweighs the drawback in accuracy loss. This means, the kinematic model of the vehicle, which describes its dynamics, should be discretized using trapezoidal collocation. The basis for this collocation is the use of trapezoidal quadrature. In order to integrate

$$Y'(x) = f(x, Y(x)) \tag{2.5}$$

over the interval $[x_n, x_{n+1}]$:

$$Y(x_{n+1}) = Y(x_n) + \int_{x_n}^{x_{n+1}} f(x, Y(x))dx, \tag{2.6}$$

the trapezoidal quadrature rule is applied to the integral, producing:

$$Y(x_{n+1}) = Y(x_n) + \frac{h}{2}[f(x_n, Y(x_n)) + f(x_{n+1}, Y(x_{n+1}))] - \frac{h^3}{12}Y'''(\xi_n), \tag{2.7}$$

where h is the step size considered. The trapezoidal rule is expressed by dropping the error term:

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]. \tag{2.8}$$

This trapezoidal rule will be used to discretize the vehicle's dynamics model, given as a set of differential equations.

2.5.2 Initial guess

A numerical optimization solver is particularly sensitive to its first iteration. The probability of solver success and its efficiency, are directly correlated to how close the initial values for the variables are from their optimal solution. A poor initial guess, i.e., a poor initialization of the solver's internal variables is time-consuming, leading to a greater number of iterations to reach the solution. In some cases, it could even lead to infeasibility. As such, the distinction is made between warm-starting and cold-starting a solver.

A cold-start means that the solver initiates its internal variables either randomly or using default values, without any prior information about the solution. A warm-start would provide the solver with some prior knowledge about the solution when initializing. To do this, an initial guess is usually provided, hence not initializing variables at random. This ultimately allows for a great reduction of computational time, especially when dealing with large optimization problems.

In the case of an NMPC controller, the optimal controls found at a given time step could be used to initialize the solver at the successive step, rather than being discarded. Given the prediction at time step t_n over the prediction horizon N, an initial guess can be provided at time step t_{n+1} for the optimal control inputs of the set $[t_{n+1}, t_{n+2}, ..., t_{n+N}]$. This guess should be close to the actual solution, as, barring any unmodelled environment or vehicle dynamics, such as wheel friction or air resistance, the predictions are approximate to the actual behaviour through the use of the vehicle's model. This means, the solver will be able to store the previous optimization variables and only solve for the last interval of the prediction horizon $t_{(n+1)+N}$, resulting in a computationally efficient solution.

2.5.3 CasADi

CasADi is an open-source tool for numerical optimization and optimal control in particular. In the use case considered, CasADi is particularly relevant as it allows to efficiently formulate and solve non-linear programming problems (NLPs), through the formulation and manipulation of expressions using its symbolic framework. Alike the use of variables in algebra, CasADi is able to store variables as representative symbols. This tool was chosen as it is efficient and versatile, with a wide range of successful implementations, across different fields. Additionally, as of early 2016, it is considered to be in maturity, having been further developed for efficiency purposes only [59].

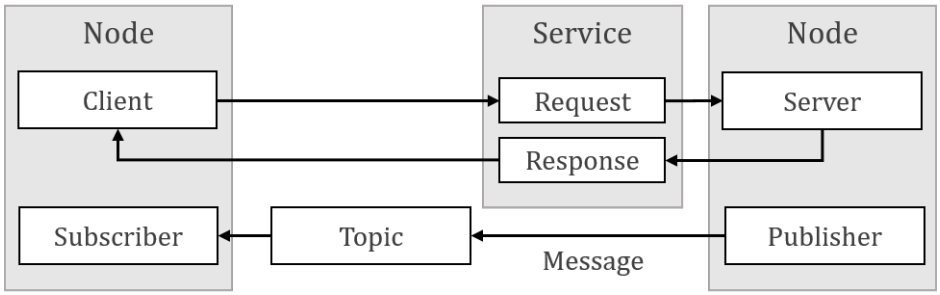


Figure 2.3: Example of the communication framework of ROS.

2.6 Robot Operating System

Robot Operating System (ROS) is an open-source software framework for robot software development that provides operating system-like functionality [60]. It provides a set of tools and services that enable developers to build and control complex robot systems. Essentially, ROS is a way to establish communication between the robot's processes.

The processes, or executables, that use the ROS framework are called nodes. These communicate with other nodes through the use of topics. Topics work using a publish/subscribe methodology, thus decoupling the production of information from its consumption. A node can send information to a topic, by publishing to it, and it can receive information from a topic, by subscribing to it. The information's structure is defined by the type of message. As an example of this communication, in the case of this work, the control inputs must be passed to the robot through a publisher specific to the velocity commands. Moreover, a service is a mechanism for a node to send a request to another node and receive a response from it in return. It works in a request/response manner from client and server. An illustration of communication between ROS nodes can be seen in Figure 2.3. Finally, the denomination of ROS package will be used throughout this document to refer to any collection of software or configuration files that perform a specific task or have a certain functionality in ROS.

Chapter 3

Proposed approach

In this chapter, the problem this thesis addresses is formally put forth. Then, the kinematic vehicle model used for development and testing of the work is shown. The NMPC controller is presented, as well as the reasoning behind its design choices and the changes made to the controller after implementation in a real-world scenario. Finally, the implementation of trajectory replanning capabilities are explained.

3.1 Problem statement

Consider a robot $\mathcal{R}$ that moves in a two-dimensional (2-D) environment, $\mathcal{E} \in \mathbb{R}^2$ occupied by a set of M obstacles, $\mathbb{O} = \{\mathcal{O}^1, ..., \mathcal{O}^M\}$, both static and dynamic. The set of all possible states of the robot is defined by the state space $\mathbb{X}$ and the set of allowable control inputs is defined by $\mathbb{U}$. Define $\mathbb{X}_{\text{free}}$ as the set of states that satisfy the global constraints. Additionally, the constraints of the vehicle model are expressed as a differential state model or transition equation:

$$\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{3.1}$$

defined for every state $\boldsymbol{x} \in \mathbb{X}$ and control input $\boldsymbol{u} \in \mathbb{U}$. Further, define time step h as a time increment for discretization of the continuous trajectory duration T and consider a feasible trajectory, given as a reference set of admissible states for each time step h, such that $\boldsymbol{x}_{\text{ref}}(h) \in \mathbb{X}$. The trajectory is drawn from robot $\mathcal{R}$'s initial state $\boldsymbol{x}(t_i) \in \mathbb{X}$ to a goal state $\boldsymbol{x}(t_g) \in \mathbb{X}$, and can, thus, be written as $\pi : [t_i, t_g] \rightarrow \mathbb{X}_{\text{free}}$.

For every time step h, it is necessary that an optimal set of controls $\boldsymbol{u}(h) \in \mathbb{U}$ is found and passed to the robot. The objective of this set of controls $\boldsymbol{u}^{\text{opt}}(h) \in \mathbb{U}$ is the minimization of the deviation from the reference state at the time considered. Further, it should consider any needed avoiding action for every obstacle $\mathcal{O}^i \in \mathbb{O}$ in the reachable state space, the coordinates in the environment the robot can reach given its constraints, such that the robot maintains a safety distance d_{safe} from every obstacle in the environment. Finally, a maximum deviation d_{replan} from the trajectory reference should be guaranteed, such that deviations greater that d_{replan} trigger automatic trajectory replanning, as a safety measure.

3.2 Kinematic vehicle model

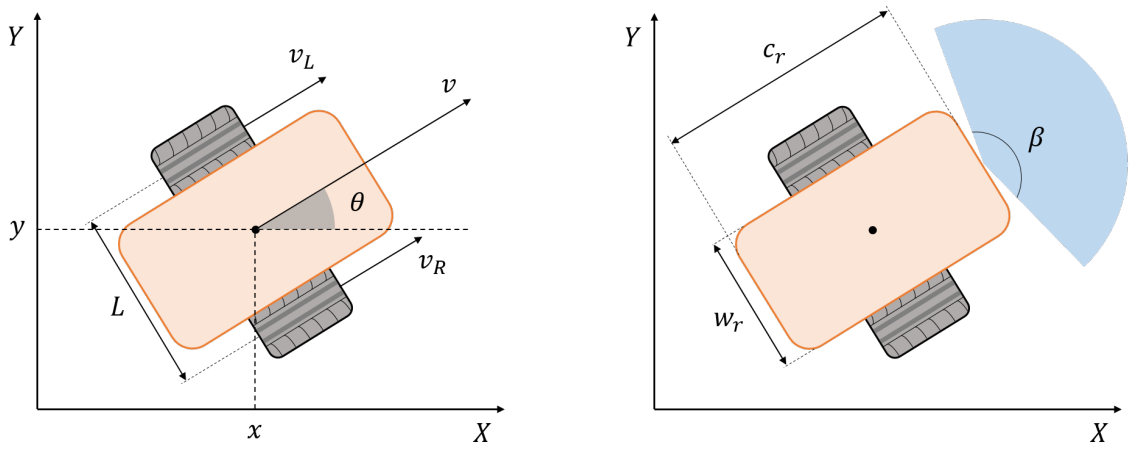
Without loss of generality, the controller is experimented for a differential drive vehicle. Nevertheless, it can be implemented for any vehicle model, given as a set of differential equations. These vehicle dynamics is given as a kinematic model. This is, the motion of the vehicle is described disregarding the forces and torques that might have caused said motion. This model is considered to be sufficient for the use case considered, as the motion is constrained to the trajectory following purpose of the controller. A more complex kinodynamic model would have to be considered, if not for the development of this controller. This model would include the dynamics of the vehicle, accounting for its mass, moments of inertia and frictional forces. For the implementation in this work, and taking the closed-feedback nature of an NMPC controller into account, the fact that the trajectories are smooth and not high velocity and the regularity of the surface in which the robot moves, the benefit in reduced computational expense from the use of the kinematic vehicle model far outweighs the drawback in accuracy loss from it.

The robot considered is a differential drive vehicle with two independently actuated wheels. These wheels sit on a common axis, separated by a wheel track, or track width, of L. In Figure 3.1, the geometry of a differential drive vehicle with two controllable wheels is shown. Alongside this, the vehicle's height c_r and width w_r, as well as its sensor angular range β are also represented. The planar depiction considered is appropriate since the modelling process is done in accordance with a 2-D perspective of motion for the vehicle.

The vehicle's state vector is given as $\boldsymbol{x} = (x_r, y_r, \theta_r, v_{R_r}, v_{L_r}) \in \mathbb{X}$, where x_r and y_r are the robot's cartesian coordinates in the world frame, θ_r its orientation with respect to the x-axis and v_{R_r} and v_{L_r} are the linear velocities of the right and left wheels, respectively. The set of acceptable, or reachable, robot states is given as $\mathbb{X}$. When considering this set of reachable states, it is important to note the nonholonomic constraints of a differential drive robot. The constraints on the movement of the system cannot be described as a set of differential equations that only consider the positions and velocities of the system. The motion of the system depends on its past states. In reality, the nonholonomic nature of the differential drive vehicle means that the possible motions of the robot are limited, as the vehicle's velocity and orientation are not independent from each other. For example, a differential drive vehicles cannot move sideways or rotate while moving in a straight line. Rotational and translational movements are coupled, and the controller must account for that.

Finally, the kinematic model of the vehicle is given by the following equations of motion

$$\begin{aligned} \dot{x}_r &= v\cos\theta, \\ \dot{y}_r &= v\sin\theta, \\ \dot{\theta}_r &= \omega, \\ \dot{v}_{R_r} &= a_R, \\ \dot{v}_{L_r} &= a_L, \end{aligned} \tag{3.2}$$



(a) Velocities, orientation and wheel track.

(b) Sensor range and physical dimensions.

Figure 3.1: Representation of the vehicle parameters used in this work.

where v is the vehicle's linear velocity, given as the average between the right and left wheels' velocities

$$v = \frac{v_R + v_L}{2} \tag{3.3}$$

and ω is the vehicle's angular velocity, through the difference of the right and left wheels' velocities

$$\omega = \frac{v_R - v_L}{L}. \tag{3.4}$$

Variables a_R and a_L represent the linear acceleration of the right and left vehicle wheels, respectively. Thus, the control vector is defined as $\boldsymbol{u} = (v, \omega, a_R, a_L)^T \in \mathbb{U}$, where $\mathbb{U}$ is the set of allowable controls.

The set of allowable controls are limited by the physical constraints of the vehicle considered. In this case, both the vehicle's wheels' velocities and accelerations are bounded by a maximum value:

$$\begin{aligned} |a_R| &< a_{\max}, \\ |a_L| &< a_{\max}, \\ |v_R| &< v_{\max}, \\ |v_L| &< v_{\max}. \end{aligned} \tag{3.5}$$

For safety, the robot is not allowed to perform reverse maneuvers, as its laser can only scan the environment in front of the robot. As such, its linear velocity cannot take negative values:

$$v \geq 0. \tag{3.6}$$

3.3 Trajectory input

The algorithm takes as an input a feasible trajectory represented by a set of N collocation points. Each of the N collocation points is characterized by the robot's state, with an associated time stamp.

Assuming the robot's initial position at $t_0 = 0$, the time stamp represents the time elapsed between the initial state and the specific time at which the robot should be in the corresponding state. Further, each collocation point is represented by a set of suboptimal controls. Considering a trajectory from t_0 to t_N, this set is given as

$$\text{trajectory} = [x_{r_0}, y_{r_0}, \theta_{r_0}, t_0, v_{R_0}, v_{L_0}, a_{R_0}, a_{L_0}, ..., x_{r_N}, y_{r_N}, \theta_{r_N}, t_N, v_{R_N}, v_{L_N}, a_{R_N}, a_{L_N}]. \tag{3.7}$$

In order to match the controller's frequency, these values must be interpolated every time that the time step - the difference between two consecutive time stamps - does not equal the operating time step of the controller. This operating time step is defined as the inverse of the controller's operating frequency. This need for an interpolation, when considering the trajectory's discretized points and the controller's operating frequency is better shown in Figure 3.2. To carry out the interpolation, *SciPy*, a *Python* library, is used. Particularly, the *interpolate* package [61].

With this package, interpolation is possible through either linear or cubic methods. A study was made on both the difference in computational efficiency and interpolation accuracy between the two processes. In Figure 3.3, 50 different trajectories were calculated using the trajectory generation algorithm, interpolated to match a controller frequency of $f_c = 5\,\text{Hz}$ and categorized according to the total duration of the generated trajectory, in seconds. The interpolation is considered to be independent of the shape of the trajectory. Both the absolute value for errors in x and y are calculated as the absolute value of the difference between the cubic interpolation and linear interpolation solutions. The total time to interpolate the given trajectory is calculated and the difference given, again, between the cubic interpolation and linear interpolation. This comparison experiment was done on a computer system equipped with an *Intel Core i7-7500U CPU (2 cores, 2.70 GHz)*, 8.00 GB of RAM, and two CPUs: an *Intel HD Graphics 620* and a *NVIDIA GeForce 940MX*, running *Windows 10* as an operating system. The code ran on *Python 3.10.11*.

There appears to be a weak or no correlation between the maximum absolute error in x and y and the trajectory's total duration. This absolute error is always below a value of 5×10^{-3} meters and, as such, can be considered as negligible. The computational efficiency, however, appears to favour the linear interpolation as the trajectory's duration increases. The real-world scenario trajectories are expected to be longer than the ones calculated in simulation and, for that reason, the linear interpolation was chosen for this work. For convenience, after the interpolation, the variables at each time step are separated and grouped into arrays, respectively, $\boldsymbol{x}_{\text{ref}}$, $\boldsymbol{y}_{\text{ref}}$, $\boldsymbol{\theta}_{\text{ref}}$, $\boldsymbol{v}_{\boldsymbol{R}\text{ref}}$, $\boldsymbol{v}_{\boldsymbol{L}\text{ref}}$, $\boldsymbol{a}_{\boldsymbol{R}\text{ref}}$ and $\boldsymbol{a}_{\boldsymbol{L}\text{ref}}$. The time stamps are used solely as a tool to interpolate and, so, are not represented in an array format.

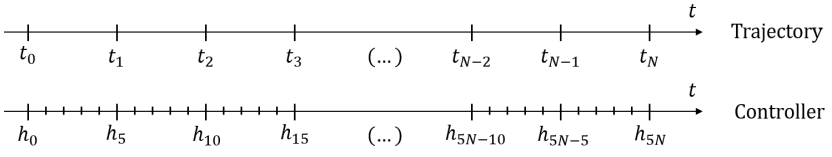


Figure 3.2: Interpolation between the discretized trajectory and controller's operating frequency.

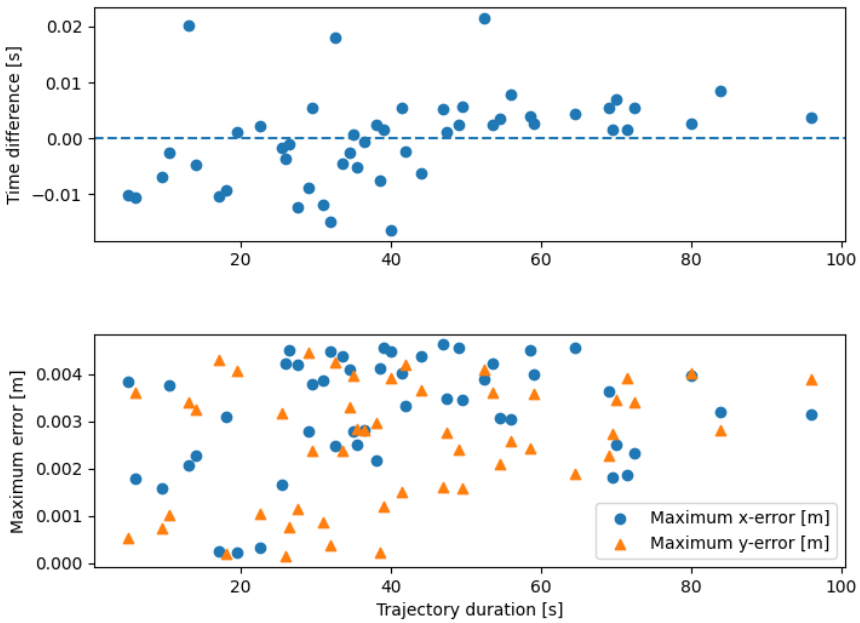


Figure 3.3: Comparison between linear and cubic interpolation for *SciPy*.

Addressing compatibility between the trajectory generation algorithm used and the work presented in this thesis, it is important that the trajectory generation and optimization is done using a nominal velocity v_{nom}, lower than the robot's actual maximum velocity v_{max}. The trajectory tracking controller has to have the ability to catch up if it lags in regards to the reference. If the trajectory is generated considering the vehicle's maximum velocity, any small perturbation, oscillation or obstacle, frequent in a real-world scenario, will lead to the robot permanently trailing behind the reference trajectory. This accumulation in trailing error eventually leads to catastrophic failure.

3.4 Optimal control problem

Considering the nature of the problem considered, namely the existence of both global and local constraints, the optimal control framework is, according to Betts [58], the best suited to guarantee accurate trajectory following. This optimal control framework is considered at the core of the Nonlinear Model Predictive Control strategy, explained in detail in Chapter 2. Thus, the primary focus of this work is the description of trajectory tracking as an optimization problem. This is, defining trajectory tracking as the optimization of a specified performance, subject to a set of constraints. Additionally, this work will focus on the implementation of reactive obstacle avoidance as part of the optimization problem.

First, it is important to note the distinction between global and local constraints. Global constraints are defined as constraints that are applicable to the trajectory as a whole, hence, that must be satisfied throughout all of the trajectory's duration. This is the case, for example, of the obstacle avoidance constraints. Local constraints, on the other hand, are specific to individual robot states or trajectory

points. As an example of these constraints, consider the robot's maximum wheel velocity $v_{\max}$. The solution to the problem should account for both of these limitations.

Remembering the robot's state vector $\boldsymbol{x} = (x_r, y_r, \theta_r, v_R, v_L) \in \mathbb{X}$, where $\mathbb{X}$ is the set of reachable vehicle states, the robot's control vector $\boldsymbol{u} = (v, \omega, a_R, a_L)^T \in \mathbb{U}$, where $\mathbb{U}$ is the set of allowable controls and given $\boldsymbol{x}_0$ as the robot's initial state, then, at each time step h, the optimal control problem is defined as

$$\begin{aligned}
\text{minimize} \quad & J_N(h, \boldsymbol{x}_0, \boldsymbol{u}(.)) := \sum_{k=0}^{N-1} l(h+k, \boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \\
\text{with respect to} \quad & \boldsymbol{u}(.) \in \mathbb{U}^N(\boldsymbol{x}_0), \quad \boldsymbol{x}_u(.) \in \mathbb{X}^N(\boldsymbol{x}_0) \\
\text{subject to} \quad & \boldsymbol{x}_u(0, \boldsymbol{x}_0) = \boldsymbol{x}_0 \\
& \boldsymbol{x}_u(k+1, \boldsymbol{x}_0) = f(\boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \\
& v_u(k, \boldsymbol{x}_0) \geq 0 \\
& \boldsymbol{d}^2_{r \to obs}(k, j) \geq d^2_{\text{safe}}, \forall j \in [1, 2, ..., M],
\end{aligned} \tag{3.8}$$

where $J_N(h, \boldsymbol{x}_0, \boldsymbol{u}(.))$ is the cost function and optimization is made over the control horizon N in relation to both the vehicle's control inputs and states. The component of the cost function at each time step over the prediction horizon is $l(h+k, \boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k))$. Variable M represents the number of obstacle points detected at each time step. The constraints that the problem is subject to pertain, respectively, to the initial position of the robot, the vehicle dynamics, the requirement of non-negative velocity and obstacle avoidance. A more detailed explanation on each of the parts that constitute the optimization problem is given subsequently. It is important to note the distinction between $\boldsymbol{x}_u$, which represents the optimized set of vehicle states, and x_{r_u}, which solely represents the optimized x-coordinate for the vehicle.

3.4.1 Variable bounds

The set of admissible controls are subject to the robot's physical constraints. This is, the feasible region of control inputs is bounded by both the maximum and minimum velocities and accelerations of both vehicle wheels. Bearing in mind the definition of the set of control inputs as $\boldsymbol{u} = (v, \omega, a_R, a_L)^T$, then from 3.3, 3.4, 3.5 and 3.6, the equations that define the set of admissible control inputs are

$$\begin{aligned}
0 \leq v \leq v_{\max}, \\
-\frac{2v_{\max}}{L} \leq \omega \leq \frac{2v_{\max}}{L}, \\
-a_{\max} \leq a_R \leq a_{\max}, \\
-a_{\max} \leq a_L \leq a_{\max}.
\end{aligned} \tag{3.9}$$

Note that both the values for the maximum velocity $v_{\max}$ and acceleration $a_{\max}$ will be given due to either tracking efficiency and safety reasons or physical limitations of the robot. The values that bound the angular velocity are a consequence of its definition, shown in 3.4.

On the other hand, the set of admissible or reachable vehicle states is not as straightforward to define. The nonholonomic nature of these constraints, which has been addressed in Section 3.2, means

that the future states depend on past states. Notwithstanding, these states are defined by the vehicle kinematics and, as such, do not require that additional boundaries are set.

3.4.2 Cost function

The cost function is responsible for describing the system performance that is to be optimized. As such, at every point over the prediction horizon, it is calculated as

$$l(h+k, \boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) = A + \lambda B = \left[\left\{x_{r_u}(k, \boldsymbol{x}_0) - \boldsymbol{x}_{\text{ref}}(h+k)\right\}\right]^2 + \left\{y_{r_u}(k, \boldsymbol{x}_0) - \boldsymbol{y}_{\text{ref}}(h+k)\right\}^2\Big] + \lambda\Big[\left\{a_R(k+1) - a_R(k)\right\}^2 + \left\{a_L(k+1) - a_L(k)\right\}^2\Big], \tag{3.10}$$

where both $\boldsymbol{x}_{\text{ref}}$ and $\boldsymbol{y}_{\text{ref}}$ are the arrays of (x, y) coordinates for the reference trajectory, x_{r_u} and y_{r_u} are the optimized x and y coordinates for the robot and λ is the weighting coefficient. This function penalizes the square of the distance to the reference trajectory and the variation between successive control inputs. These penalization terms are explained in greater detail along this section. The use of quadratic not only terms promotes computational efficiency, since, for these terms, every local minimum is also global, and can be attained at a relatively low computational cost, but also avoids positive and negative values from cancelling out.

Distance to reference

The term that penalizes distance to reference is given as

$$A = \left\{x_{r_u}(k, \boldsymbol{x}_0) - \boldsymbol{x}_{\text{ref}}(h+k)\right\}^2 + \left\{y_{r_u}(k, \boldsymbol{x}_0) - \boldsymbol{y}_{\text{ref}}(h+k)\right\}^2. \tag{3.11}$$

This expression computes the squared Euclidean distance between the calculated robot's coordinates at each point in the prediction horizon $(x_{r_u}(k, \boldsymbol{x}_0), y_{r_u}(k, \boldsymbol{x}_0))$ and the coordinates given for the time considered by the reference trajectory $(\boldsymbol{x}_{\text{ref}}(h+k), \boldsymbol{y}_{\text{ref}}(h+k))$. The use of quadratic terms promotes smoothness and continuity, and penalizes great deviations more severely than the case of a linear function. Smaller changes are penalized less severely than greater deviations, thus, when reacting to a deviation from the trajectory, the robot reacts more abruptly when its farther away, smoothing its trajectory on approach to the reference.

The dependence of the reference coordinates on h, this is, the time step at which the optimal control problem is solved, guarantees that the comparison follows along the trajectory as the robot moves. For the initial time step at $h = 0$, the comparison should be made point-wise for the first N points. Then, for $h = 1$, the reference coordinates used for the calculation over the prediction horizon should correspond to the points from 1 to $N + 1$, hence, the dependence on $h + k$. The robot's coordinates calculated over the prediction horizon are determined using the vehicle's dynamics. They depend on the robot's initial state $\boldsymbol{x}_0$ and on the step in the prediction horizon.

Successive control input differences

The term that penalizes successive control differences is given as

$$B = \Big\{a_R(k+1) - a_R(k)\Big\}^2 + \Big\{a_L(k+1) - a_L(k)\Big\}^2. \tag{3.12}$$

This expression penalizes successive control input differences by penalizing the difference between the magnitude of both the right and left wheels' accelerations at successive time steps over the control horizon ($k+1$ and k, respectively). This term allows for the reduction of oscillatory behaviour, through the reduction of control jerk. Control jerk is defined as a measure of how quickly or abruptly the acceleration of a system changes. Formally, control jerk is defined as the third derivative of position with respect to time.

In the presence of disturbances or noise, the controller is discouraged from performing overly aggressive control actions, which would eventually amplify the perturbation introduced initially. As a result, the robot is able to react to a deviation or perturbation without overcompensating the correction. Overshooting is reduced and the return to the reference trajectory is smoother and more controlled.

Weighting coefficient

The weighting coefficient λ adjusts the relative importance of each term in relation to the overall cost function. The weighting coefficient of the distance to reference term, not shown, is assumed to have a value of one and is, consequently, not represented or further mentioned. Changing the value of λ allows to balance the objectives and control the trade-offs involved in the definition of the problem. Assuming a lower value of λ would guarantee a more accurate reference tracking, at the expense of a greater control effort and greater oscillations. Conversely, a higher value of λ would guarantee a lower control effort, at the expense of a possible greater deviation from the reference trajectory.

3.4.3 Constraints

The constraints define the feasible region of solutions for the problem considered, i.e., the globally optimal solution that minimizes the cost function must satisfy these restrictions. Hence, these are hard constraints. They pertain to the initial state of the robot, the vehicle dynamics, the impossibility of performing reverse maneuvers and, finally, to the obstacle avoidance capabilities of the robot.

Initial state

The initial state constraint is expressed as

$$\boldsymbol{x}_u(0, \boldsymbol{x}_0) = \boldsymbol{x}_0. \tag{3.13}$$

This is derived directly from the formulation of the Nonlinear Model Predictive Control algorithm, as it states that the robot's state at the time step considered is defined as the initial condition in order to,

subsequently, solve the Optimal Control Problem. As will be explained further, in Subsection 3.7.4, the robot's position and orientation is given by the AMCL pose estimate, through a *tf* transform.

Obtaining both the x_r and y_r vehicle coordinates at the initial time step is trivial, but the same does not happen for the heading angle θ_r. This angle is given in the form of quaternions, a mathematical representation of rotations in three-dimensional space that allow for compact and computationally efficient manipulation of orientation. These are represented as a vector (q_0, q_1, q_2) and a scalar q_3 components. The heading angle is calculated from these components through

$$\theta = \text{atan2}\Big(2(q_2 q_3 + q_0 q_1), 1 - 2(q_1^2 + q_2^2)\Big). \tag{3.14}$$

Finally, the vehicle is always assumed to start from a static position, such that $v_R(0) = v_L(0) = 0$. For the following time steps, the initial velocity is given as the optimal velocity found in the previous time step, so $v_R(h) = v_R^{\text{opt}}(h-1)$ and $v_L(h) = v_L^{\text{opt}}(h-1)$. Though it is possible to retrieve the robot's current linear and angular velocities through the odometry readings, the change made for compatibility between the robot's control inputs and the model, explained in Subsection 3.6.2, requires that the initial state's velocities are defined this way. Moreover, adding the odometry callback function just for this purpose would contribute to a decrease in computational efficiency. At time step $h = 0$, the initial state is defined as $\boldsymbol{x}_0 = (x_r, y_r, \theta_r, 0, 0)$ and at every other time step $h \neq 0$ as $\boldsymbol{x}_0 = (x_r, y_r, \theta_r, v_R^{\text{opt}}(h-1), v_L^{\text{opt}}(h-1))$.

Vehicle dynamics

The vehicle dynamics constraint is expressed as

$$\boldsymbol{x}_u(k+1, \boldsymbol{x}_0) = f(\boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)). \tag{3.15}$$

As was mentioned in Chapter 2 the continuous-time dynamics of the differential drive robot are discretized through trapezoidal quadrature. For visual purposes, the notation $x_r(k) := x_{r,k}$ is used for all variables. The discretized dynamics are given as

$$\begin{aligned}
x_{r,k+1} &= x_{r,k} + \frac{1}{2}h(v_k \cos(\theta_{r,k}) + v_{k+1}\cos(\theta_{r,k+1})),\\
y_{r,k+1} &= y_{r,k} + \frac{1}{2}h(v_k \sin(\theta_{r,k}) + v_{k+1}\sin(\theta_{r,k+1})),\\
\theta_{r,k+1} &= \theta_{r,k} + \frac{1}{2}h(\omega_k + \omega_{k+1}),\\
v_{R,k+1} &= v_{R,k} + \frac{1}{2}h(a_{R,k} + a_{R,k+1}),\\
v_{L,k+1} &= v_{L,k} + \frac{1}{2}h(a_{L,k} + a_{L,k+1}).
\end{aligned} \tag{3.16}$$

Positive linear velocity

The positive linear velocity constraint is expressed as

$$v_u(k, \boldsymbol{x}_0) \geq 0. \tag{3.17}$$

This prevents the robot from performing reverse manoeuvres, as a safety measure. The laser sensor range is limited to the front of the robot, hence, it prevents the robot from moving without knowledge of its environment.

Obstacle avoidance

The obstacle avoidance constraints are expressed as

$$\boldsymbol{d}^2_{r\to\text{obs}}(k,j) \geq d^2_{\text{safe}}, \forall j \in [1,2,...,M]. \tag{3.18}$$

These constraints are squared in order to promote computational efficiency, by removing the square root, but have a complex formulation. For that reason, their logic and implementation are fully explained in Section 3.5.

3.5 Obstacle avoidance constraints

Obstacle avoidance in this work consists of two sequential actions: obstacle identification and the obstacle avoidance itself. The first is the definition of points in the reachable environment considered as obstacles. The second is the actual avoiding actions taken to keep within a safety distance of said obstacles. Two approaches were initially experimented when tackling this, one based solely on the laser scan data and the other on occupancy grid maps. These approaches were developed and experimented as an integral part of this work.

Laser scan approach

The laser scan data approach consists on using the data received at each instant from the robot's onboard laser sensor. The data is given in polar coordinates, by means of an angle and a corresponding distance to an obstacle. First, this data is filtered to include only the reachable obstacles, i.e., obstacles that the robot is able to reach within the prediction horizon. These obstacles are at a maximum distance of

$$d_{\max} = Nhv_{\max}, \tag{3.19}$$

where N is the prediction horizon length, h is the time step considered (in seconds) and $v_{\max}$ is the maximum robot's velocity (in meters per second). The cartesian coordinates obtained are set as the obstacle points. Avoidance is achieved by introducing hard constraints such that the robot's position over the prediction horizon is always at least at a distance greater than the safety distance of every obstacle point considered, as seen in 3.18. The solver is then able to calculate the optimal controls such that these constraints are ensured (and the deviation from the reference trajectory is as little as possible).

To reduce the computational burden of this approach, a voxel grid, as explained in Chapter 2, is used. If a voxel grid cell contains any obstacle point detected by the laser, then the midpoint of said cell

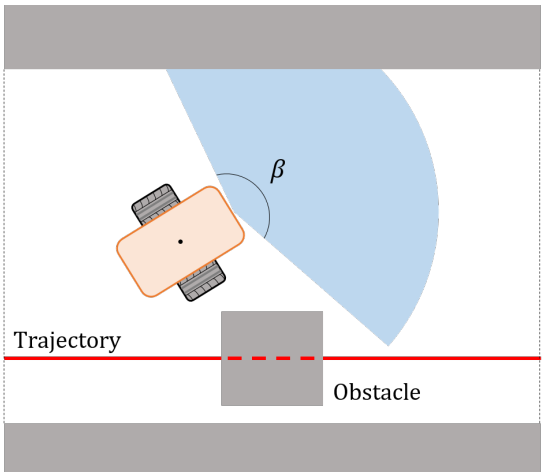


Figure 3.4: Example of robot blindness when overtaking an obstacle with the laser scan approach.

is considered as the obstacle point instead. This allows for a bundling of the original obstacle points, which ultimately results in a reduction of constraints and contributes positively to the computational efficiency of the problem.

The laser scan approach, however, has some limitations. First, the laser suffers from large blind spots when overtaking an obstacle. As seen in Figure 3.4, when the robot deviates from its trajectory to avoid colliding with the static obstacle box in its way, it becomes blind to the obstacle. It has no memory of it being there. This will lead to the robot prematurely trying to rejoin its trajectory and eventually colliding with the obstacle. Further, when using this laser scan approach it is not possible to limit the robot's movement to a predefined navigable area. When working in a factory environment, for safety purposes, it is imperative that the robot keeps within the defined area assigned to its navigation. This is, that there is a possibility to virtually define an area in which the robot is permitted to be in.

Occupancy map approach

The other approach relies on using occupancy grid maps. Occupancy grid maps divide the environment into cells, and identify each cell with a number that indicates the likelihood that the cell is occupied. These numbers range from 0 (unoccupied cell) to 100 (total likelihood that the cell is occupied). These maps overlay the data between a pre-defined static map and the laser scan to map the occupancy of the robot's environment. This mapping of the environment is final, this is, when the laser scan records the presence of an obstacle, the occupancy map will include this obstacle in that position until proven otherwise. Unlike the laser scan approach, this approach has memory capabilities. Further, both the laser scan rate and the update frequency of the map are greater than the controller's operating frequency.

To use the occupancy map as a constraint, it is possible to restrict the robot's movement to points with an occupancy value less than a maximum occupancy $occ_{\max}$. To do this, at each iteration of the numerical solver, the occupancy of the optimal robot's position would be checked by means of an interpolating function. If any (x, y) point over the prediction horizon is found to have an occupancy value greater than $occ_{\max}$, then this solution is considered infeasible and discarded, prompting a new solver

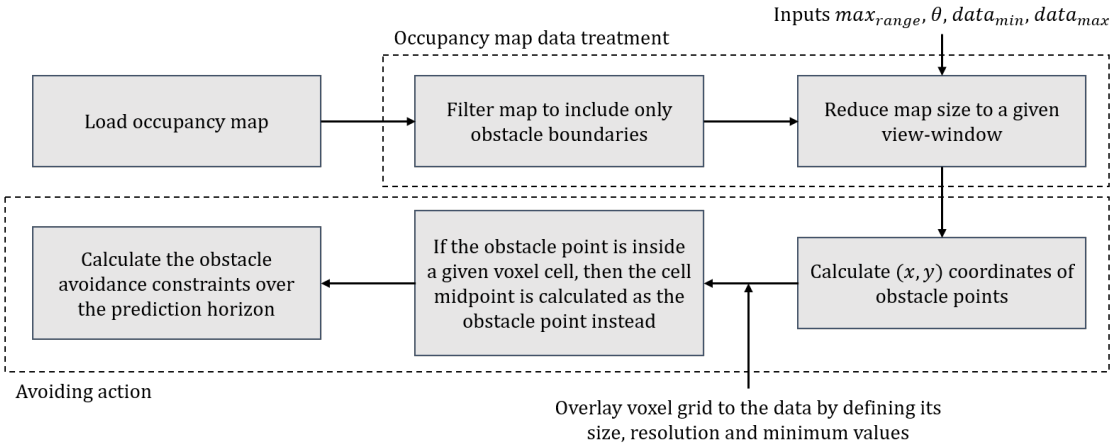


Figure 3.5: Methodology used to define the obstacle avoidance constraints at each time step.

iteration. However, the use of this interpolating function in order to retrieve the values of occupancy in a continuous manner, does not allow for the use of CasADi's *expand*[1] functionality, which is essential to achieve the computational efficiency needed for real-time implementation.

Adopted approach

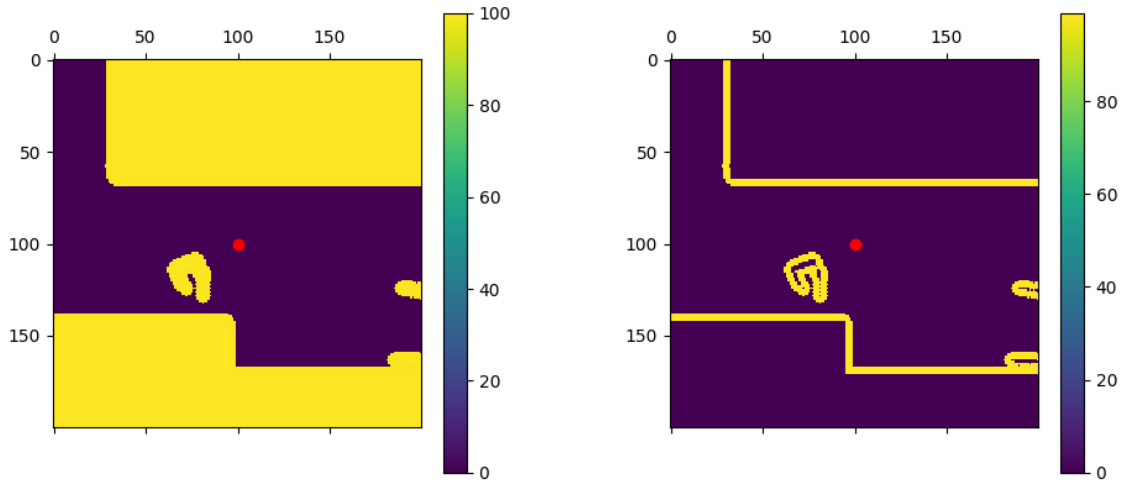
The adopted solution to the obstacle avoidance problem combines both previous approaches in order to include the memory capabilities of the occupancy map approach, to avoid blindspots, and the computational efficiency of the laser data approach. Moreover, it is important that the final adopted solution also considers the artificially defined navigable areas. This is, that there is a way to restrict the robot's movement inside an area with no physical but virtual barriers, which is only possible for the occupancy map approach. The implemented strategy is shown in Figure 3.5. At every controller cycle, this methodology takes the occupancy map as an input and outputs the definition of the obstacle avoidance constraints over the prediction horizon.

3.5.1 Occupancy map data treatment

Occupancy map

First, the data from the occupancy map is loaded, containing information on both the pre-defined navigable area and the laser scan data, treated as occupied cells. As was explained previously, this map describes the environment taking into account probability of occupation for each cell. For computational efficiency purposes, it is important that the data from this map is interpreted solely considering the obstacle barriers, and not the inside of the obstacle itself. As such, the data is treated to include the information pertaining to the inflation layer of the obstacles, as seen in Figure 3.6. This inflation layer is created as a safety measure and consists on an expansion of the obstacles' limits by a predefined amount. Both the definition of the inflation layer and the predefined expansion amount are a function of

[1]The *expand* function is part of CasADi's library and transforms all matrix symbolic values calculations to simple symbolic values calculation, thus greatly reducing the computational burden of the numerical solver. An example of this use can be seen in `https://casadi.sourceforge.net/v2.3.0/api/html/Function/expand.pdf` (accessed June 29th, 2023).



(a) Untreated occupancy map of the robot's environment.

(b) Treated occupancy map of the robot's environment.

Figure 3.6: Difference in treatment of the robot's (red dot) occupancy map to consider solely obstacle barriers.

the occupancy map and, as such, are outside the scope of this work.[2] It should be noted that considering the space inside the obstacles as free poses no problem to safe operation of the robot, since it will never be able to cross the limits of the obstacles, thus, contributing simply to the computational efficiency of this problem.

Without loss of generality, the default full occupancy map used for this work is given as a 10-by-10 meter square centered on the robot, or, considering the used map resolution of 0.05 meters per occupancy cell, it is a square of 200-by-200 cells. This occupancy map is given as an input from the project. This map was proven to be too large, as the computational requirements needed for the treatment of a map this size were not fit for the online implementation nature of an NMPC controller. Thus, the map must be reduced to consider up to a maximum range. This maximum range is a tunable parameter of the developed controller.

View-window

A greater value for the maximum range that the robot is able to perceive increases the responsiveness capabilities of the vehicle. The robot is able to detect the obstacle further away and react to it in a more timely manner. However, considering a larger map means a much greater number of obstacle points, set as constraints for every step in the prediction horizon N and, as such, it increases the computational expense of the problem exponentially. Ideally, a reduced version of the map should be considered, a view-window, that would grant it enough time to react to the obstacles in the environment whilst being as small as possible, in order for its weight in the computational cost of the problem to be negligible. To do this, a square centered on the robot is not the ideal shape to consider.

[2]The definition of both the occupancy map and inflation layers are internal properties of ROS. A further explanation can be found on: `https://wiki.ros.org/nav_msgs` (accessed October 18th, 2023).

When considering this square, the computational weight of the half-square behind the robot is just too large for the little result it produces in terms of obstacle avoidance capabilities. The robot "should see" as much as possible towards its heading direction, in order to react to the obstacles in a more timely manner, and only consider enough space behind it to make use of the memory capabilities of this method. The vehicle just needs enough view behind it in order to avoid blindspots, whilst using the most out of the computational burden of considering the map to consider the obstacles ahead of it.

Considering what has just been stated, the robot's heading angle must be taken into account when calculating the current view-window of the robot. For example, if the heading angle of the robot is in the interval $\theta \in [\frac{\pi}{8}, \frac{3\pi}{8}[$ radians, the robot is considered to be moving primarily in the positive x-axis and positive y-axis directions, and, as such, its view-window should be extended in these directions. Having the maximum range of the view-window defined by the variable max_{range}, the extension of the view-window is defined by the variables $data_{\min}$ and $data_{\max}$, as seen in Figure 3.7.

A greater value for the max_{range} variable will increase the responsiveness capabilities of the robot when detecting an obstacle, since it detects said obstacle further away. However, it also increases the computational effort of the problem, since it accounts for a larger map. The divisions of the heading angle consider intervals of $\frac{\pi}{4}$ radians, starting at an angle of $-\frac{\pi}{8}$ radians. An example of the view-windows calculated for every heading angle interval can be seen in Appendix A. In Figure 3.8, an example can be seen of the treatment of the view-window data having considered the following values:

$$\begin{cases} \theta = \frac{5\pi}{4}\text{rad} \\ max_{\text{range}} = 4\text{m} \\ data_{\min} = 0.2 \\ data_{\max} = 0.8 \end{cases} \tag{3.20}$$

Note also that the calculation of these view-windows are independent of the original occupancy map dimensions and are used in this work for these dimensions without loss of generality of the algorithm.

3.5.2 Avoiding action

In order to fulfill the avoiding action from the obstacles in the environment, first it is necessary to describe the environment into obstacle points. As such, from the occupancy map's origin coordinates and resolution, it is possible to calculate for the (x, y) coordinates of these points in the environment. The obstacle points correspond to the cells marked as occupied in the view-window of the treated occupancy map data. For this work, a cell is defined as occupied if its probability of occupation is greater or equal to 99.

Then, a voxel grid is overlaid on top of the now known (x, y) obstacle points. This grid is defined by its voxel cell size, in meters. For an increase in computational efficiency, this overlaying of the data considers solely the inner data in which there are occupied cells. This is, any periphery of the view-window in which there are no obstacles is not considered for the voxel grid calculation. Finally, in order to bundle the obstacle points that are close together, if a voxel cell has at least one obstacle point inside

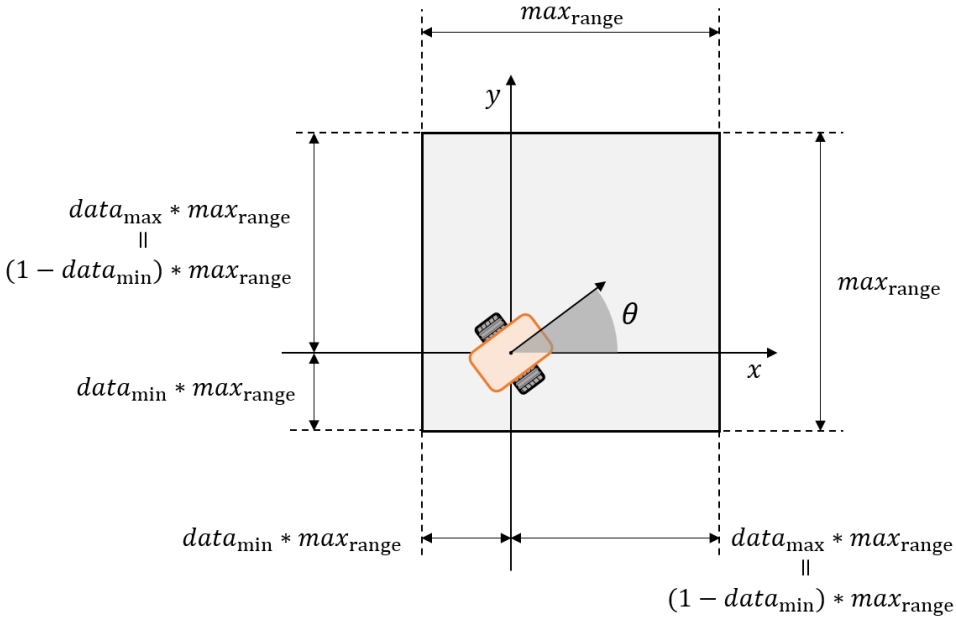
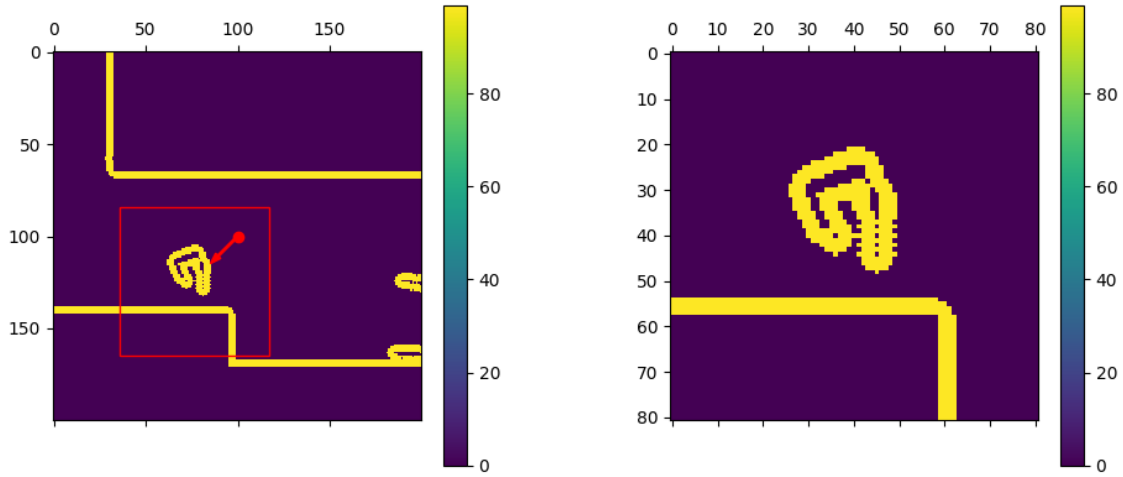


Figure 3.7: Representation of the robot's occupancy map view-window calculation for an heading angle interval of $\theta \in [\frac{\pi}{8}, \frac{3\pi}{8}[\text{rad}$.



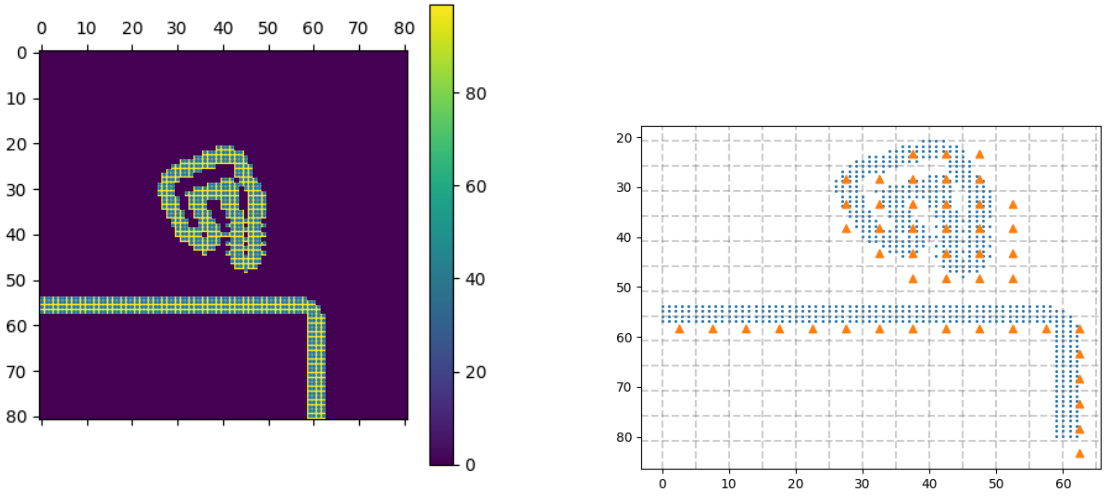
(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

Figure 3.8: Robot's view-window treatment of the occupancy map data for a heading angle of $\theta = \frac{5\pi}{4}\text{rad}$.

it, then the midpoint of that cell is instead used and considered as an obstacle point. This allows for a great reduction of points considered as obstacles and, thus, greatly contributes to the improvement of the computational efficiency of the solver through the reduction of the number of constraints. In Figure 3.9, an example is shown for the treatment of the view-window data of Figure 3.8, that transforms 700 obstacle points into 47 voxel midpoints by overlaying a grid with a cell size of 0.25 meters.

Finally, in order to provide obstacle avoidance capabilities to the controller, hard constraints are set, in which, for every point over the prediction horizon, the robot must keep a safety distance d_{safe} from



(a) Treatment of view-window into 700 obstacle points.

(b) Treatment of 700 obstacle points into 47 voxel midpoints for a voxel cell size of 0.25 meters.

Figure 3.9: Treatment of view-window into voxel midpoints for a heading angle of $\theta = \frac{5\pi}{4}\text{rad}$.

every calculated voxel cell midpoint. This means that any solution that considers the robot entering a space within d_{safe} of any obstacle in the environment is infeasible. The robot will move away from the obstacles, whilst trying to maintain its trajectory as close as possible to the reference. Formally, this set of obstacle avoidance constraints is expressed as:

$$\boldsymbol{d}^2_{r\to\text{obs}}(k,j) \geq d^2_{\text{safe}}, \forall j \in [1,2,...,M]. \tag{3.21}$$

The fact that the obstacles in the environment are described as sets of points allows for the developed controller to not need *a priori* knowledge of their position, dimension, shape or trajectory. There is no need for an extra input that describes the obstacle as a certain shape or mathematical set. Moreover, the fact that the calculations for the voxel cell midpoints are made at every cycle of the controller allows to account for the movement of the obstacles in the environment. Reactiveness of the controller comes from its operating frequency.

3.6 Experimental constraints

When transitioning from a theoretical to experimental setup, some considerations must be taken into account regarding the design of the controller. In this section, changes made because of minimum values of actuation and accepted control inputs are explained.

3.6.1 Minimum actuation

Regarding the physical limitations of the actuators in the robot, both the linear and angular velocity given as the robot's control inputs have minimum values, under which the actuators do not react and the

robot does not move.

To test these minimum input values, a simple practical experiment was conducted. First, a constant angular velocity command is given to the robot. After a minute, the robot's pose was checked to see if any movement had been registered since the controls had been given. A further check was made in person to confirm that the robot had or not moved. These tests were conducted for varying values of angular velocity and were categorized on a yes or no (*Y/N*) frame, dictating whether or not movement was registered. The same was done for varying constant values of linear velocity. The results are shown in Table 3.1. Note that the actuators are equal and behave equally regardless of the direction of motion, so the results are valid for both motors and for negative or positive values of angular velocity. This is also true for the linear velocity case, but there is already a previous non-negative constraint in place. From the experiment, the minimum values of actuation are $|\omega|_{\min} = 0.05\,\mathrm{rad\,s^{-1}}$ and $v_{\min} = 0.02\,\mathrm{m\,s^{-1}}$.

Additionally, the maximum angular velocity at which operation in a real factory scenario is considered safe is $|\omega|_{\max} = 0.9\,\mathrm{rad\,s^{-1}}$. This maximum value is not verified by the velocity bounds in 3.9, thus must also be introduced as a constraint.

The updated optimal control problem is, then

$$\begin{aligned}
\text{minimize} \quad & J_N(h, \boldsymbol{x}_0, \boldsymbol{u}(.)) := \sum_{k=0}^{N-1} l(h+k, \boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \\
\text{with respect to} \quad & \boldsymbol{u}(.) \in \mathbb{U}^N(\boldsymbol{x}_0), \quad \boldsymbol{x}_u(.) \in \mathbb{X}^N(\boldsymbol{x}_0) \\
\text{subject to} \quad & \boldsymbol{x}_u(0, \boldsymbol{x}_0) = \boldsymbol{x}_0 \\
& \boldsymbol{x}_u(k+1, \boldsymbol{x}_0) = f(\boldsymbol{x}_u(k, \boldsymbol{x}_0), \boldsymbol{u}(k)) \\
& v_u(k, \boldsymbol{x}_0) \geq 0.02 \\
& 0.81 \geq \omega_u^2(k, \boldsymbol{x}_0) \geq 0.0025 \\
& \boldsymbol{d}_{r \to obs}^2(k, j) \geq d_{\text{safe}}^2, \forall j \in [1, 2, ..., M].
\end{aligned} \tag{3.22}$$

Note that the angular velocity constraint is given as a quadratic function, rather than an absolute value, for computational efficiency purposes. A lower bound was imposed on the angular velocity to account for the inability of the motors to react to commands below the value of $|\omega|_{\min}$, not compromising the overall convexity of the problem as it is already non-convex due to the nature of the obstacle avoidance constraints and the nonlinearity of the vehicle dynamics. Nonetheless, this could have been discarded

Table 3.1: Experiment for minimum angular and linear velocity inputs that produce movement.

Angular velocity ω $[\mathrm{rad\,s^{-1}}]$	**Movement registered?** [Y/N]	**Linear velocity** v $[\mathrm{m\,s^{-1}}]$	**Movement registered?** [Y/N]
0.03	N	0.01	N
0.04	N	0.015	N
0.045	N	0.02	Y
0.05	Y	0.025	Y
0.055	Y	0.03	Y

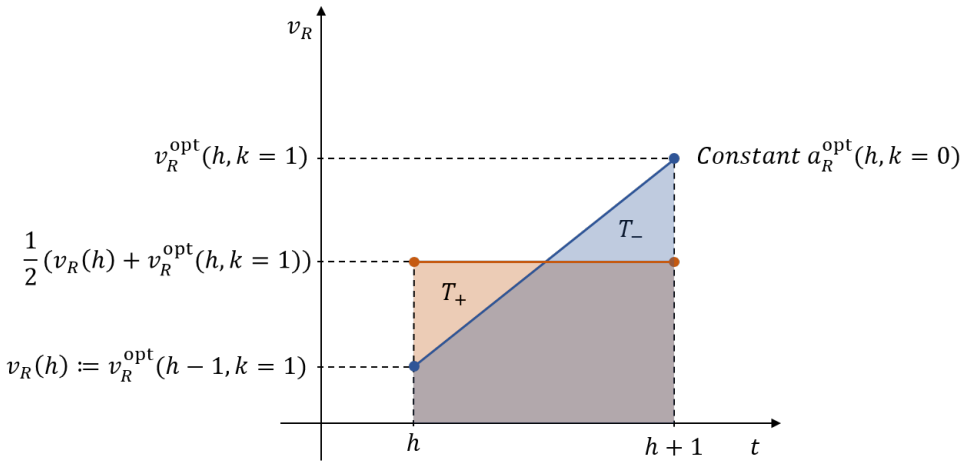


Figure 3.10: Equivalence between constant acceleration and constant velocity for robot's right wheel.

and replaced by a deadband to improve the efficiency of the solver and avoid splitting the feasible set, thus preventing possible cases of early stoppage or falling in local minima by local solvers. The use of this constraint accentuates the need for a good initial guess, as will be explained in Subsection 3.7.3.

3.6.2 Control inputs

Another imposed design change is the adaptation of the accepted control inputs. While the vehicle dynamics model considers a constant acceleration representation, the robot takes constant linear and angular velocities as its control inputs. As such, a transformation must occur before these commands can be passed to the robot.

Instead of feeding the robot the right and left wheel's optimal accelerations at $k = 0$, calculated at time step h, an average is calculated between the robot's current wheel velocities and the optimal wheel velocities, calculated at h, for $k = 1$. The latter averaged wheels' velocities are then fed to the robot as its input. In Figure 3.10, a graphical representation of this logic is presented, to show the equivalence between the two methods.

In fact, because the model used assumes a constant acceleration, the function that represents velocity as a function of time will always be linear. This means that the area under the curve for both the constant acceleration at $k = 0$, represented in blue, and constant velocity at $k = 1$, represented in orange, is effectively the same. Essentially, this means that the change in position, what is effectively tracked against the reference, is the same for the two methods. They are equivalent in terms of their output. From this formulation and 3.3 and 3.4, the linear and angular velocities fed to the robot as control inputs are calculated as:

$$\begin{aligned} v^{\text{opt}}(h) &= \frac{1}{2}\left\{\frac{v_R(h) + v_R^{\text{opt}}(h, k=1)}{2} + \frac{v_L(h) + v_L^{\text{opt}}(h, k=1)}{2}\right\}, \\ \omega^{\text{opt}}(h) &= \frac{1}{L}\left\{\frac{v_R(h) + v_R^{\text{opt}}(h, k=1)}{2} - \frac{v_L(h) + v_L^{\text{opt}}(h, k=1)}{2}\right\}. \end{aligned} \tag{3.23}$$

3.7 Controller design

When developing the controller, a few more design choices are made, namely regarding the controller's solver, operating frequency, warm-start and positioning. Additionally, a strategy is devised in order to deal with the NMPC's limitations regarding the final part of the trajectory. These choices and strategy are explained throughout this section.

3.7.1 Solver type

CasADi offers a choice of multiple nonlinear optimization solvers. For the use case considered, an Interior Point Optimizer (IPOPT) was chosen. This algorithm is based on the concept of finding locally optimal solutions within the interior of a feasible region, rather than on the boundary of said region, hence, the interior point designation. It may also be called as a barrier method, as it resorts to barrier functions, which are continuous value functions that approach infinity as they get closer to the edge of the feasible region. The fact that this method can only calculate for locally optimal solutions means that the accuracy of the initial guess provided is even more important [62].

This choice of solver was considered for its efficiency and flexibility. A comparative study made by Silva [1], shows that, for this scenario, IPOPT is the most appropriate choice. It is not only suited to dealing with large-scale nonlinear optimization problems, but also supports both equality and inequality constraints and can handle both convex and non-convex problems.

3.7.2 Control rate

When choosing the control rate, this is, the frequency at which every NMPC cycle is run, a trade-off is considered between the needed reactivity of the robot and the constraints associated with the limits of the actuators. A higher frequency allows for a greater responsiveness to obstacles in the environment or deviations from the reference, but it also may not be suited to dealing with the actuators physical constraints. Note that, in this work, control rate and controller's operating frequency are used without distinction and in reference to the same parameter.

For this reason, the controller was chosen to operate at a frequency of $f_c = 5\,\mathrm{Hz}$. This is the maximum frequency at which the actuators can react and operate in a safe manner. The time step will then be $h = f_c^{-1} = 0.2\,\mathrm{s}$. The cycle time, defined as the time for the robot to get its position, solve the optimal control problem and feed the optimal control inputs to the robot, should stay below this value. Essentially, due to the online optimization nature of an NMPC, computational efficiency has a direct influence on controller performance.

3.7.3 Initial guess

A solver is particularly sensitive to its first iteration. The closer the initial guess is to the actual optimal solution, the more efficient the solver will be, which will directly influence the controller's performance. Note that h is the time step of the discretized interpolated trajectory and k is the step in the optimization

problem. Considering that both the linear and angular velocities are calculated from the right and left wheels' velocities, the optimization vector is defined at each point over the prediction horizon as

$$\boldsymbol{X}(k,h) = [x_r(k+h), y_r(k+h), \theta_r(k+h), v_R(k+h), v_L(k+h), a_R(k+h), a_L(k+h)]. \tag{3.24}$$

At time step $h = 0$, the initial guess is defined directly from the given reference trajectory as

$$\boldsymbol{X}(k,0) = [\boldsymbol{x}_{\mathrm{ref}}(k), \boldsymbol{y}_{\mathrm{ref}}(k), \boldsymbol{\theta}_{\mathrm{ref}}(k), \boldsymbol{v}_{R_{\mathrm{ref}}}(k), \boldsymbol{v}_{L_{\mathrm{ref}}}(k), \boldsymbol{a}_{R_{\mathrm{ref}}}(k), \boldsymbol{a}_{L_{\mathrm{ref}}}(k)]. \tag{3.25}$$

At every other time step, the optimal states from the previous solver iteration are used. For a given time step h, $k-1$ optimal states from the solution at time step $h-1$ are used. Using the same compact notation as in 3.16, the optimal states used must advance in accordance with the time step increase, resulting in the provided initial guess of

$$\boldsymbol{X}(k,h) = \begin{cases} [\boldsymbol{x}_r^{opt}, \boldsymbol{y}_r^{opt}, \boldsymbol{\theta}_r^{opt}, \boldsymbol{v}_{R_r}^{opt}, \boldsymbol{v}_{L_r}^{opt}, \boldsymbol{a}_{R_r}^{opt}, \boldsymbol{a}_{L_r}^{opt}](k+1, h-1), & k < N-1 \\ [x_{\mathrm{ref},k+h}, y_{\mathrm{ref},k+h}, \theta_{\mathrm{ref},k+h}, v_{R_{\mathrm{ref},k+h}}, v_{L_{\mathrm{ref},k+h}}, a_{R_{\mathrm{ref},k+h}}, a_{L_{\mathrm{ref},k+h}}], & k = N-1 \end{cases} \tag{3.26}$$

Considering that the vehicle dynamics model is accurate in its representation of the robot's real behaviour, in situations where reactive obstacle avoidance is not required, the solver should only iterate over the last state in the prediction horizon - the state that is defined by the suboptimal values given by the reference trajectory. This greatly contributes for computational efficiency and would allow for a increase in prediction horizon length without an accompanying increase in solver time.

3.7.4 Robot localization

Robot localization is the process of determining the robot's position and orientation in the environment. This can be done based on sensor readings, either from the wheel encoders or from other measurements of the robot's movement relative to its starting point. This is called odometry. Though fast and reactive, odometry positioning can accumulate errors over time or for long distances due to uneven terrain, wheel slippage, sensor noise and other factors.

As a way to correct the inaccuracies from the sole use of the odometry, a probabilistic approach to positioning was proposed by Fox [63], called the Adaptive Monte Carlo Localization (AMCL) approach. It takes in information from a laser-based map, laser scans, transform messages and, using a particle filter, it tracks the position of the robot against a known map. This particular approach to pose estimation can correct for drift and other errors that occur in the odometry positioning, thus, making it more suited to deal with long trajectories and unknown or cluttered environments. A scheme of the AMCL algorithm localization, as implemented by its ROS package, can be seen in Figure 3.11.

For the work developed in this thesis, the robot's position and orientation were tracked using the AMCL pose estimate through a *tf* transform. The *tf* system is a tree-based data structure to represent the relations between different coordinate frames in ROS. It can be accessed and altered using the *tf*

and *tf2* ROS packages. Accessing the AMCL pose estimation in this manner allows for a bypass of the update frequency constraints between coordinate frames. The maximum update frequency between the odometry and map frames is around 5Hz, which is insufficient for the purpose of this work. Using the AMCL ROS package would mean that the message used to get the vehicle's position and orientation could be unsynchronized from the actual position and orientation by up to 0.2 seconds. The use of the *tf* transform for positioning allows for near instantaneous updates and is needed to guarantee precision.

3.7.5 Last set of control inputs

The formulation of an NMPC controller does not allow for the last $N-1$ control inputs to be given in a closed-feedback format, as is the case for all the previous commands. Doing so, would require the Optimal Control Problem to be solved beyond the last point in the reference trajectory. As an example, consider a prediction horizon of $N=3$ and a trajectory with t_s time steps. At time step $h=t_s-1$, the cost function would have to be calculated for the set of points at time steps $[t_s-1, t_s, t_s+1]$, which is impossible, as it goes beyond the trajectory's duration.

As a consequence, the last $N-1$ control inputs are fed to the robot in an open-loop manner. For a trajectory with total number of time steps t_s, at time step $h=t_s-N$, instead of retrieving the optimal controls at $k=0$, the optimal controls over the prediction horizon are kept. Then, these controls are fed to the robot at each corresponding time step. This is, for time step $h=t_s-N+1$, the control input will be the one calculated for $k=1$ at time step t_s-N. No comparison is made to the robot's position estimate and no optimal control problem is solved under these new conditions, hence, the open-loop procedure. At the last point in the trajectory, commands are given to the robot to stop.

3.8 Trajectory replanning

During the robot's normal operation in a real-world environment, it is expected that its position might trail behind the reference trajectory. This is, it is expected that there is a delay between the robot's actual position at time t and its reference state at said time. This delay happens due to unmodelled robot dynamics, oscillations, latency, among others and, if unaccounted for, severely compromises the accuracy in the trajectory tracking. It is also expected that the robot trails behind its reference state when performing obstacle avoidance, particularly when the obstacles are large enough so that they may

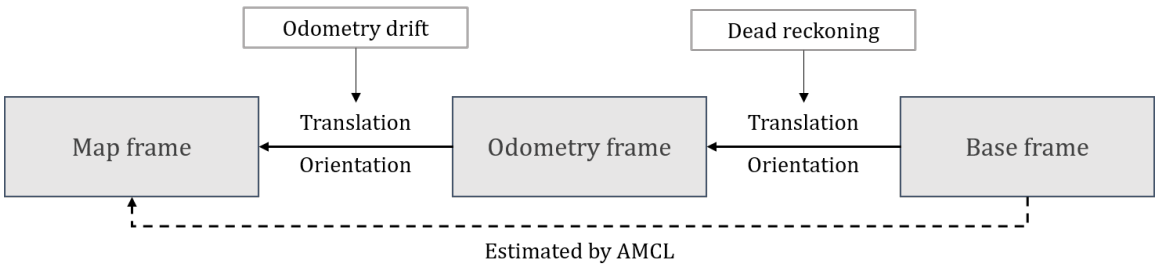


Figure 3.11: Relations between coordinate frames for AMCL map localization.

prompt accentuated evasive behaviour.

As was explained previously, in Section 3.3, in order to tackle this delay, a distinction is made between nominal and maximum velocities of the robot. The nominal velocity v_{nom} is the one that the trajectory planner considers as its maximum allowable velocity, when defining the trajectory from the robot's current position to its goal state. The maximum velocity v_{max}, on the other hand, is the velocity that the trajectory tracking algorithm considers as the maximum allowable velocity when tracking the given trajectory. The maximum velocity must have a value greater than nominal, in order to allow for the robot to catch-up to the reference, when it trails behind it.

It is now important to recall that one of the main drivers of this work is the need for a system that tracks trajectories rather than paths. Following a reference trajectory allows for an added safety dimension when planning for multi-robot systems, as is the case for the experimental application of this work. In essence, the planner knows when a certain robot will be at a certain state and can plan the other robot's trajectories around it. If the robots were tracking a path, in order to guarantee that two robots do not collide, the planner could only plan the second robot's path after the first robot had reached its goal. This wait would only need to happen for paths with one or more coincident states. Notwithstanding, it would greatly impact the efficiency of the whole operation.

Given the trajectory tracking requirements, it is important that the controller also considers a maximum deviation from the reference trajectory d_{replan} above which a trajectory replanning is triggered. The reasoning behind this replanning trigger is that a delay large enough such that the robot is unable to catch up to the reference trajectory in an acceptable time span would ultimately defeat the purpose of trajectory, instead of path, tracking. As such, at each time step h, an Euclidean distance is calculated between the robot's current position (x, y) and its reference position at the next time step $(x_{\mathrm{ref}}(h+1), y_{\mathrm{ref}}(h+1))$:

$$d = \sqrt{[x - x_{\mathrm{ref}}(h+1)]^2 + [y - y_{\mathrm{ref}}(h+1)]^2}. \tag{3.27}$$

If this distance d is greater than d_{replan}, then the robot is stopped, the trajectory tracking is interrupted and a new trajectory is calculated from the robot's current position to its goal, recalling the trajectory generation algorithm in use. The goal is the same as it was for the initial trajectory, and will remain as such if any further needs for replanning arise, until said goal is reached.

Chapter 4

Trajectory tracking results

In this chapter, the trajectory tracking capabilities of the controller are studied and compared against the TEB baseline, currently implemented as a solution in the project and previously explained in Subsection 1.3.3. First, the environment used for testing shown, as well as the methodology explained. Then, a study on the validity of the used vehicle model is done. For controller tuning, a study is done regarding the influence of varying parameters such as the prediction horizon length and the weighting coefficient of the cost function and how this affects the computational efficiency of the controller. The main experimental results presented pertaining to the trajectory tracking capabilities of the developed controller are shown and compared against the baseline TEB solution. Finally, studies are conducted on both the influence of the maximum velocity given to the robot and how convergence is attained when away from the reference.

4.1 Methodology

The results of this work were obtained using a differential drive robot operating on Imeguisa's factory floor. This environment, as well as the robot used, are shown in Figure 4.1. The robot's dimensions, length and width, are as follows: $c_r = 1.13\mathrm{m}$ and $w_r = 0.695\mathrm{m}$. Though testing in simulation was a great aide in developing the algorithm in this work, those results are not presented, as the end goal was always real-world adaptation and implementation.

For each experiment, there are certain variables whose value can be changed in order to either tune the controller or adapt to the circumstances considered. These variables have been explained throughout this document and are summarized in Table 4.1. These variables pertain to either the actual function of the controller, the robot, the obstacle avoidance and replanning capabilities of the algorithm.

Finally, the differential drive robot with which the algorithm was experimented has an onboard computer with an *Intel Core i5-1145G7E*, a 2.60 GHz processor and 8.00 GB of RAM. The operating system used is *Ubuntu 20.04.1*. The laser mounted onboard, inside the robot and, as such, not visible, is a *SICK nanoScan3* safety laser scanner.

(a) Imeguisa's factory floor and corresponding navigable area.

(b) Differential drive robot used for experiments.

Figure 4.1: Environment and robot used for real-world testing of the developed algorithm.

Table 4.1: Experiment's identifying criteria and variables.

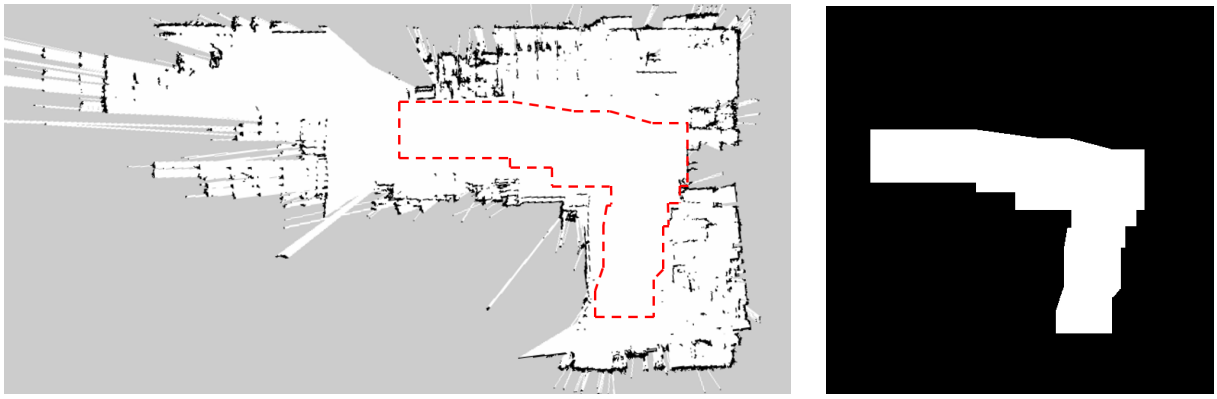
Type	Variable	Description	Unit
Controller	f_c	Controller's operating frequency	Hz
	N	Prediction horizon	NA
	λ	Cost function's weighting coefficient	NA
Vehicle	L	Vehicle's wheel-base	m
	$v_{\max}$	Vehicle's maximum velocity	$\mathrm{m\,s^{-1}}$
	$a_{\max}$	Vehicle's maximum acceleration	$\mathrm{m\,s^{-2}}$
Obstacle avoidance	d_{safe}	Safety distance from obstacles	m
	$voxel_{\mathrm{size}}$	Voxel cells' dimensions	m
	max_{range}	Maximum range for occupancy map	m
	$data_{\min}$	Minimum value for view-window of occupancy map	NA
	$data_{\max}$	Maximum value for view-window of occupancy map	NA
Replanning	d_{replan}	Distance from trajectory that triggers replanning	m

4.1.1 Mapping of the environment

Regarding the mapping of the environment, as mentioned in Subsection 3.7.4, the AMCL positioning estimate used is a probabilistic approach based on overlaying data from sensors with a previously given map of the environment. This method provides for some tuning of its parameters. For replicability of the experiments in this work, the values used for the AMCL positioning are given in Appendix B. These parameters are not explained in detail, as this goes beyond the scope of this work.[1]

The environment must be mapped to represent it as the robot will encounter it. This map should also coincide with the map used to generate the trajectory, done so by Silva's algorithm [1]. The truer the depiction of the mapped environment to the actual real scenario, the more accurate the overlaying of sensor data will be, which ultimately leads to a better robot positioning estimate. The factory map is

[1] For a detailed explanation on each of these internal parameters, please refer to the official website for the AMCL localization system: `http://wiki.ros.org/amcl` (accessed September 3rd, 2023).



(a) Environment map on *RViz* with outline of navigable area.

(b) Navigable area.

Figure 4.2: Mapping of the factory environment used for testing.

presented as seen on *RViz*[2] in Figure 4.2, along with the defined navigable area. The overlaying of the navigable area with the map defines the borders of the environment in which the robot is free to move. Note that the navigable area is defined such that the occupancy map cells in which the robot is not able to move into are marked as occupied.

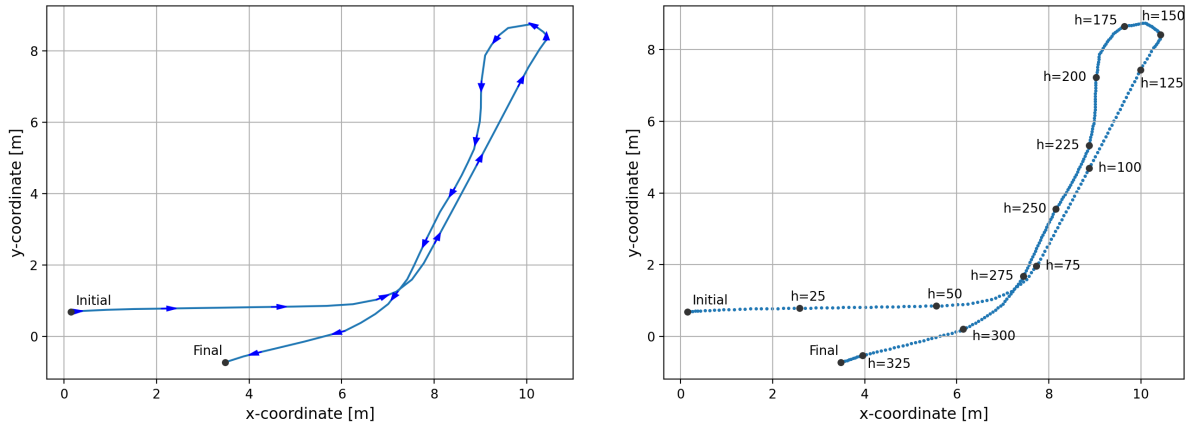
From experiences conducted in the factory environment, as a part of this work, it was proven that the use of an outdated map of the environment, that failed to account for obstacles along the borders of the navigable area, leads to catastrophic oscillatory behaviour. The error introduced in pose estimation leads to over-correction of the controller, due to non-continuous change in estimated position. The AMCL algorithm tries to adapt the pose to the sensing of a foreign environment and changes its estimate abruptly and discontinuously. The robot "is lost" and, therefore, cannot perform smooth tracking of the given reference.

Finally, it is important to consider that the localization mechanism of the robot is experimental, as this is a research project. Because of this, certain shortcomings must be acknowledged *a priori*, that ultimately introduce some error in the accuracy trajectory tracking and the precision of the obstacle avoidance of the controller. One case in which this is particularly visible is when the robot is rotating, and its motor-based odometry system struggles to precisely follow the changes in laser data. As such, the position of the obstacles might shift momentarily in the environment and appear to be at an erroneous location. This makes the obstacle avoidance capabilities of the algorithm particularly harder, especially when translating to these real-world experiments, in addition to introducing some oscillation when the robot is performing movements that require angular velocity.

4.1.2 Data collection and treatment

The reference trajectory used for all the tests featured in this section is shown in Figure 4.3. This trajectory presents no obstacles in the robot's path, so no avoiding action is needed during tracking, and it was chosen since it features all possible components of movement that the robot might be subject to: straight lines, a left-hand turn, a right-hand turn and a turnaround. Note that this trajectory features

[2] *RViz* is the graphical visualization tool for ROS. It allows for the visualization of the robot model, logging of sensor information and replays of this information.



(a) Orientation from start to goal.

(b) Time step distribution for reference trajectory.

Figure 4.3: Reference trajectory used for real-world tracking experiments.

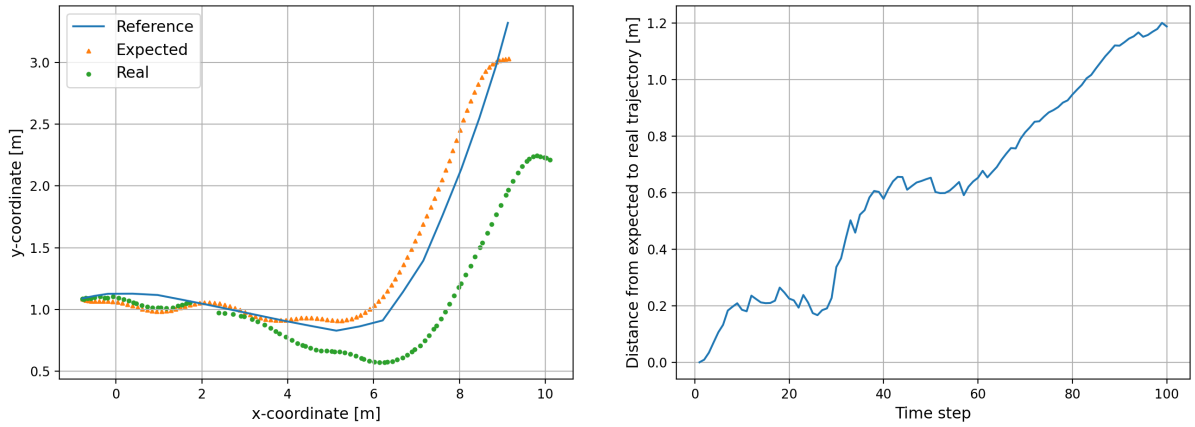
more difficult maneuvers than the robot will encounter in an industrial scenario, for purposes of rigorous testing of the developed controller. The reference trajectory has a total length of 28.24 meters. From the time step distribution it becomes apparent that the initial part of the trajectory has a higher speed when compared to the latter part of it, with the turnaround maneuver occurring at lower speed conditions.

For the treatment of the trajectory tracking data some factors must be considered. First, when testing the controller's performance, it is important to measure distance to reference at each time step. This will be considered as a Euclidean distance in the (x, y) plane and is a direct measure of trajectory tracking performance. Note that for testing TEB as a baseline solution, which follows a path, it is impossible to calculate this distance to reference at each time step, since it does not track a time-parameterized reference. Additionally, the distance to the goal configuration should be assessed. The robot should be able to reach an area very close to its goal, in order to fulfill with its docking purposes, as was mentioned in Chapter 1. Finally, due to the online implementation nature of the NMPC framework, the computational efficiency of the controller must also be studied.

4.2 Validation of the vehicle model

In order to validate the vehicle model, put simply, to confirm that there are no major deviations due to unmodelled dynamics, an open-loop set of controls was given to robot and its trajectory compared to the reference. This set of controls was calculated through an optimal control problem (OCP) with a prediction horizon that matches the trajectory's duration. Solely for this section in this chapter, the given reference trajectory is not as shown in Figure 4.3, since it is too complex for the study considered.

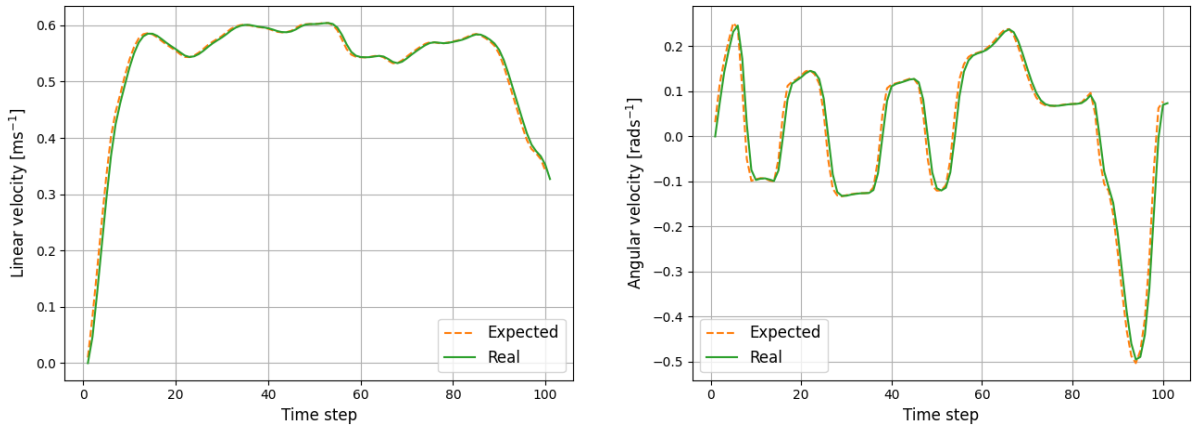
In Figure 4.4, it is possible to see the comparison between calculated reference trajectory, the expected trajectory and the real trajectory, calculated through odometry readings. The expected trajectory are the optimal (x, y) robot positions calculated by the solver over the length of the prediction horizon (in this case, the trajectory's duration), while the real trajectory is the one that is actually done by the robot. From analysis, it is clear that there is a mismatch between these two trajectories, though conserving a



(a) Comparison between reference, expected and real trajectories.

(b) Euclidean distance between expected and real trajectories at each time step.

Figure 4.4: Trajectory study of the vehicle model.



(a) Comparison for linear velocity.

(b) Comparison for angular velocity.

Figure 4.5: Comparison between the expected and actual linear and angular velocities.

similar shape throughout. Moreover, the drift between the expected and real robot trajectories, calculated as an Euclidean distance at each time step between the two, increases over the duration of the trajectory.

In Figure 4.5, a comparison is shown between the expected velocities and the ones that are passed to the robot. Here, the expected velocities are the optimal velocities calculated by the solver and the real velocities are the commands that are actually given to it. As it is observable, there is negligible difference between these values, for both the linear and angular velocity cases. Remembering the assumption made in Subsection 3.6.2, regarding the equivalence between a constant acceleration model and the accepted constant velocities, it is possible to conclude that the assumption is completely valid.

Having confirmed the validity of the assumption made regarding the accepted control inputs, the error between the expected and real trajectories supports the argument that there are unmodelled vehicle dynamics. The used model is not accurate in its representation of the robot's real behaviour. Either due to unmodelled friction, accelerations or hardware delay, the confirmation that the used model is not accurate favours the use of a closed-feedback control technique, such is the case for the developed NMPC

controller. Further, if the controller is shown to have accurate tracking capabilities, then the developed work is not only robust to real-world conditions, but also robust to unmodelled vehicle dynamics.

4.3 Tuning of the controller

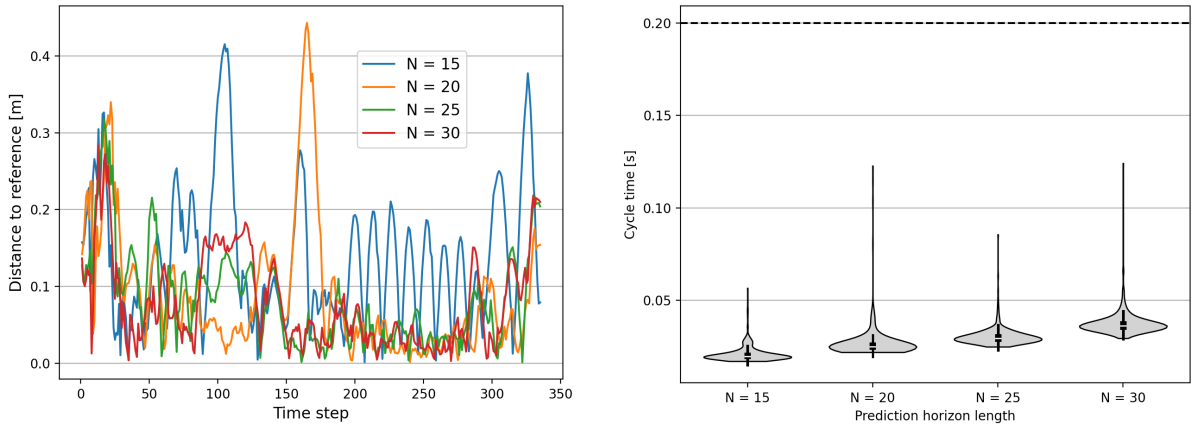
Two main aspects influence the performance of the developed NMPC controller: the prediction horizon length N and the weighting coefficient λ of the defined cost function. In this section, varying values will be studied for these two variables and the controller performance will be evaluated on both the Euclidean distance to the reference trajectory at each time step and computational efficiency. The used reference trajectory is the one shown in Figure 4.3. The remaining experiment's identifying criteria used for this study are as follows: $f_c = 5\text{Hz}$; $L = 0.633\text{m}$; $v_{\max} = 0.7\text{m}\,\text{s}^{-1}$; $a_{\max} = 0.5\text{m}\,\text{s}^{-2}$; $d_{\text{safe}} = 0.8\text{m}$; $voxel_{\text{size}} = 0.5\text{m}$; $max_{\text{range}} = 3.5\text{m}$; $data_{\min} = 0.2$; $data_{\max} = 0.8$ and $d_{\text{replan}} = 5\text{m}$.

It should be noted that the computational efficiency throughout this work was studied using violin plots. These plots feature the median of the data (represented as a white line), a box with the interquartile range (IQR) and whiskers that span 1.5 times the length of the IQR to flag possible outliers in the data. Moreover, this computational efficiency is studied regarding the full cycle times. The full cycle includes the entire action of what constitutes an NMPC, i.e., the whole process from getting the robot's initial position until actually passing the optimal controls to the robot.

4.3.1 Prediction horizon length influence

When testing for the influence of the prediction horizon length N on the controller's performance, it is expected that the greater the value for this parameter, the better the tracking results in terms of distance to reference, at the expense of a large increase in computational cost. The larger the prediction horizon, the farther into the future the controller is able to see, thus having improved anticipation. However, the computational complexity of having to calculate for a greater number of time steps eventually leads the solver to take up too much time and end up contributing negatively for the tracking of a trajectory. Note, however, that if the vehicle model is inaccurate, as was shown to be the case for this work, increasing the value for N may not produce beneficial results in terms of tracking. For these experiments, the chosen value for the weighting coefficient was $\lambda = 0.25$. The minimum value chosen for N was of $N = 15$, since values smaller than that produced catastrophic results. Prediction horizon lengths smaller than this provide too limited future information and the robot is unable to follow the trajectory.

The increase in computational cost could be mitigated if the initial guess given to the solver is accurate. To test this warm-start, the controller was tested in blind conditions. This is, the entire data treatment pertaining to the obstacle avoidance constraints was disabled. In Figure 4.6, the results for both the Euclidean distance to reference at each time step and computational efficiency are presented for this blind controller for varying values of N from $N = 15$ until $N = 30$, in increments of 5 time steps. Note that the oscillatory behaviour for $N = 15$ shows that the robot is unable to smoothly compensate for deviations. Due to its limited knowledge of the future, given by a smaller prediction horizon, it tries to



(a) Influence of the prediction horizon length N on the Euclidean distance to reference at each time step of a blind controller.

(b) Influence of the prediction horizon length N on computational efficiency of a blind controller.

Figure 4.6: Influence of the prediction horizon length N on the blind controller's performance.

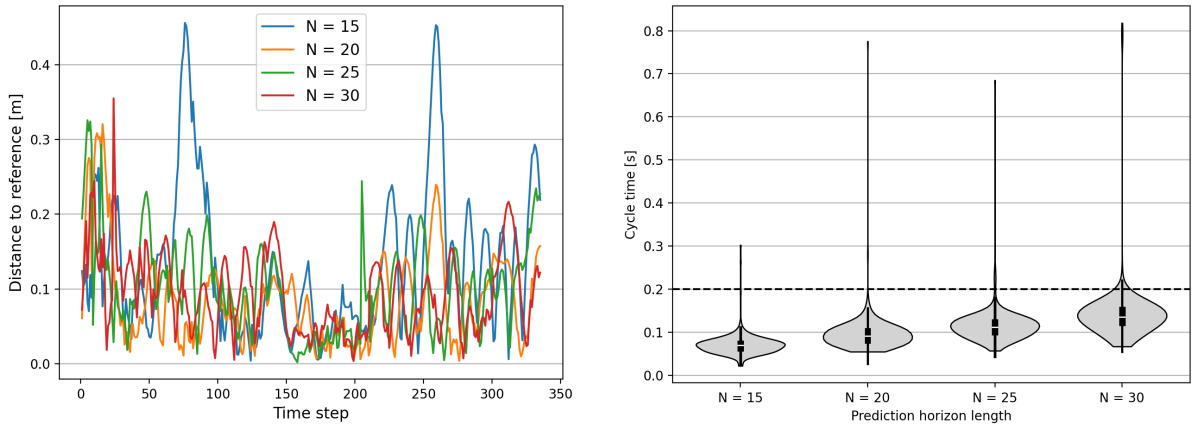
correct its deviation and return to the trajectory in a way that is too aggressive and leads to deviation in the opposite way, creating an oscillatory cycle. Focusing solely on the computational efficiency results, it is possible to see that the small increase in cycle time with the increase in prediction horizon length favours the argument that the initial guess procedure used for this work is correct. When the initial guess is accurate, the solver only needs to iterate over the last time step in the prediction horizon, as was explained in Subsection 3.7.3 of this document.

When analyzing the results for the prediction horizon length influence of the normal controller in Figure 4.7 for varying values of N from $N = 15$ until $N = 30$, in increments of 5 time steps, and especially when comparing the computational efficiency data with the blind controller results in Figure 4.6, it is possible to see how computationally expensive it is to calculate for a greater number of obstacle avoidance constraints. As the prediction horizon length increases, so does the computational burden of the cycle, with the results of $N = 30$ being unfit for online implementation, as the upper whisker goes beyond the threshold of 0.2 seconds.

Regarding tracking of the reference trajectory, as expected, the greatest error when measuring Euclidean distance to reference is seen for the smallest value of N. However, the opposite does not happen. The smallest overall error in tracking happens for $N = 20$, with a greater error across the duration of the trajectory for both the $N = 25$ and $N = 30$ cases. This shows that the negative impact of the increase in computational burden from the increase in the prediction horizon length N outweighs the positive impact from the additional anticipation benefit of said increase. The ideal prediction horizon length for the conditions considered is $N = 20$.

4.3.2 Weighting coefficient influence

The value defined for the weighting coefficient λ attributes more importance to either minimizing the distance to the reference at each time step or minimizing the difference between successive control inputs. Regarding the defined cost function, a greater value for the weighting coefficient attributes more



(a) Influence of the prediction horizon length N on the Euclidean distance to reference at each time step.

(b) Influence of the prediction horizon length N on computational efficiency.

Figure 4.7: Influence of the prediction horizon length N on the controller's performance.

importance to minimizing the successive control differences and vice-versa, regarding distance to reference. It is expected that the smaller the value of λ, the smaller the error in tracking. It is also expected that a change to this weighting coefficient value has little to no influence on the computational burden of the problem. The chosen prediction horizon length for this study was $N = 20$.

In Figure 4.8 it is possible to see both the Euclidean distance to reference at each time step and the computational efficiency for varying values of the weighting coefficient, from $\lambda = 0.0$ to $\lambda = 0.75$ in increments of 0.25. The maximum value of the value for this study was chosen to be below the unitary value, attributed as the weight for the distance to reference in the cost function. It desirable that the cost function attributes a lower significance to penalizing successive control inputs regarding the penalization of distance to reference, and, as such, the value of the weighting coefficient should be smaller than one.

The majority of the tracking data is in accordance with what was expected, with greater overall tracking results for $\lambda = 0.25$ and worse for both $\lambda = 0.50$ and $\lambda = 0.75$. Further, varying the weighting coefficient, as expected, appears to have no effect on the computational efficiency, as the violin plots show approximate values for the median, box and whiskers across the different values. In terms of the use case considered, the best value for the weighting coefficient seems to be $\lambda = 0.25$.

The worst results in terms of tracking happen for the value of $\lambda = 0.0$, which seems contrary to what might be expected. This happens because the absence of penalization in the robot's control inputs leads to the performance of aggressive maneuvers, which also jeopardizes the localization estimation. An aggressive, higher speed maneuver leads the robot away from the trajectory and will eventually lead the robot to overcompensate its behaviour when returning, resorting to another aggressive maneuver. This is particularly evident during the turnaround maneuver, where the error is much greater due to this aggressiveness. Not penalizing the successive control inputs has a negative effect on trajectory tracking.

4.4 Tracking a model trajectory

In this section, the tracking capabilities are studied, regarding the model trajectory in Figure 4.3. For statistical significance of the results, the studies conducted consider five experiments of both the developed controller and of TEB as a baseline solution.

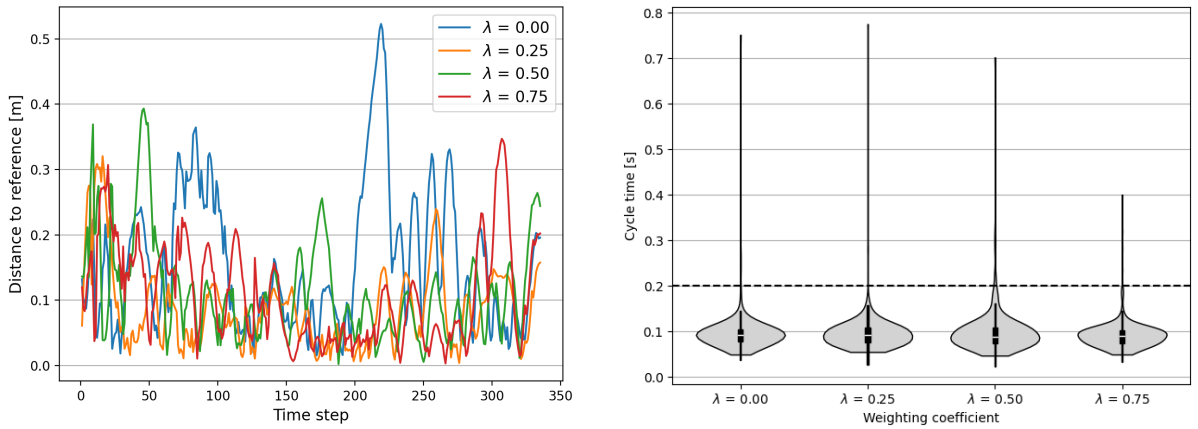
4.4.1 Controller performance

To test for the controller performance the used experiment's identifying criteria are as follows: $f_c = 5\text{Hz}$; $N = 20$; $\lambda = 0.25$; $L = 0.633\text{m}$; $v_{\max} = 0.7\text{m}\,\text{s}^{-1}$; $a_{\max} = 0.5\text{m}\,\text{s}^{-2}$; $d_{\text{safe}} = 0.8\text{m}$; $voxel_{\text{size}} = 0.5\text{m}$; $max_{\text{range}} = 3.5\text{m}$; $data_{\min} = 0.2$; $data_{\max} = 0.8$ and $d_{\text{replan}} = 5\text{m}$.

First and foremost, the controller is studied regarding its tracking abilities. At every time step over the course of the trajectory's duration, the robot's position $(x_r, y_r)(h)$ is measured against the reference at said time step $(x_{\text{ref}}, y_{\text{ref}})(h)$ by means of an Euclidean distance. In Figure 4.9 it is possible to see the overlaying of the tracking experiments to the reference trajectory. The plotting of a point-wise distribution of the trajectory instead of a continuous plot reflects the comparisons made at each time step. Further, in Figure 4.10 the Euclidean distance between the tracking and reference at each time step is shown over the trajectory's duration for every experiment.

The maximum tracking error of the controller is of 0.31 meters, which, given the robot's dimensions and environment, is not considered to be significant. Additionally, the controller's error in tracking is inflated by the tracking error introduced by the AMCL positioning estimate, already mentioned to be experimental, thus adding to the successful performance of the controller.

It is also possible to note that the robot is more accurate in its tracking at low-speed conditions, as is the case for the turnaround maneuver. This is an observation that favours the argument that the main driver of error is the AMCL positioning estimate. Because of a lagging mismatch between the odometry data of the motors and interpretation of that information into the AMCL algorithm, the robot temporarily misplaces the objects in the environment. This is shown in Figure 4.11, in which the laser



(a) Influence of the weighting coefficient λ on the Euclidean distance to reference at each time step.

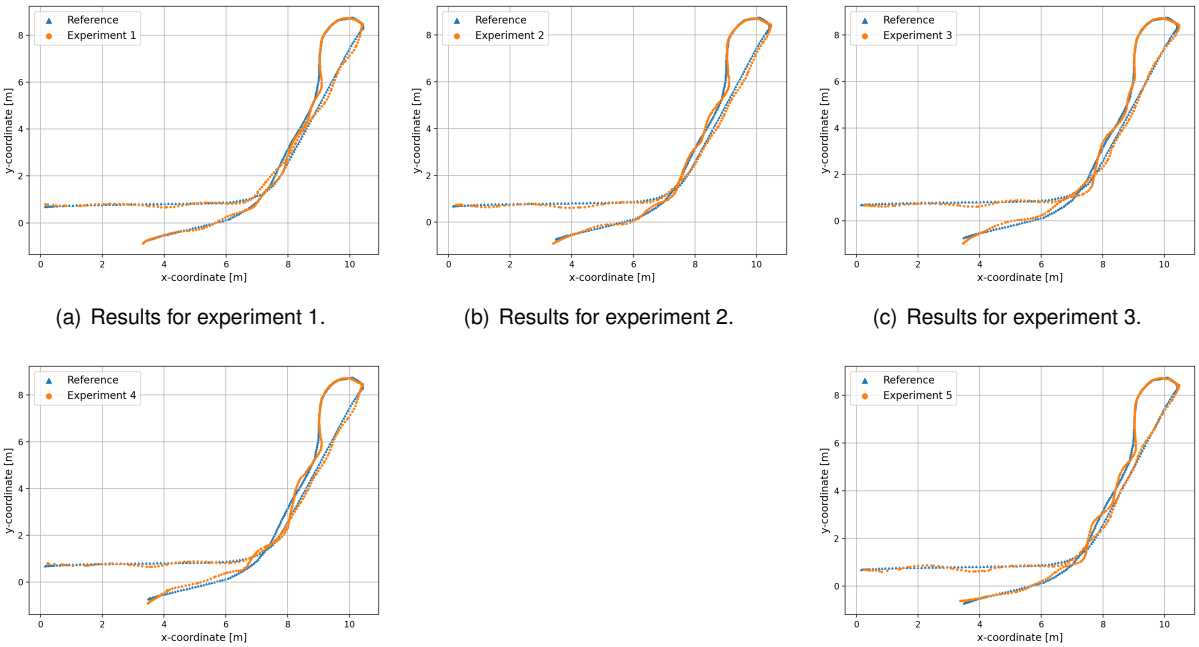
(b) Influence of the weighting coefficient λ on computational efficiency.

Figure 4.8: Influence of the weighting coefficient λ on the controller's performance.

scan is represented against the static map. When the robot rotates, the scan is misplaced against the static map, contributing to the malfunctioning of the AMCL positioning of the robot and, as such, the malfunctioning of the controller as it gets its position at every cycle from this estimate. The greater the velocity at which the robot turns, the greater this odometry mismatch and the greater the oscillations introduced in the robot's tracking. Moreover, the fact that the tracking is better at low-speed conditions, favours the argument that the trajectory generation algorithm should use a lower nominal velocity v_{nom} in its operation.

In order to measure the robot's ability to reach its target configuration, the final position of the robot should be compared to the reference goal. For the practical purposes of docking, as will happen in the scenario for which the algorithm was developed, explained in Chapter 1, this comparison also features the difference between the robot's final heading angle against its reference target heading angle. In Table 4.2, these errors are measured by the absolute value between the robot's position and the reference at each time step. From the interpretation of this data, the robot performs well in reaching its target configuration. The errors in the (x, y) coordinates and heading angle θ are negligible, when compared to the dimensions of the robot and the scenario considered. Moreover, these results further confirm that the choice of passing the last set of control inputs to the robot in an open-loop format has little to no impact on the controller's performance.

Finally, due to the online implementation nature of the NMPC framework, it is important to study the computational efficiency of the developed controller. This computational efficiency is studied taking into account the duration of three different actions: the full cycle time, the data treatment time and the solver time. The full cycle time has been previously explained in Section 4.3 and the data treatment times refer to the time it takes from receiving the occupancy map to defining the obstacle avoidance constraints, as



(a) Results for experiment 1.

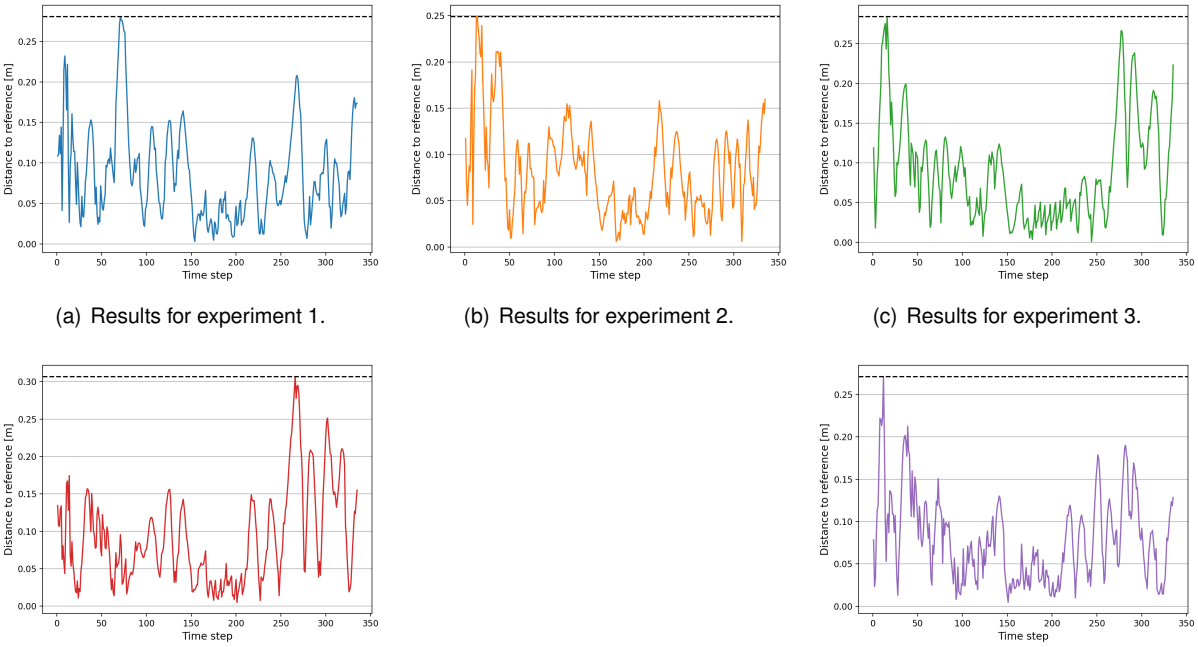
(b) Results for experiment 2.

(c) Results for experiment 3.

(d) Results for experiment 4.

(e) Results for experiment 5.

Figure 4.9: Trajectory tracking of the developed controller for five different experiments.



(a) Results for experiment 1.

(b) Results for experiment 2.

(c) Results for experiment 3.

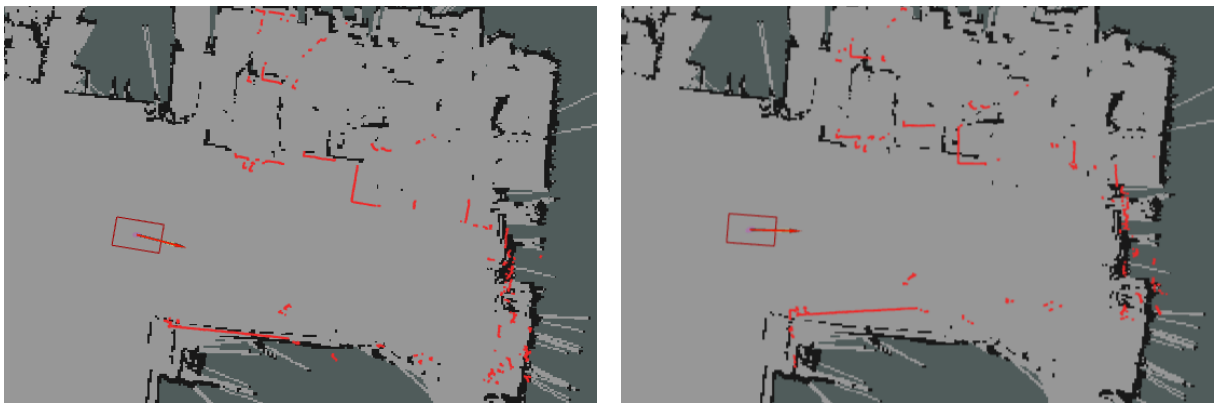
(d) Results for experiment 4.

(e) Results for experiment 5.

Figure 4.10: Euclidean distance to reference at each time time step for the five experiments of the developed controller.

is explained in Section 3.5. Finally, the solver time pertains to the time it takes for the CasADi solver to solve for the optimal control problem defined. The graphs in Figure 4.12 show violin plots for the three different actions' times of all five conducted experiments.

From the data, the controller performs in accordance with its computational efficiency requirements and is optimized in doing so. Apart from outliers, the computational times are all well within the threshold of 0.2 seconds, defined as a consequence of the controller's operating frequency of $f = 5\text{Hz}$. Note that an outlier is defined as a data point in which the computational time exceeds the defined threshold. Moreover, when defining an outlier as a data point in which the cycle time exceeds the defined threshold, there are only 8 outliers in a universe of 1580 data points. Though some outliers exceed the threshold



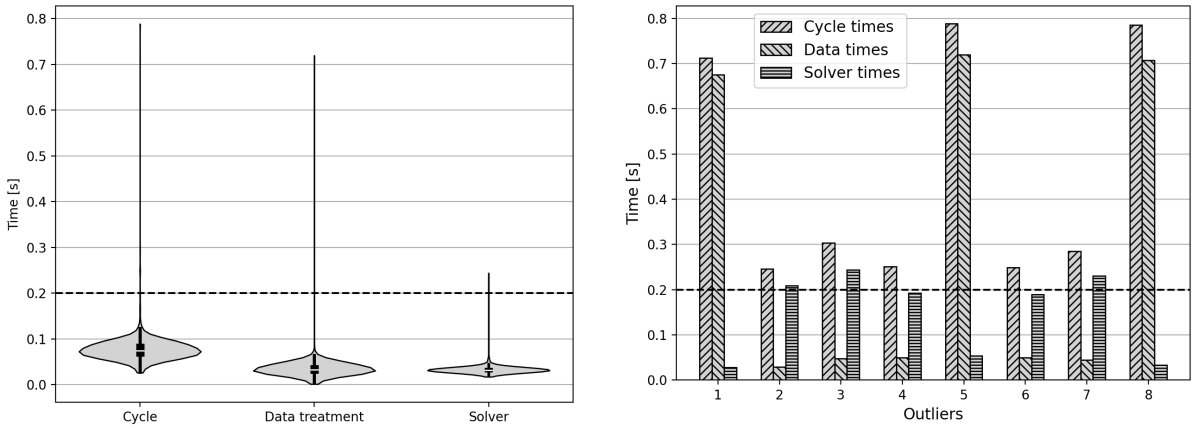
(a) Correct overlay of laser scan with static map.

(b) Error in overlaying laser scan to static map when turning the robot.

Figure 4.11: Example of AMCL positioning error when turning the robot.

Table 4.2: Distance to goal configuration for the controller's trajectory tracking experiments.

Experiment	**Error in x-coordinate** $\lvert\Delta x\rvert$ [m]	**Error in y-coordinate Δy [m]** $\lvert\Delta y\rvert$ [m]	**Heading angle error** $\lvert\Delta\theta\rvert$ [rad]
1	0.17	0.14	0.42
2	0.08	0.18	0.05
3	0.02	0.24	0.16
4	0.01	0.18	0.13
5	0.12	0.10	0.51
Average	0.08	0.13	0.03



(a) Computational efficiency for the tracking experiments.

(b) Outliers in computational efficiency.

Figure 4.12: Computational efficiency study for the trajectory tracking capabilities of the controller.

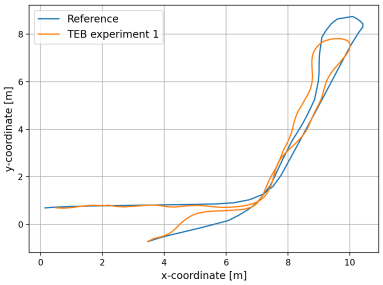
limit significantly, especially the ones whose main driver is a delay in the data treatment time, the closed feedback nature of the controller is able to handle these disturbances, as they are few and sparse. The small number of outliers due to delays in solver time further confirms that the chosen solver is fit for the use case considered.

4.4.2 Comparison with baseline solution

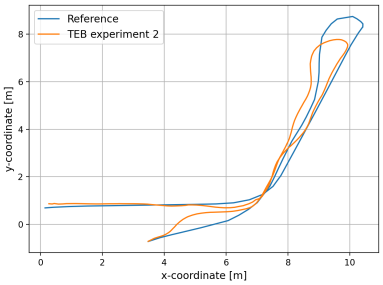
To test for TEB's trajectory tracking capabilities, the internal variables defined are shown in Appendix B. These parameters are not explained individually, as that is outside the scope of this work and are solely shown for replicability of the conducted experiments.[3] It is important to note, though, that some variables were set to be in accordance with the corresponding parameters of the developed controller. For example, both the maximum linear and angular velocities of TEB were set to match the maximum velocities of the controller and the look-ahead distance for planning of TEB set to match the value for the controller's max_{range} parameter value.

In Figure 4.13, the 5 different tracking experiments for TEB are shown. The most notorious behaviour from the experiments is that the robot was "cutting corners". This is particularly noticeable during the

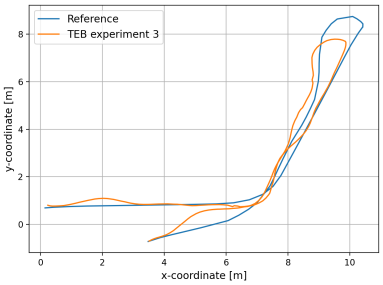
[3]For a detailed explanation on each of these internal parameters, please refer to the official website for the TEB local planner: `http://wiki.ros.org/teb_local_planner` (accessed October 9th, 2023).



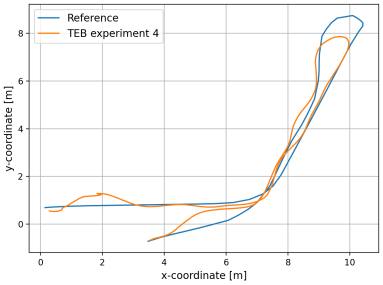
(a) Trajectory tracking for baseline TEB experiment 1.



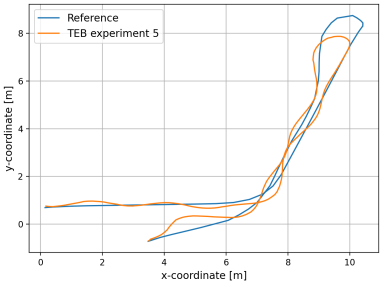
(b) Trajectory tracking for baseline TEB experiment 2.



(c) Trajectory tracking for baseline TEB experiment 3.



(d) Trajectory tracking for baseline TEB experiment 4.



(e) Trajectory tracking for baseline TEB experiment 5.

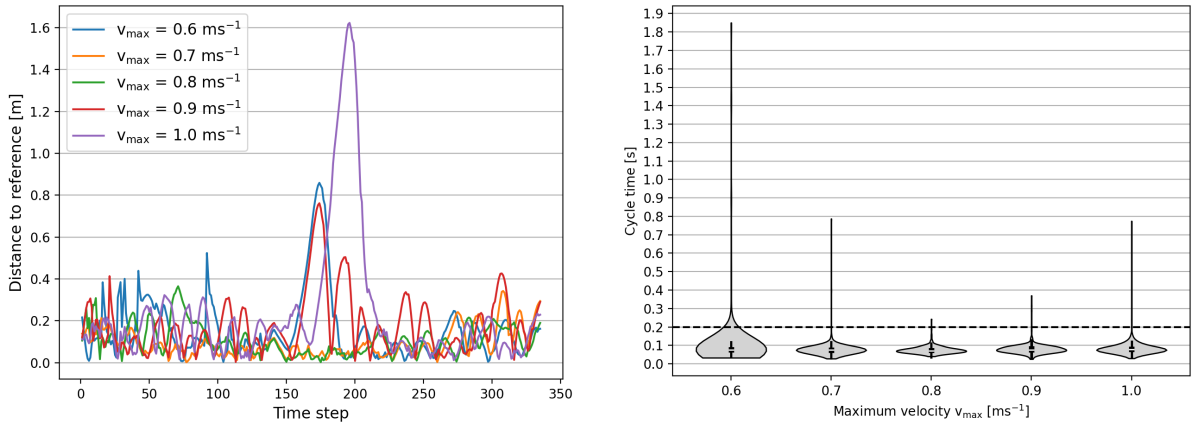
Figure 4.13: Trajectory tracking study of the developed controller for five different experiments.

turnaround maneuver and is due to the piece-wise optimization feature of TEB for its reference path. Moreover, there is significant oscillatory behaviour of the robot after both the right-hand and left-hand turns. The approach to the final configuration also shows great deviation from reference, since TEB greatly values accurate goal configuration over the tracking of the reference.

When comparing the results in Subsection 4.4.1 and in this subsection, the developed controller seems to outperform the baseline in regards to tracking of the given reference. The controller exhibits tracking behaviour that remains closer to the reference and also displays less pronounced oscillatory behaviour during the experiment. Though TEB is more accurate in reaching the target or goal configuration, the error introduced by the controller, comparatively, is not significant. Finally, as was mentioned previously, and is the main driver for this work, the controller is also able to follow a trajectory rather than a path, thus also having this as an advantage over the TEB local planner.

4.5 Maximum velocity study

One other aspect that is important to understand is how the tracking capabilities of the robot vary with the maximum velocity $v_{\max}$ that is given to it. As was mentioned previously, this velocity must be greater than the nominal velocity $v_{\text{nom}} = 0.6\text{m}\,\text{s}^{-1}$ at which the trajectory is generated and lower than the maximum allowable velocity, for safety reasons, of $v_{\max} = 1.0\text{m}\,\text{s}^{-1}$. This experiment was conducted for the same values as the tracking of a model trajectory, in Section 4.4, except, of course, for the maximum velocity values, which were made to vary from $v_{\max} = 0.6\text{m}\,\text{s}^{-1}$ to $v_{\max} = 1.0\text{m}\,\text{s}^{-1}$ in increments of $0.1\text{m}\,\text{s}^{-1}$. In Figure 4.14 it is possible to see both the tracking error for each velocity, measured as an



(a) Euclidean distance to reference at each time step for every maximum robot velocity.

(b) Computational efficiency graphs for every maximum robot velocity.

Figure 4.14: Tracking and computational efficiency results for varying maximum robot velocities.

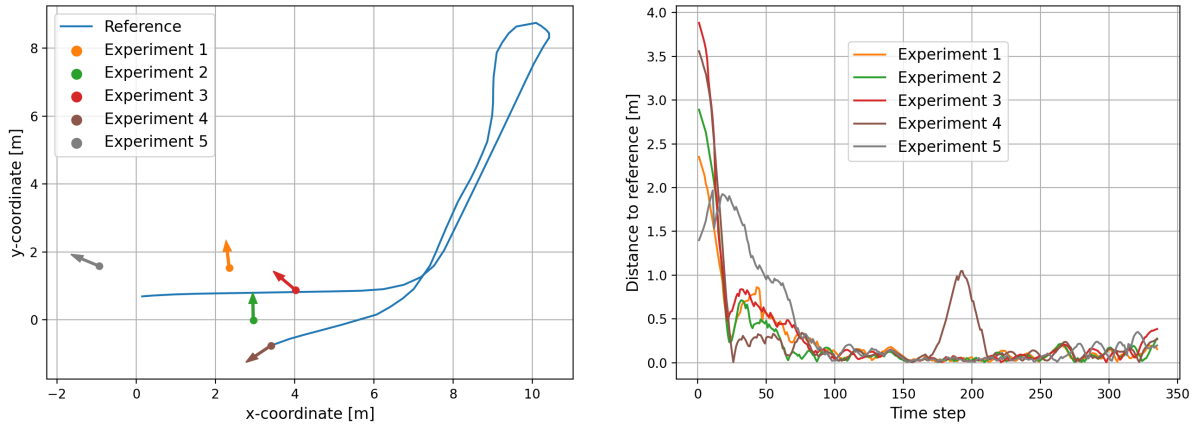
Euclidean distance from reference at each time step, and the computational efficiency for each of the varying velocity experiments, measured as total cycle times, in seconds.

For simplicity of notation in this text, the conducted experiments will be labeled from 1 to 5. Thus, the experiment that corresponds to $v_{\max} = 0.6\mathrm{m\,s^{-1}}$ will be experiment 1, the one that corresponds to $v_{\max} = 0.7\mathrm{m\,s^{-1}}$ will be experiment 2 and so forth until experiment 5, which will be the one that corresponds to the maximum allowable velocity of $v_{\max} = 1.0\mathrm{m\,s^{-1}}$.

From the Euclidean distance to reference graph, it is evident that the controller performs the worst for experiments 1, 4 and 5. These three all feature great tracking error for the low-speed turnaround maneuver, with experiment 5 clearly displaying the greatest imprecision, consistent with the fact that it is the experiment with the highest value for the maximum velocity. Both experiments 1 and 4 feature the greatest tracking error for the remainder of the trajectory, with experiment 1 being more accurate for the latter part of the reference trajectory and the other way around for experiment 4. This observation is consistent with the need for a value of $v_{\max}$ greater than v_{nom}, since experiment 1 shows this great error for the higher speed part of the trajectory, due to delays regarding its tracking.

Regarding the computational efficiency of these experiments, the cycle times are analyzed for each experiment. Experiment 1 is immediately noted to have a high percentage of outliers with great computational expense, almost to the point of it being unfit given the online implementation nature of the NMPC framework. A great part of the cycle times are beyond the upper whisker and the upper outlier is almost over 9 times the maximum allowed cycle time. For the remainder of the experiments, the box and whiskers are well within the established threshold of 0.2 seconds, though the upper whisker for experiment 4 has a greater value than the rest. The outliers of both experiments 2 and 5 are quite significant, when compared to the ones in experiments 3 and 4, but are of the same magnitude as the ones in Section 4.4, which have already been labelled as negligible, due to the scarcity of said outliers and given the closed loop feedback nature of the controller.

Finally, it is possible to see that the velocities that most benefit the controller's performance in terms of trajectory tracking are both $v_{\max} = 0.7\mathrm{m\,s^{-1}}$ and $v_{\max} = 0.8\mathrm{m\,s^{-1}}$. As was previously mentioned



(a) Initial position and heading angle of the robot for every experiment, compared to the reference trajectory.

(b) Decreasing Euclidean distance to reference trajectory.

Figure 4.15: Convergence of the robot to the reference trajectory when starting from a random point.

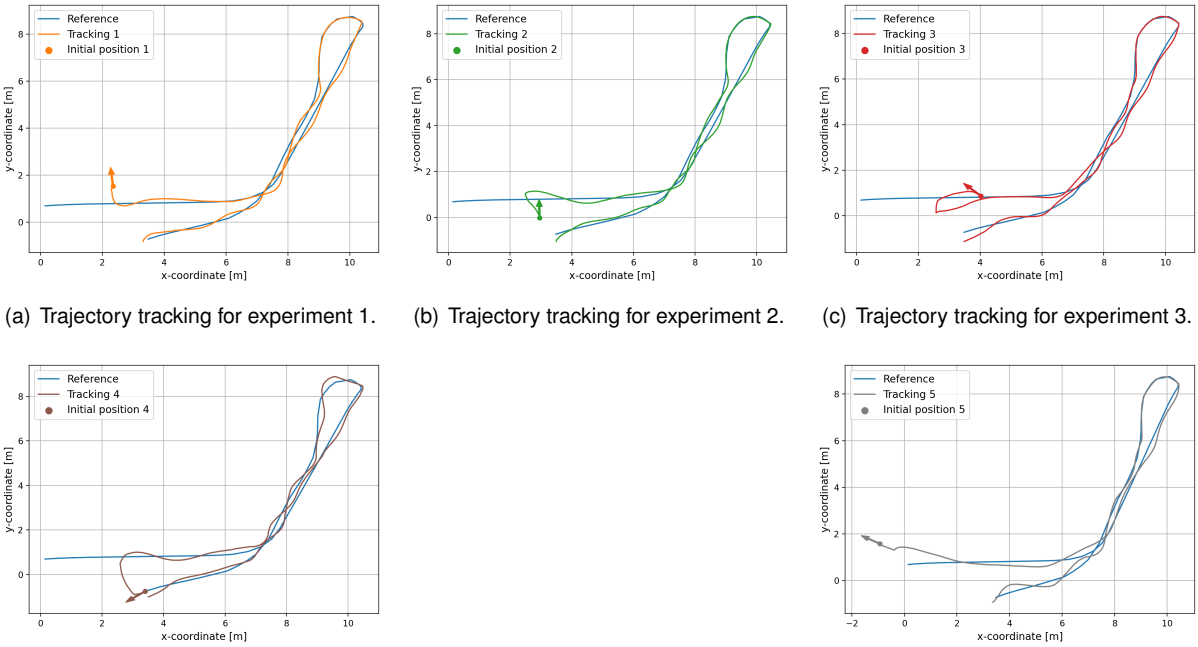
in this section, this is consistent with the assumed need for a value of the robot's maximum velocity $v_{\max}$ greater than the value of the velocity at which the trajectory is generated v_{nom}, however, this value should not be much greater. This $v_{\max}$ velocity should also be lower than the maximum allowable velocity $v_{\max} = 1.0\,\text{m}\,\text{s}^{-1}$ in order not to promote aggressive maneuvers, shown in Subsection 4.3.2 to worsen the robot's trajectory tracking.

4.6 Convergence study

Another significant aspect worth studying is understanding how the controller deals with discontinuities when in use. Either from errors in the AMCL positioning estimate, or errors from the trajectory generation algorithm, it is important that the robot is able to converge back to its reference trajectory when in a different position. To simulate these errors, the robot is placed at a random position at the initial time of the trajectory t_0. These positions were chosen having the available space as a constraint, in order to maintain an obstacle free trajectory, and are randomized for both the (x_r, y_r) robot coordinates and its heading angle θ_r. For these experiments, the parameters used were the same as the ones used in Section 4.4.

In Figure 4.15, it is possible to see the representation of the initial positions and heading angles of the robot for the different experiments. Further, it is possible to see the Euclidean distance to reference at each time step for these same experiments. Additionally, in Figure 4.16 the tracked trajectories are shown against the reference for every experiment. Finally, in Table 4.3 there are the initial configurations of the robot for the different experiments considered.

The controller performance is in accordance with what is expected in terms of robustness, converging to the reference trajectory inside 100 time steps, independently of the initial position of the robot. This convergence is particularly sensitive, when considering the trajectory aspect of the tracking. The controller "must decide" on whether it is more beneficial to wait for the trajectory to "catch-up" to the robot's current position or chase said trajectory. Given the results in this section, it appears that the



(a) Trajectory tracking for experiment 1.

(b) Trajectory tracking for experiment 2.

(c) Trajectory tracking for experiment 3.

(d) Trajectory tracking for experiment 4.

(e) Trajectory tracking for experiment 5.

Figure 4.16: Trajectory tracking for different convergence study experiments.

Table 4.3: Initial robot configuration for the convergence study experiments.

Experiment	**Initial x-coordinate** $x_r(t_0)$ [m]	**Initial y-coordinate** $y_r(t_0)$ [m]	**Initial heading angle** $\theta_r(t_0)$ [rad]
Reference	0.15	0.69	6.11
1	2.35	1.53	1.67
2	2.95	-0.01	1.58
3	4.03	0.87	2.36
4	3.40	-0.76	3.80
5	-0.93	1.58	2.69

choice in cost function is fit to dealing with disturbances and uncertainties.

For experiment 4, there is a significant error in tracking for the turnaround maneuver. Though this error is large, especially when comparing to the maximum error for the controller's tracking performance in Section 4.4, this is considered to be an outlier, usual when testing in changing real-world conditions.

4.7 Discussion of the results

Addressing the results in this chapter as a whole, the controller is shown to outperform the TEB baseline solution for tracking a given reference. Additionally, the controller is also shown to perform better for tracking at lower speeds rather than higher speeds, which indicates that the AMCL positioning estimate is the main driver of error and oscillatory behaviour.

Regarding tuning of the controller, it was shown that, for tracking performance, it is preferable that the weighting coefficient has a lower, non-zero, value. Moreover, a trade-off in performance is shown

between the increase in anticipation with a longer prediction horizon and the resulting increase in the computational burden of the problem. The validity of the assumption that tracking should consider a maximum velocity larger than the maximum velocity at which the trajectory is generated was also confirmed.

The controller is shown to be robust to discontinuities in its operation. It is able to converge back to the trajectory when starting at a random configuration. Additionally, having proven that there are unmodeled vehicle dynamics and that the assumed equivalence in control inputs is valid, the shown tracking performance of the controller accentuates its robustness to uncertainties.

Finally, regarding computational efficiency, the chosen solver is shown to be fit for the use case considered. A further study of the computational efficiency outliers show them as few and sparse. The computational efficiency of the controller was shown to be better for lower values of the prediction horizon length N and higher values of the robot's maximum velocity $v_{\max}$, with the weighting coefficient λ having little to no impact on said efficiency.

Chapter 5

Obstacle avoidance results

In this chapter, the obstacle avoidance capabilities of the controller are studied, and compared against the TEB baseline. First, the effect of voxel cell size on the computational efficiency of the controller is studied. Then, the controller's static obstacle avoidance capabilities are studied, for different geometries of obstacles. Additionally, the dynamic obstacle avoidance capabilities of the controller are shown for scenarios in which there's an obstacle directly crossing the robot's way and for when the robot has to overtake an obstacle going in the same direction as its trajectory. Finally, the trajectory replanning capabilities of the controller are shown.

5.1 Methodology

The methodology in this chapter follows the same principles as mentioned in Chapter 4. The environment, AMCL positioning estimate parameters and the robot are the same. The trajectory that is given to the robot will be different depending on the study conducted, in order to better fit the needs of said study. If nothing is indicated otherwise, the values used for the identifying criteria in this chapter's experiments are as follows: $f_c = 5\text{Hz}$; $N = 20$; $\lambda = 0.25$; $L = 0.633\text{m}$; $v_{\max} = 0.8\text{m}\,\text{s}^{-1}$; $a_{\max} = 0.5\text{m}\,\text{s}^{-2}$; $d_{\text{safe}} = 0.8\text{m}$; $voxel_{\text{size}} = 0.5\text{m}$; $max_{\text{range}} = 3.5\text{m}$; $data_{\min} = 0.2$; $data_{\max} = 0.8$ and $d_{\text{replan}} = 10\text{m}$.

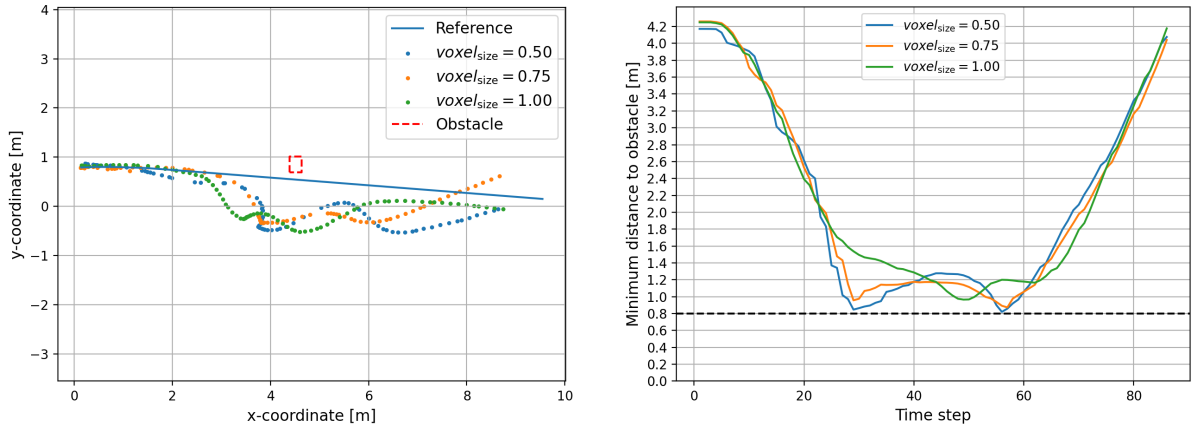
Considering data treatment, when testing for these obstacle avoidance capabilities of the controller, the most important criteria to assess is whether or not the robot maintains a distance greater than the safety distance from the obstacle introduced in its environment. Moreover, it is important to understand how swiftly the robot returns to the original trajectory after performing the needed avoiding action. Finally, as was done in the previous Chapter 4, it is paramount that the computational efficiency of the algorithm is studied for every experiment considered. Throughout this chapter, most of the avoiding action is done on the right-side of the obstacle. It should be noted that this is not a preference of the robot, but happens because there is a space constraint on the left-hand side of the obstacle. Due to this space constraint, it is not possible for the robot to perform its avoiding action considering both the voxel cell size and safety distance used in these experiments.

5.2 Study of voxel cell size on computational efficiency

It is also important to study and understand how much the defined voxel cell size impacts the computational efficiency of the developed controller. To do this, a rectangular-shaped obstacle was put in the way of the robot and the voxel cell size was made to vary between $voxel_{\text{size}} = 0.50\text{m}$ to $voxel_{\text{size}} = 1.0\text{m}$, in 0.25 meter increments. The minimum value was chosen since all the experiments ran for $voxel_{\text{size}} = 0.25\text{m}$ had catastrophic results. Moreover, the maximum value was chosen taking into account the space restrictions of Imeguisa's factory floor. Note that the cells in the voxel grid are defined as squares.

In Figure 5.1, the results for the obstacle avoidance capabilities of the controller are shown in terms of both the trajectory of the robot and the minimum distance to the obstacle for every voxel cell size considered. The minimum distance to obstacle is an Euclidean distance calculated at each time step between the robot's position and the closest point of the obstacle shape. To do this, the obstacle shape is discretized into points with a distance between of one millimeter. This distance between discretized obstacle points has little impact on the computational expense of the calculations and using this value guarantees that no further error is introduced at this stage. Though the tracking varies in shape, it is possible to see that the robot performs its avoiding action in accordance with what is expected. The robot is also successful in both maintaining a distance greater than the safety distance from the obstacle at all times and in converging back to the trajectory after avoiding the obstacle in its way. Further, in Table 5.1, it is possible to see that the values for the minimum distance to obstacle increase as the voxel cell size increases. This is because, for a bigger cell size, the obstacle points considered, i.e., the midpoints of said cells, are further apart.

A note should also be made on two characteristics of the robot's trajectory for the different voxel cell sizes. The first one is that the robot does not reach the end of its reference trajectory. It ends the tracking behind it. This is because of the controller's nature of performing trajectory, rather than path, tracking, i.e., following a time-parameterized reference. Even for a maximum robot velocity $v_{\max}$ greater



(a) Representation of reference, obstacle and tracking for every voxel cell size.

(b) Minimum Euclidean distance to obstacle at each time step for every voxel cell size.

Figure 5.1: Trajectory and minimum distance results for voxel cell size study.

Table 5.1: Minimum distance to obstacle for voxel cell size study experiments.

Experiment	**Voxel cell size** $voxel_{\text{size}}$ [m]	**Minimum distance to obstacle** $(d_{r\to o})_{\min}$ [m]
1	0.50	0.82
2	0.75	0.87
3	1.00	0.97

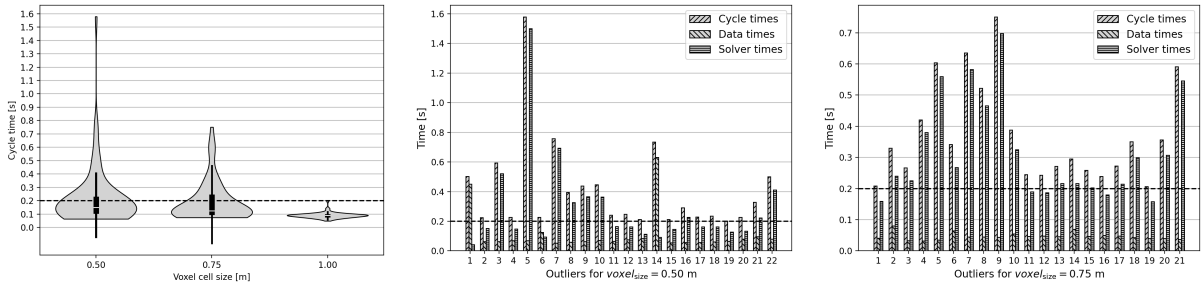
than the nominal velocity v_{nom}, the considered small reference trajectory is not enough for the robot to compensate for the delay due to the needed avoiding action. The robot lacks enough space after the obstacle in order to realign itself with the reference. The second characteristic is that the robot does not perform one simple movement, but rather tries to rejoin the reference trajectory before it is able to, given the defined obstacle avoidance constraints. The robot is still subject to some blindspots, as was the case for the laser scan approach explained in Section 3.5. The robot does not have the needed computational resources in order for the controller to be able to consider a space behind the robot large enough so that this does not happen.

In Figure 5.2, the computational efficiency data is shown for the different voxel cell sizes considered. From the violin plots, it is observable that both the cycle times for $voxel_{\text{size}} = 0.50\text{m}$ and $voxel_{\text{size}} = 0.75\text{m}$ far exceed the computational threshold imposed by the controller's operating frequency. In both scenarios, the third quartile already exceeds this value, with the upper whisker exceeding twice the value of the threshold. For the case of $voxel_{\text{size}} = 1.00\text{m}$, however, every cycle time respects the imposed upper threshold. Again, defining as a computational outliers every iteration for which the cycle time exceed the threshold of 0.2 seconds, it is possible to see that the volume of outliers for these cases is quite significant, when considering the duration of the trajectory.

When analyzing the time distribution of the computational outliers, notice that the main driver of error are the solver times, rather than the data treatment times. Having proven the efficiency of the solver for trajectory tracking in Subsection 4.4.1, these results show the great computational expense of having to satisfy the obstacle avoidance constraints. Having to satisfy for a smaller feasible region of optimal controls, when in the presence of an obstacle, whilst also having an initial guess that is much further from the optimal controls leads the solver to resort to a number of iterations that is just too large. Either the use of voxel cell sizes greater than 1 meter or the use of a computer with more processing power is needed if for safe implementation of the controller on factory floor. On the contrary, the small number of outliers due to the data treatment times show that the coded algorithm of this work is optimized.

5.3 Static obstacle avoidance

In this section, the controller's avoidance capabilities are studied, regarding static, i.e. not moving, obstacles. To do this, different obstacles were put in the robot's way and the followed trajectory tracked against the reference. Having already shown in the previous Section 5.2 that the robot is able to deviate from a rectangular-shaped obstacle, in order to prove that the developed work promotes successful



(a) Violin plots for cycle times for every voxel cell size.

(b) Computational efficiency outliers for $voxel_{\text{size}} = 0.50\text{m}$.

(c) Computational efficiency outliers for $voxel_{\text{size}} = 0.75\text{m}$.

Figure 5.2: Computational efficiency data for voxel cell size study.

avoiding action whilst being agnostic to the shape and dimensions of the obstacle considered, different obstacles were experimented, namely a parallelepiped-shaped obstacle, a cylinder-shaped obstacle and, finally, a cone-shaped obstacle. For a more simple representation of the results, both the controller's and TEB's performance data will be shown in the same graphs.

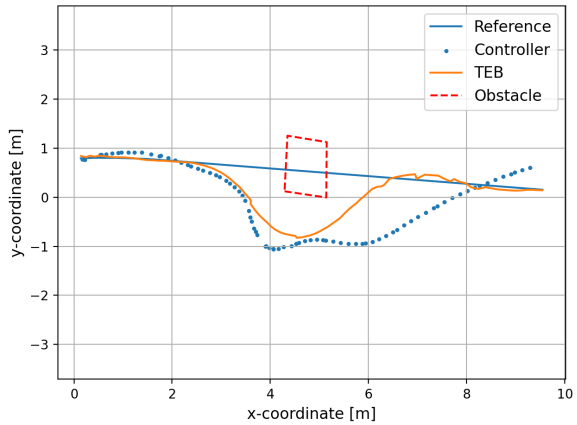
Due to the constraints in the available space for testing, the used voxel cell size was of $voxel_{\text{size}} = 0.50\text{m}$. From the previous Section 5.2, it is already expected that the computational efficiency results for these cases may not be up to the needed standard. Nevertheless, one of the most notable features of the developed work is that the controller is easily tunable to fit the needed use case or situation, through changes in the values for the experiment's identifying criteria. Thus, knowing already that it is possible to achieve the needed computational requirements through the use of a larger voxel cell size or more powerful computer, if the obstacle avoidance capabilities are proven for smaller cell sizes, then the controller is deemed fit for its real-world implementation. Additionally, considering that the expected use of the controller considers a much larger available navigable area, the expected used voxel cell size will also be larger.

5.3.1 Parallelepiped-shaped obstacle

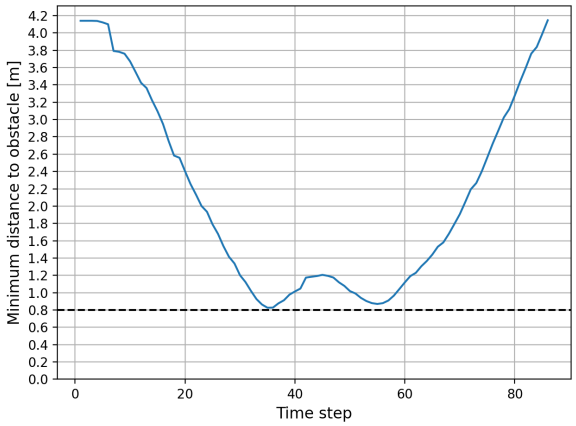
In Figure 5.3, the obstacle avoidance results are shown for a parallelepiped-shaped obstacle put in the robot's way. It is possible to see that the controller is effective in deviating from the impediment. It maintains a minimum distance to obstacle greater than safety at all times, with a minimum value of this distance of 0.83 meters. Moreover, the robot is able to converge back to the reference trajectory after performing the needed avoiding action.

Regarding the computational efficiency of the experiment, as expected, though the results are better than the ones in the previous Section 5.2, the number of outliers are quite large. Again, the main driver of going beyond the defined computational efficiency threshold is proven to be an excess in solver times. It is important to note that though this error in computational efficiency may cause some drift, both the inflation layer of the obstacles and considering the voxel cell midpoints as the obstacle points lessen the adverse drifting consequences of said temporal excess.

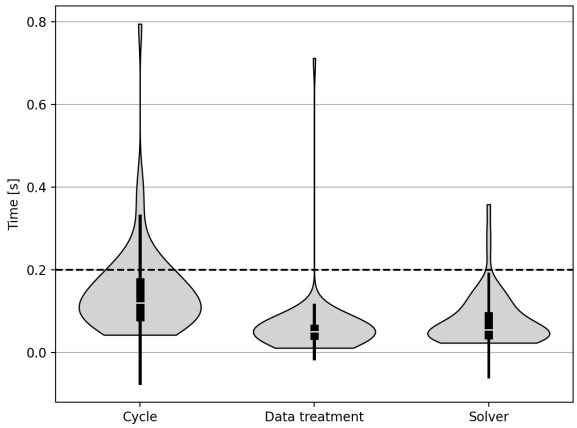
When comparing to the TEB baseline, it is noticeable that due to the path following nature of this local planner, it is able to perform the avoiding maneuver in a more continuous manner. The speed at which



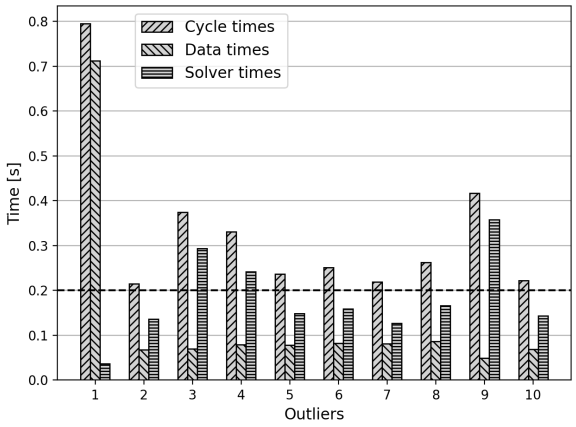
(a) Representation of reference, controller's and TEB's trajectory and obstacle position and dimensions.



(b) Minimum distance to obstacle at each time step.



(c) Computational efficiency violin plots.



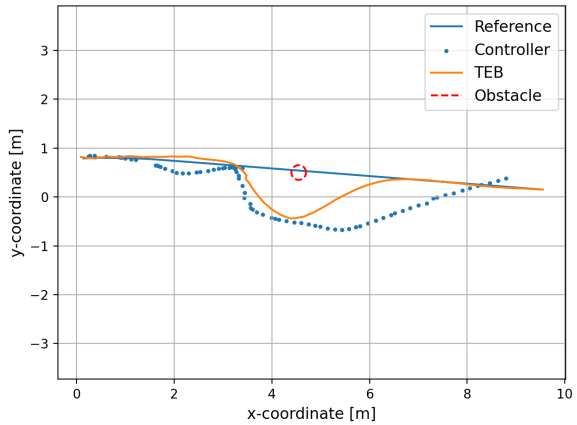
(d) Computational efficiency outliers.

Figure 5.3: Results for avoidance from a static parallelepiped-shaped obstacle.

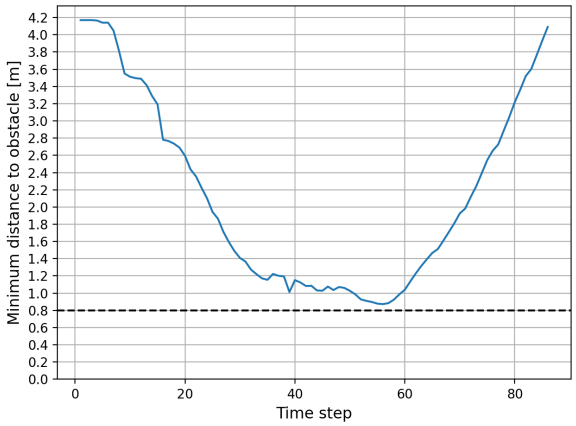
TEB performs the avoiding action is much slower than the one at which the controller does it. Having a minimum distance to the obstacle of 0.49 meters, it is possible to say that TEB performs a smoother avoiding action because it is not constrained by the trajectory following aspect that is one of the main drivers for the development of this controller and of this work. Note that TEB must maintain a distance of at least half the width of the robot from the robot, i.e., must be at a distance greater than 0.3475 meters from the obstacle at all times.

5.3.2 Cylinder-shaped obstacle

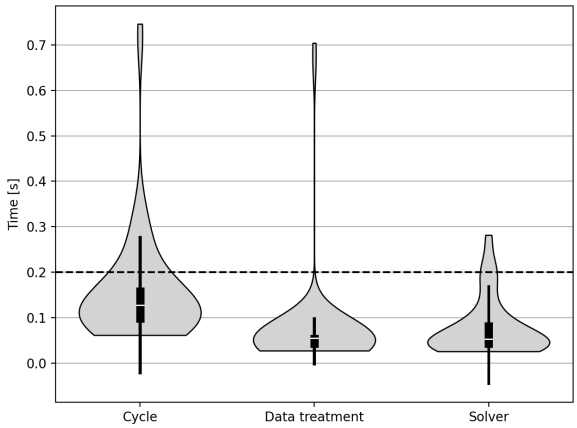
In Figure 5.4, the obstacle avoidance results are shown for a cylinder-shaped obstacle put in the robot's way. It may be observed that the controller is effective in deviating from the obstacle and maintaining a distance greater than the safety distance from the obstacle at all times. Additionally, it is effective in converging back to its reference trajectory after performing the needed avoiding action. The minimum distance from the obstacle for the controller is of 0.87 meters. For the baseline TEB it is of 0.71 meters. It is also observable that, again, TEB is smoother in both the deviation of the obstacle and the returning back to the reference, due to its path, rather than trajectory, tracking.



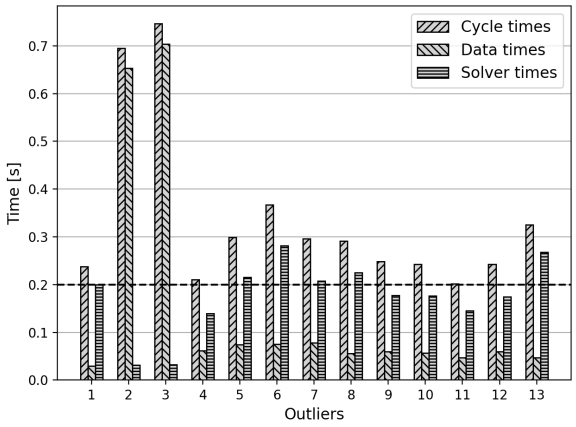
(a) Representation of reference, controller's and TEB's trajectory and obstacle position and dimensions.



(b) Minimum distance to obstacle at each time step.



(c) Computational efficiency violin plots.



(d) Computational efficiency outliers.

Figure 5.4: Results for avoidance from a static cylinder-shaped obstacle.

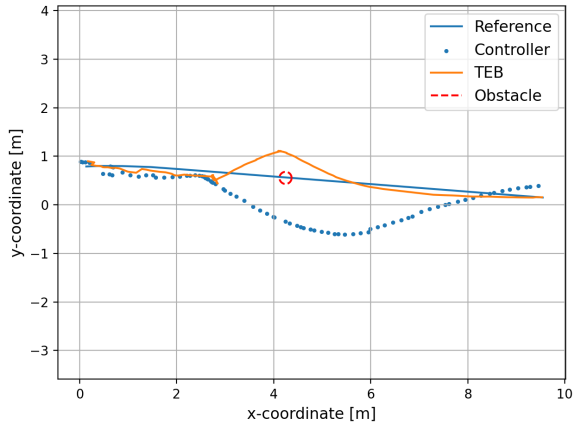
It is noticeable that, in this case, for an obstacle of smaller dimensions, thus requiring less avoiding action, the robot performs one continuous maneuver. There is no attempt to prematurely rejoin the trajectory, as there are no blindspots. When addressing implementation in the real factory environment, and considering the tunable nature of the developed controller, the expected obstacle dimensions should be taken into account when setting the values for $voxel_{\text{size}}$, max_{range}, $data_{\text{max}}$ and $data_{\text{min}}$, in order to allow for this smoother, more continuous obstacle avoidance behaviour whilst maintaining the needed computational requirements.

Further, regarding computational efficiency, as expected, the solver is better at finding the optimal controls needed to perform the avoiding action. When comparing the computational efficiency with the results for a larger obstacle in Subsection 5.3.1, it is also noticeable that the solver is much more efficient in this case, thus contributing for a lower overall value for the cycle time. In this case, the upper whisker does not go beyond 0.3 seconds and the interquartile range is well within the threshold of 0.2 seconds. This is an expected behaviour of the controller, since the efficiency of the solver has been proven for conditions with no obstacles. As such, a smaller obstacle, i.e., a smaller deviation from the behaviour in no obstacle conditions, induces less solver error.

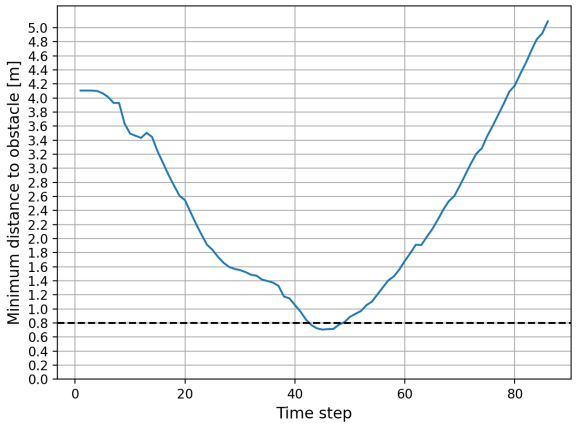
5.3.3 Cone-shaped obstacle

In Figure 5.5, the obstacle avoidance results are shown for a cone-shaped obstacle placed in the robot's way. Though the avoiding action is smooth and the robot is successful in converging back to the reference, the minimum distance from the obstacle is not greater than the safety distance at all times. The minimum value for this distance is of 0.71 meters. In fact, due to the laser being mounted on the robot at a height, the cone is interpreted as an obstacle shaped as a smaller circle than that of its base. This hardware limitation could be mitigated by considering a larger voxel cell size $voxel_{\text{size}}$, which was proven in the previous Section 5.2 to be effective in increasing the minimum distance to the obstacle.

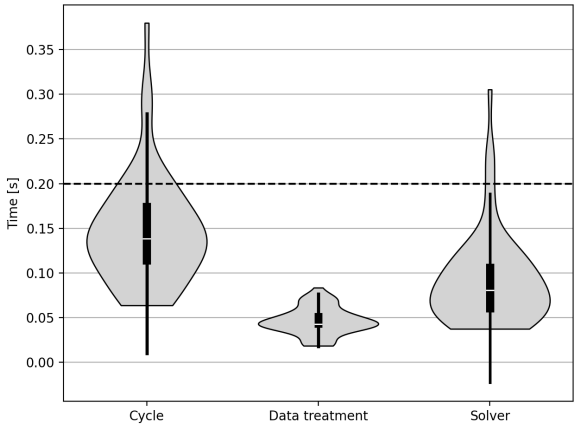
Regarding both the analyses of TEB as a baseline and computational efficiency, the results are comparable to those of the previous Subsection 5.3.2. As a baseline, TEB is effective in its avoiding action, reaching a minimum distance from the obstacle of 0.35 meters. The avoiding action is, for the first time, seen to be done on the left-hand side, since, in this case, the obstacle dimensions allow for it in the case of TEB. As expected, a smaller obstacle proves to have better results regarding the cycle and solver times, due to proving to be a smaller deviation from the no-obstacle trajectory tracking conditions.



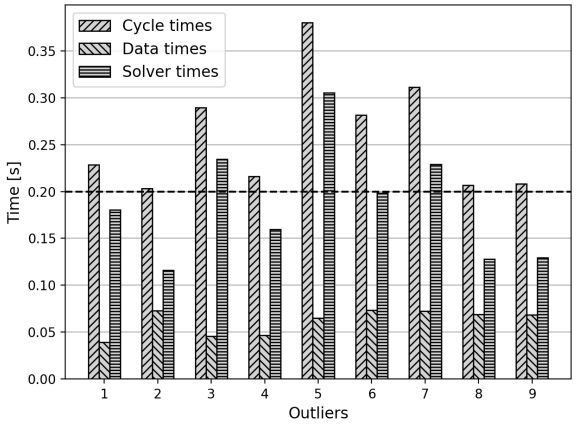
(a) Representation of reference, controller's and TEB's trajectory and obstacle position and dimensions.



(b) Minimum distance to obstacle at each time step.



(c) Computational efficiency violin plots.



(d) Computational efficiency outliers.

Figure 5.5: Results for avoidance from a static cone-shaped obstacle.

5.4 Moving obstacle across trajectory

In order to test for the controller's capability of avoiding moving obstacles, two experiments were conducted. The first experiment, in this section, simulates an obstacle perpendicularly crossing the robot's way. In Figure 5.6, the experiment setup used is shown. A worker, maintaining an approximately constant speed, pushes a box across the robot's way, moving from the right to the left of the robot, as indicated by the arrow. The dashed line represents the given reference trajectory. This experiment aims at simulating as closely as possible the type of impediments that the controller will encounter when used as part of the AGiLE project for which it is developed.

When considering the replicability of the conducted experiments, the lack of knowledge regarding the worker's exact velocity is not ideal. This was, however, the only way to conduct an experiment of this type, since no other robots were available to use. For parity when comparing both the controller's and TEB's performance, the worker, as mentioned, attempted to maintain a constant velocity throughout the conducted experiments.

When performing the avoidance of dynamic obstacles, as was previously mentioned, there is no need for an *a priori* knowledge of the robot's trajectory. Instead, by doing the occupancy map treatment and recalculating the obstacle avoidance constraints for every cycle, the controller is able to account for the movement of the obstacle. The operating frequency of the controller is responsible for tracking the movement of the obstacle and setting constraints that allow for the avoiding action of a moving obstacle.

In Figure 5.7, it is possible to see the avoiding action for both the developed controller and the TEB baseline tracked against the reference trajectory. Though more clear for the controller's case, it is evident that in both scenarios there is an attempt to turn to left before eventually avoiding the obstacle on its

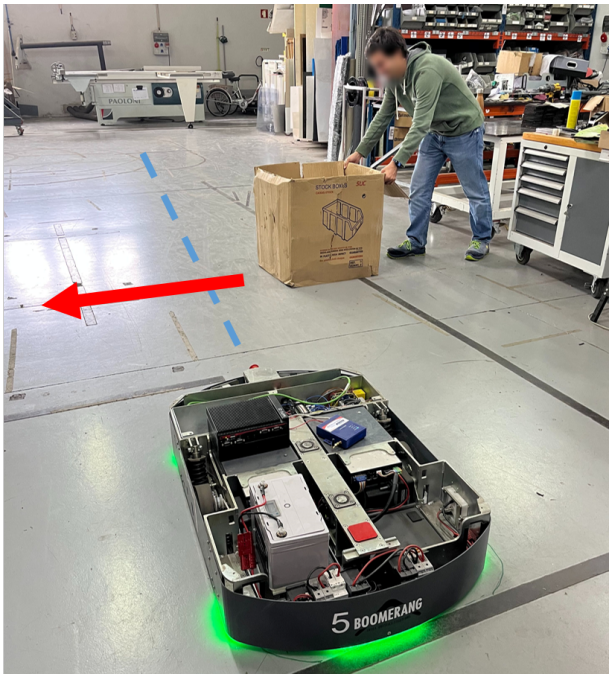
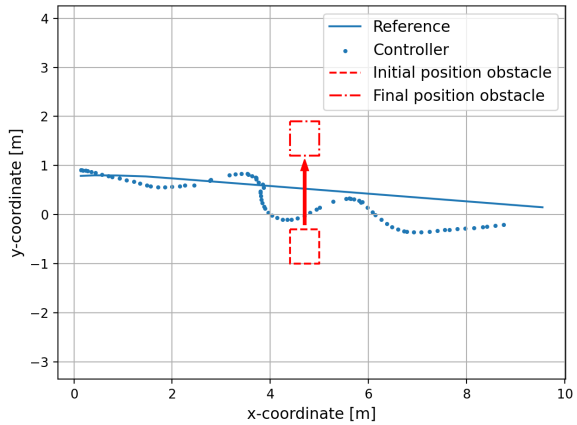
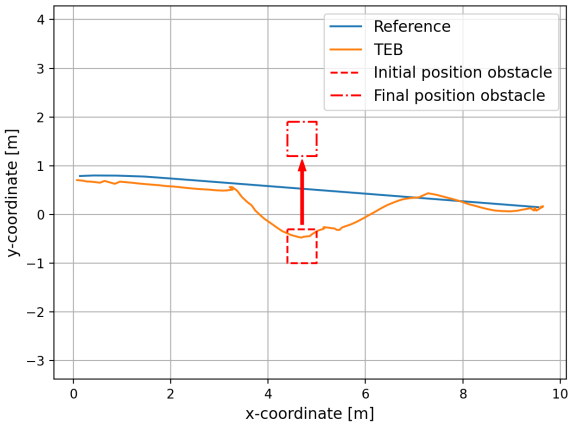


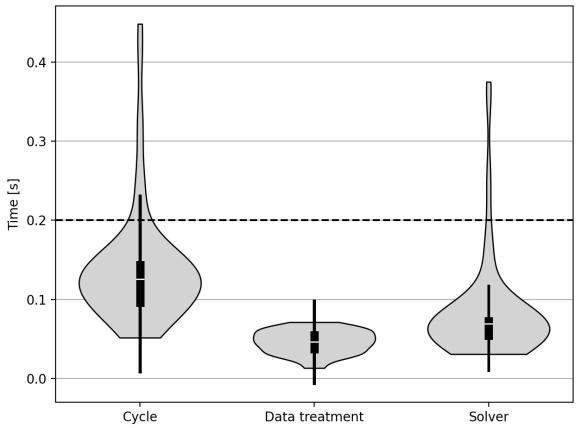
Figure 5.6: Representation of the experiment setup for a moving obstacle across the trajectory of the robot.



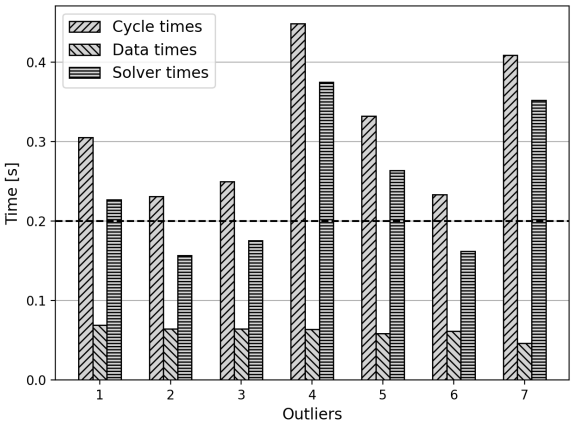
(a) Representation of reference, controller's and obstacle's trajectory.



(b) Representation of reference, TEB's and obstacle's path.



(c) Computational efficiency violin plots.



(d) Computational efficiency outliers.

Figure 5.7: Results for avoidance from a dynamic obstacle crossing across the robot's way.

right-hand side. This behaviour shows the adaptability of both the developed controller and TEB to the moving aspect of the obstacle. In the case of the controller, this behaviour proves valid the assumption that the recalculation of the obstacle avoidance constraints at every controller cycle allows to account for the movement of dynamic obstacles. Note that the minimum distance to the obstacle is not possible to be accurately tracked, since there is no information about the worker's travelling velocity, and, as such, is not presented in this section. Due to the path tracking nature of TEB it is, again, smoother in converging back to the reference, when comparing to the developed controller.

Moreover, when analysing the results for computational efficiency, it is possible to see that these prove much better than for other obstacle avoidance scenarios already analysed. The recalculation of the obstacle avoidance constraints at every cycle, i.e. the data treatment times, is never the main driver of cycle time delay. This is, if the solver is possible to be made more efficient, as was shown to be possible in Section 5.2 by increasing the voxel cell size, there is a possibility of having an obstacle avoidance with no computational efficiency outliers.

Finally, it is possible to state that both the developed controller and the TEB baseline have comparable performances when performing obstacle avoidance of an impediment crossing the robot's way

perpendicularly. In both scenarios, the robot is able to avoid the obstacle completely, accounting for its right to left trajectory, as indicated by the arrow in the graphs and the experiment setup in Figure 5.6.

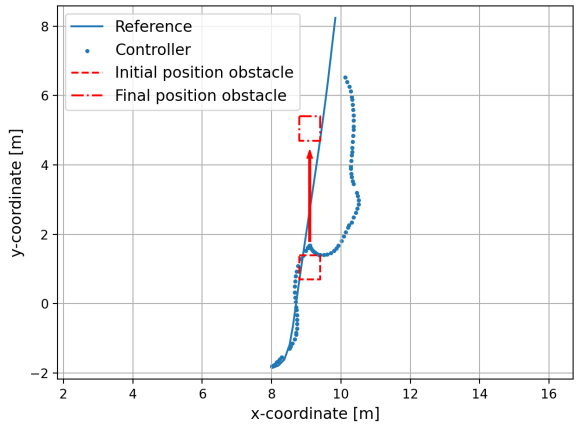
5.5 Overtake study

The second experiment, aimed at testing the dynamic obstacle avoidance capabilities of the developed controller focuses on the ability of the robot to overtake an obstacle going the same way as its reference trajectory. In Figure 5.8, the experiment setup is shown. Similarly to the circumstances in the previous Section 5.4, this experiment was designed in order to simulate as closely as possible the real conditions the controller might face when implemented as part of the project it integrates. Again, for replicability of the conducted experiment, it is not ideal that it is not possible to exactly track the worker's velocity. Nevertheless, he is instructed to attempt to maintain a constant velocity throughout.

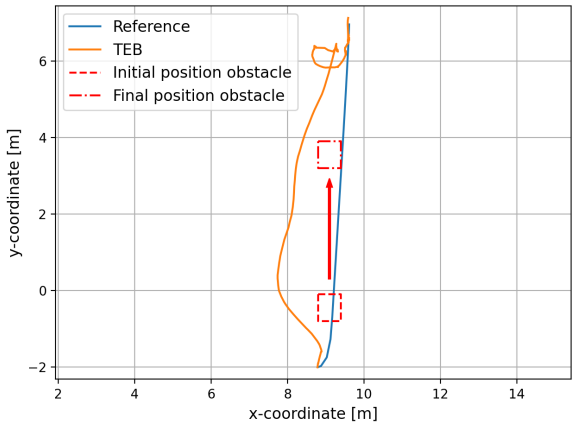
In Figure 5.9, the results are shown regarding the overtake of a dynamic obstacle for both the controller and the TEB baseline. It is evident that both techniques are able to avoid the obstacle, however, they behave differently when converging back to the trajectory. The developed controller is able to overtake the obstacle, keeping alongside it, whilst following the trajectory as closely as possible. As soon as the trajectory duration is over, the controller finishes its tracking. On the contrary, TEB is unable to finish its tracking if not by reaching the defined final or target configuration of the robot.

In the case of this experiment, the obstacle box is not pulled by the worker beyond the points represented in the graph as the obstacle's final position. However, if the obstacle was to be pulled up to a point where it would be covering the final position of the trajectory, TEB would have been unable to complete its tracking. These were the problems first described in Subsection 1.3.3 regarding overtaking for TEB. Since the robot must get to its final configuration, problems arise when there are situations with two robots, or when the obstacle does not allow for the convergence to this final configuration. This is exemplified in this experiment by the spin the robot does for the TEB tracking so that it can converge

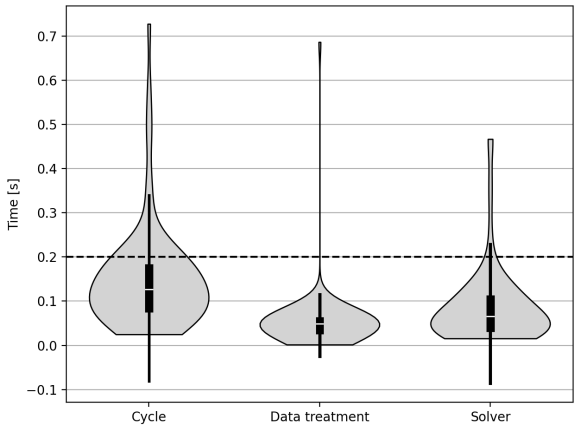
Figure 5.8: Representation of the experiment setup for the overtake study.



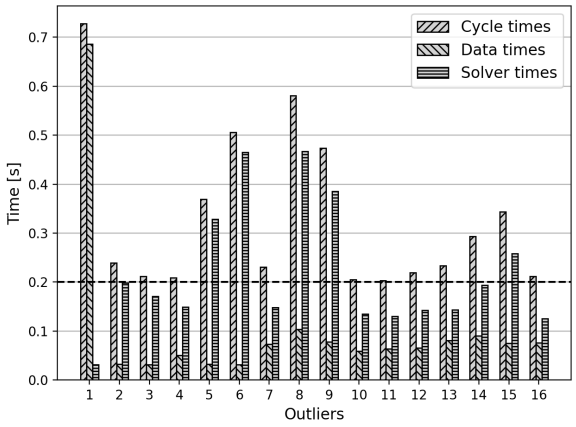
(a) Representation of reference, controller's and obstacle's trajectory.



(b) Representation of reference, TEB's and obstacle's path.



(c) Computational efficiency violin plots.



(d) Computational efficiency outliers.

Figure 5.9: Results for overtaking a dynamic obstacle moving along the robot's way.

back to the position defined as the target for the reference.

The developed controller, unlike TEB, does trajectory, rather than path, tracking. Because of this, i.e., following a time-parameterized reference, there are no problems when it is not possible to converge back to the reference trajectory. The robot's behaviour will simply be to follow the reference as closely as possible, while maintaining a distance greater than the safety distance from the obstacles in the environment. When the trajectory reaches its duration, the tracking ends, wherever the robot may be.

Regarding the overtake maneuver, it is possible to say that the performance of the controller is comparable to that of the baseline TEB regarding the avoiding action. Nevertheless, the controller has clear advantages regarding real-life situations due to its trajectory tracking capabilities. The fact that TEB in not flexible, in that it has to abide by the defined final configuration, eventually leads to problems in changing real-world conditions.

Finally, as expected, the computational efficiency data is in accordance with what has been seen previously throughout this work. The upper whisker of the cycle times exceeds the defined threshold value of 0.2 seconds, defined by the controller's operating frequency, though the interquartile range is within said threshold. Additionally, the majority of the outliers, defined as points at which there is an

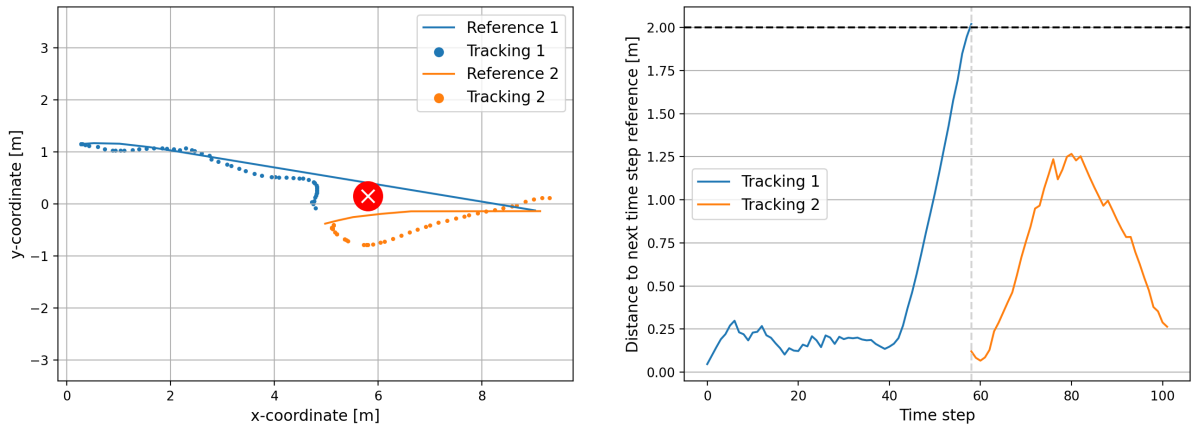
excess of the cycle time over the computational threshold, are due to an excess in solver time due to the deviation from reference and the larger number of constraints to satisfy.

5.6 Trajectory replanning

In order to test for the trajectory replanning capabilities of the controller, an obstacle was set in the way of the robot's trajectory. The distance for replanning was set as a value low enough, so that it would trigger this replanning when the robot is performing its avoiding action. In this case, the value $d_{\text{replan}} = 2\text{m}$ was chosen. The remaining values for the experiment's identifying criteria were the same as the ones used in Section 5.3.

In Figure 5.10 it is possible to see both the robot's avoiding action and consequent deviation from its original trajectory and the recalculation of said reference. Note that the representation of the obstacle is not to scale and is merely indicative of its position regarding the reference trajectories. Moreover, it is important to state that the Euclidean distance used in this section is not alike the one used as a metric for controller performance throughout this work. For the latter, a comparison is made between the robot's position at a given time step h and the reference at the same time step. For the metrics in this section, the robot's position at time step h is compared against the reference at time step $h + 1$, since the rationale behind this replanning feature is that the robot is defined as being unable to reach its next reference point. This inability is defined by the d_{replan} variable.

It is evident that the controller is efficient in its replanning capabilities. As soon as the considered distance exceeds the defined threshold of $d_{\text{replan}} = 2\text{m}$, the tracking of the reference trajectory is aborted and another reference trajectory is calculated from the robot's current position to the goal configuration. The algorithm is successful in maintaining the same goal configuration when replanning. The short distance between the final position of the *Tracking 1* and the initial position of *Reference 2* is due to a safety feature of the robot. This feature decelerates the robot at a constant rate rather than give it



(a) References and tracking for the replanning study with representation of obstacle (red dot with cross) not to scale.

(b) Euclidean distance to next time step reference for the different trajectories in the replanning study.

Figure 5.10: Trajectory replanning of the controller when at a distance greater than d_{replan} from the reference trajectory.

commands to stop immediately, thus preserving the safe storage of whatever the robot may be carrying. It is also worth noting that the trajectory generation algorithm does not account for the presence of the obstacle when it is called upon for the replanning of the trajectory.

5.7 Discussion of the results

Addressing the results in this chapter as a whole, the developed controller was shown to have comparable obstacle avoidance capabilities to the baseline TEB for static obstacles and obstacles that cross the robot's way perpendicularly. Further, it was found to outperform the baseline when overtaking an obstacle going the same way as the robot.

The developed controller was shown to be able to perform successful avoiding action whilst being agnostic to the obstacle's shape and dimension. However, a limitation was found regarding obstacles that have a varying cross-section, especially ones whose cross-section area decreases with height, due to the positioning of the laser onboard the robot.

The work was also evidenced to have no requirements for the *a priori* knowledge of a moving obstacle's trajectory, thus confirming that the recalculation of the obstacle constraints at every controller cycle allows to account for the movement of said obstacles. Finally, the controller was shown to be successful in integrating a replanning capability into its framework, aborting its tracking instantaneously when the distance from the reference is greater than the distance that triggers said replanning d_{replan}.

Finally, regarding computational efficiency, the voxel cell size was proven to be a great driver in computational expense of the problem. Further analysis of the outliers confirms that the chosen solver is not the most appropriate to dealing with obstacle avoidance scenarios, as it is evidenced as the most frequent cause of excess in cycle times beyond the maximum allowed threshold. Though the space constraints did not allow for a use of a voxel cell size that promoted computational efficiency throughout the conducted experiments, the studies show that it is possible to achieve said efficiency if larger voxel cell sizes are considered.

Chapter 6

Conclusions

In this chapter, all the work developed in this thesis is summarized and main conclusions are drawn. The main achievements of this work are also shown. Additionally, opportunities for further development are highlighted.

6.1 Achievements

The controller developed in this work proves effective in both trajectory tracking and avoidance of static and dynamic obstacles. The studies conducted show that the controller is able to accurately track a trajectory that features all components of movement. Moreover, it can deviate from differently shaped static obstacles and moving obstacles both crossing the robot's way perpendicularly and in the same direction of its movement. Regarding the comparison to the TEB, the developed controller outperforms the baseline regarding tracking and has comparable capabilities regarding obstacle avoidance.

When testing in a real-world environment, it was proven that the vehicle model used has some error regarding the actual behaviour of the robot. The fact that trajectory tracking is still effective despite this modelling error further shows how powerful the NMPC framework is as a control technique. Further, the controller was shown to have tunable parameters that influence its capabilities, as well as its computational efficiency, and shown to converge back to the trajectory when starting at an initial point different than that of its reference trajectory. This convergence, again, points to the robustness of the developed controller to unexpected scenarios. The need for the use of a maximum velocity in tracking greater than the maximum velocity at which the trajectory is generated was also confirmed.

For obstacle avoidance, the controller was shown to be effective in deviating from both static and moving obstacles and the used voxel cell size shown to have a direct impact on the computational efficiency of the problem. By testing the avoidance of different static obstacles, the controller was shown to be agnostic to the obstacle's shape and dimensions, thus being fully equipped to dealing with unpredictability. When testing for moving obstacles, the controller proves successful in deviating from an obstacle crossing its way and overtaking an obstacle going the same direction as the robot.

The TEB baseline proves to be smoother in deviating from the static and crossing obstacles, how-

ever, loses in performance against the controller for the overtaking maneuver. The assumption that recalculating the obstacle avoidance constraints at every controller cycle allows to account for the movement of a dynamic obstacle is proven to be valid. Moreover, the trajectory replanning capabilities of the controller were shown to be accurate and effective in recalculating a reference trajectory for conditions considered to be unsafe for the robot's operation - in this case, being at a distance greater than the distance that triggers replanning from the reference.

Regarding computational efficiency, due to the online implementation nature of an NMPC controller, the cycle of the controller should run below the computational threshold defined by said controller's operating frequency. This efficiency is proven for the trajectory tracking but not for the experiments conducted for obstacle avoidance. However, it is shown to be possible to remain within the computational threshold by increasing the voxel cell size. This shows both the computational expense for the solver of having to calculate for the obstacle avoidance constraints and was only not possible to use due to a limitation in available space for the conducted experiments.

During the testing for both the trajectory tracking and obstacle avoidance capabilities of the controller, the assumptions made upon development were confirmed to be valid, namely regarding the equivalence between the constant acceleration and constant velocity models, the chosen solver type, the accuracy of the initial guess provided and, finally, the increased computational efficiency of using a voxel grid. The latter is confirmed by the fact that an increase in the voxel cell size, and thus a greater bundling of the obstacle points, promotes computational efficiency.

When comparing to the related work, the controller developed in this thesis is both able to be agnostic to the obstacle's trajectory and dimensions and is tested in real-world conditions. The experiments conducted in the real environment, as well as the novel integration of obstacle avoidance into the NMPC controller framework sets this thesis apart from the remaining work in this field.

6.2 Future work

Addressing future work should account for both the future of the AGiLE project and the further development of the controller for general use. As was mentioned, one of the greatest features of the work in this thesis is that the developed controller is both of a tunable nature and adaptable to different environments, scenarios, obstacles and vehicles.

First, addressing the future of the AGiLE project, and the implementation of this tool into the project's logistical flow, it is imperative that extensive testing is done in the real conditions of the Volkswagen Autoeuropa factory, for which the project is being developed. This testing, as well as exposure to different conditions will allow for a better tuning of the controller in order to be the best fit for the real needs of the project. Also considering the full scope of the project, as tracking at lower speed conditions was shown to be superior to tracking at higher speeds, the possibility of generating the trajectories at a lower nominal velocity v_{nom} is worth noting.

Additionally, and as a future work of this project, both the AMCL positioning estimate error and vehicle model error should be corrected. Both these errors introduce oscillations into the controller's tracking and

damage its obstacle avoidance capabilities. To correct for the AMCL positioning estimate, the odometry mismatch should be accounted for by studying, understanding and correcting for the readings from the motors of the robot. In this case, the motors fail to account for either negative and positive velocities and, as such, should be corrected to consider the signed, rather than absolute, velocity values. The vehicle model error requires solely that a study is made on its behaviour for real-world conditions, in order to correctly model the variables that influence its movement in the (x, y) plane.

Regarding the further development of the controller, as is the real-time implementation nature of the NMPC technique, the improvement of computational efficiency should be addressed. This is particularly noticeable for the obstacle avoidance cases, in which the solver is responsible for the majority of the outliers in the cycle times. The small number of outliers due to the time for the data treatment can also be tackled, when considering the same techniques that are subsequently proposed as future work for this improvement of solver times.

For the improvement of the solver times for the obstacle avoidance cases, not considering the obvious solution of simply increasing the processing power of the computer in which the controller is run, a few options could be studied as the future development of this work. First, in order to reduce the computational burden of calculating for such a high number of constraints, and given that most obstacles points are the barriers of the pre-defined navigable area, these barriers could be described as geometrical lines instead of a set of points. This would transform a set of distance-to-point obstacle avoidance constraints into a single distance-to-line linear constraint, which could have significant impact on the computational efficiency of the problem. Moreover, alongside this, the development of an adaptive, i.e. varying, voxel grid could be studied. This varying grid would have a smaller cell sizes closer to the robot, for precision in avoidance, and larger cell sizes farther from the robot, for increased computational efficiency when calculating for the constraints over the length of the prediction horizon.

For the future of the use case considered, the study and use of a different solver should not be discarded. The choice of the Interior Point Optimizer (IPOPT) was done with a support on literature that featured testing for the same project for which the controller was developed. However, this literature fails to account for obstacle avoidance scenarios, where the solver was proven to have a poorer performance, as shown by the study of the computational efficiency outliers for these results.

Finally, given that the controller is developed to be independent of the type of vehicle considered, studies could be conducted for different types of vehicle dynamics, in order to experimentally prove that the controller has this adaptability and flexibility in dealing with different models or dynamics.

Bibliography

[1] I. Silva. Decentralized trajectory optimization for a fleet of industrial mobile robots. Master's thesis, Instituto Superior Técnico, 2023.

[2] C. Kao et al. Productivity improvement: Efficiency approach vs effectiveness approach. *Omega*, 23(2):197–204, 1995. doi: 10.1016/0305-0483(94)00058-I.

[3] Organisation for Economic Co-operation and Development. Glossary of Key Terms in Regulatory Impact Analysis, ND. URL `https://www.oecd.org/regreform/sectors/2376087.pdf`. Accessed: March 21, 2023.

[4] Council of Supply Chain Management Professionals (CSCMP). Supply Chain Management Definitions and Glossary of Terms, 2013. URL `https://cscmp.org/CSCMP/Educate/SCM_Definitions_and_Glossary_of_Terms.aspx`. Accessed: March 21, 2023.

[5] M. ten Hompel et al. *Material Flow Systems*. Springer, 3rd edition, 2007. ISBN 9783540732365.

[6] H. Shakhatreh et al. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, 7:48572–48634, 2019. doi: 10.1109/ACCESS.2019.2909530.

[7] W. Jin et al. Research on Application and Deployment of UAV in Emergency Response. In *2020 IEEE 10th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, 2020. doi: 10.1109/ICEIEC49280.2020.9152338.

[8] P. Pina and G. Vieira. UAVs for science in Antarctica. *Remote Sensing*, 14(7):1610, 2022. doi: 10.3390/rs14071610.

[9] D. Giordan et al. The use of unmanned aerial vehicles (UAVs) for engineering geology applications. *Bulletin of Engineering Geology and the Environment*, 79:3437–3481, 2020. doi: 10.1007/s10064-020-01766-2.

[10] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012. doi: 10.1002/rob.20414.

[11] M. Rokonuzzaman et al. Review and performance evaluation of path tracking controllers of autonomous vehicles. *IET Intelligent Transport Systems*, 15:646–670, 5 2021. doi: 10.1049/itr2.12051.

[12] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon University, 1992.

[13] S. F. Campbell. *Steering control of an autonomous ground vehicle with application to the DARPA urban challenge*. PhD thesis, Massachusetts Institute of Technology, 2007.

[14] W.-J. Wang et al. The improved pure pursuit algorithm for autonomous driving advanced system. In *2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA)*, pages 33–38, 2017. doi: 10.1109/IWCIA.2017.8203557.

[15] A. AbdElmoniem et al. A path-tracking algorithm using predictive Stanley lateral controller. *International Journal of Advanced Robotic Systems*, 17(6), 2020. doi: 10.1177/1729881420974852.

[16] J.-P. Laumond, editor. *Robot Motion Planning and Control*. Springer, 1st edition, 1998. ISBN 9783540762195. doi: 10.1007/BFb0036069.

[17] M. Morari et al. Model predictive control: Theory and practice. *IFAC Proceedings Volumes*, 21(4): 1–12, 1988. doi: 10.1016/B978-0-08-035735-5.50006-1.

[18] M. L. Darby and M. Nikolaou. MPC: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342, 2012. doi: 10.1016/j.conengprac.2011.12.004.

[19] F. Yakub and Y. Mori. Effects of roll dynamics for car stability control by laguerre functions. In *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013*, pages 330–335, 2013. doi: 10.1109/ICMA.2013.6617940.

[20] E. Vinodh Kumar and J. Jerome. Robust LQR Controller Design for Stabilizing and Trajectory Tracking of Inverted Pendulum. *Procedia Engineering*, 64:169–178, 2013. doi: 10.1016/j.proeng.2013.09.088.

[21] A. Levant. Sliding order and sliding accuracy in sliding mode control. *International Journal of Control*, 58(6):1247–1263, 1993. doi: 10.1080/00207179308923053.

[22] S. V. Drakunov and V. I. Utkin. Sliding mode control in dynamic systems. *International Journal of Control*, 55(4):1029–1037, 1992. doi: 10.1080/00207179208934270.

[23] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5:90–98, 1986.

[24] M. G. Park et al. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In *2001 IEEE International Symposium on Industrial Electronics Proceedings*, volume 3, pages 1530–1535, 2001. doi: 10.1109/ISIE.2001.931933.

[25] J. Sun et al. Collision Avoidance for Cooperative UAVs With Optimized Artificial Potential Field Algorithm. *IEEE Access*, 5:18382–18390, 2017. doi: 10.1109/ACCESS.2017.2746752.

[26] F. Janabi-Sharifi and D. Vinke. Integration of the artificial potential field approach with simulated annealing for robot path planning. In *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pages 536–541, 1993. doi: 10.1109/ISIC.1993.397640.

[27] J. A. Douthwaite et al. Velocity obstacle approaches for multi-agent collision avoidance. *Unmanned Systems*, 7(01):55–64, 2019. doi: 10.1142/S2301385019400065.

[28] J. van den Berg et al. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008. doi: 10.1109/ROBOT.2008.4543489.

[29] Y. Cho et al. Experimental validation of a velocity obstacle based collision avoidance algorithm for unmanned surface vehicles. *IFAC-PapersOnLine*, 52(21):329–334, 2019. doi: 10.1016/j.ifacol.2019.12.328. 12th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2019.

[30] C. Y. Tan et al. Three Dimensional Collision Avoidance for Multi Unmanned Aerial Vehicles Using Velocity Obstacle. *Journal of Intelligent & Robotic Systems*, 97:227–248, 2020. doi: 10.1007/s10846-019-01055-5.

[31] W. Shaobo et al. A collision avoidance decision-making system for autonomous ship based on modified velocity obstacle method. *Ocean Engineering*, 215, 2020. doi: 10.1016/j.oceaneng.2020.107910.

[32] M. Cord and P. Cunningham, editors. *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Springer, 1st edition, 2008. ISBN 9783540751700. doi: 10.1007/978-3-540-75171-7.

[33] M. van Otterlo and M. Wiering, editors. *Reinforcement Learning: State-of-the-Art*. Springer, 1st edition, 2012. ISBN 9783642276446. doi: 10.1007/978-3-642-27645-3.

[34] J. Wu et al. Learning-based fixed-wing UAV reactive maneuver control for obstacle avoidance. *Aerospace Science and Technology*, 126, 2022. doi: 10.1016/j.ast.2022.107623.

[35] P. Bhopale et al. Reinforcement Learning Based Obstacle Avoidance for Autonomous Underwater Vehicle. *Journal of Marine Science and Application*, 18:228–238, 2019. doi: 10.1007/s11804-019-00089-3.

[36] D. Gandhi et al. Learning to Fly by Crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017. doi: 10.48550/arXiv.1704.05588.

[37] C. Rösmann et al. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6, 2012. ISBN 978-3-8007-3418-4.

[38] C. Rösmann et al. Efficient trajectory optimization using a sparse model. In *2013 European Conference on Mobile Robots*, pages 138–143, 2013. doi: 10.1109/ECMR.2013.6698833.

[39] C. Rösmann et al. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017. doi: 10.1016/j.robot.2016.11.007.

[40] C. Rösmann et al. Planning of multiple robot trajectories in distinctive topologies. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015. doi: 10.1109/ECMR.2015.7324179.

[41] C. Rösmann et al. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686, 2017. doi: 10.1109/IROS.2017.8206458.

[42] N. Metsänen. Assessment of local path planners in a indoor and outdoor robot. Master's thesis, Aalto University, 2020.

[43] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control Theory and Algorithms*. Springer, 2nd edition, 2011. ISBN 9783319460246. doi: 10.1007/978-0-85729-501-9.

[44] M. Schwenzer et al. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5-6):1327–1349, 2021. doi: 10.1007/s00170-021-07682-3.

[45] U. Z. A. Hamid et al. Modular design of artificial potential field and nonlinear model predictive control for a vehicle collision avoidance system with move blocking strategy. In *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, volume 232, pages 1353–1373. SAGE Publications Ltd, 9 2018. doi: 10.1177/0954407017729057.

[46] M. A. Abbas et al. Obstacle avoidance in real time with Nonlinear Model Predictive Control of autonomous vehicles. In *Canadian Conference on Electrical and Computer Engineering*. Institute of Electrical and Electronics Engineers Inc., 9 2014. doi: 10.1109/CCECE.2014.6901109.

[47] M. Gaertner et al. Collision-Free MPC for Legged Robots in Static and Dynamic Scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021. doi: 10.1109/ICRA48506.2021.9561326.

[48] J. M. Park et al. Obstacle avoidance of autonomous vehicles based on model predictive control. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 223:1499–1516, 2009. doi: 10.1243/09544070JAUTO1149.

[49] M. Kamel et al. Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. doi: 10.1109/IROS.2017.8202163.

[50] M. Sani et al. Dynamic Obstacles Avoidance Using Nonlinear Model Predictive Control. In *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6, 2021. doi: 10.1109/IECON48115.2021.9589658.

[51] B. Lindqvist et al. Nonlinear MPC for Collision Avoidance and Control of UAVs With Dynamic Obstacles. *IEEE Robotics and Automation Letters*, 5(4):6001–6008, 2020. doi: 10.1109/LRA.2020.3010730.

[52] I. Sánchez et al. Nonlinear Model Predictive Path Following Controller with Obstacle Avoidance. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 102, 5 2021. doi: 10.1007/s10846-021-01373-7.

[53] A. Sathya et al. Embedded nonlinear model predicitve control for obstacle avoidance using PANOC. In *2018 European Control Conference (ECC)*, 2018. doi: 10.23919/ECC.2018.8550253.

[54] L. Stella et al. A Simple and Efficient Algorithm for Nonlinear Model Predictive Control. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 9 2017. doi: 10.48550/arXiv.1709.06487.

[55] Y. Zhou et al. Voxelization modelling based finite element simulation and process parameter optimization for Fused Filament Fabrication. *Materials & Design*, 187, 2020. doi: 10.1016/j.matdes.2019.108409.

[56] M. Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, 2017. doi: 10.1137/16M1062569.

[57] A. Rao. A Survey of Numerical Methods for Optimal Control. *Advances in the Astronautical Sciences*, 135, 2010.

[58] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2nd edition, 2010. ISBN 9780898716887. doi: 10.1137/1.9780898718577.

[59] J. A. E. Andersson et al. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[60] M. Quigley et al. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[61] P. Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

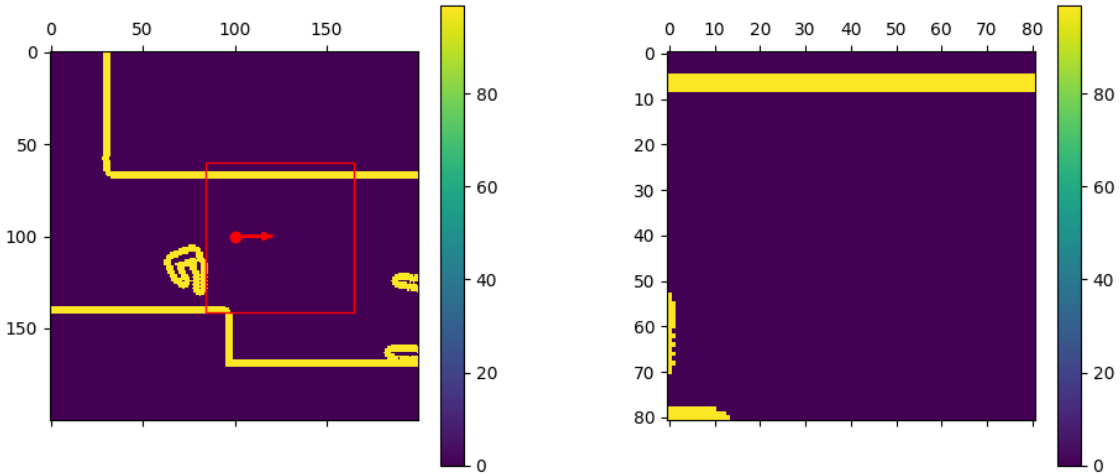
[62] A. Wächter and L. Biegler. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical programming*, 106:25–57, 03 2006. doi: 10.1007/s10107-004-0559-y.

[63] D. Fox et al. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proceedings of the National Conference on Artificial Intelligence*, pages 343–349, 1999.

Appendix A

View-windows for heading angles

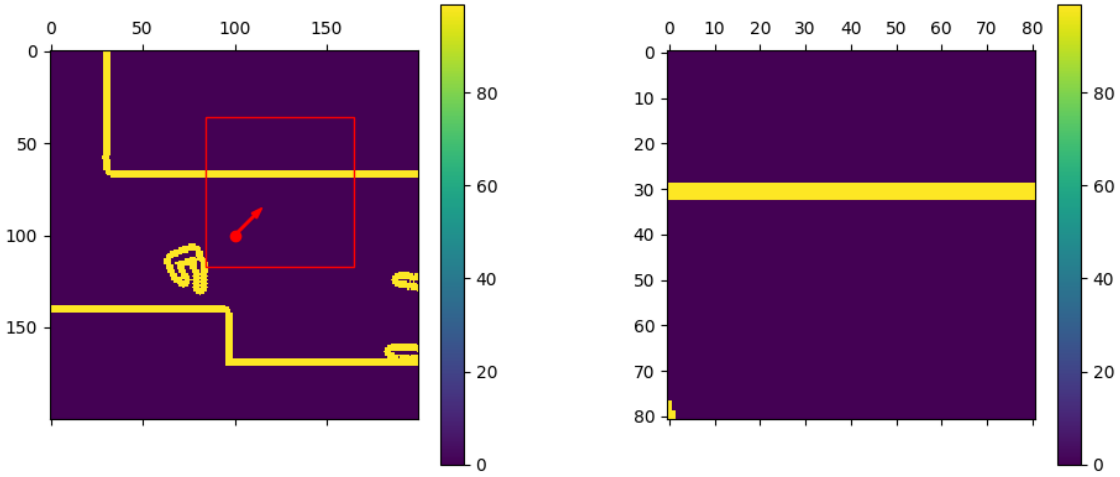
When defining the obstacle avoidance constraints in this work, it is important to understand how the selection of a view-window is made when treating the occupancy map data. According to the robot's heading angle, the robot could be either moving primarily in positive or negative x-direction, positive or negative y-direction or a combination of two of these directions. In the figures presented in this appendix, every heading angle interval's view-windows are shown for a sample full occupancy map. It is important to understand how the view-window accompanies the rotation of the robot, hence why they are shown for every possible scenario. For every figure, the values of the variables used are as follows: $max_{\text{range}} = 4\text{m}$, $data_{\text{min}} = 0.2$ and $data_{\text{max}} = 0.8$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

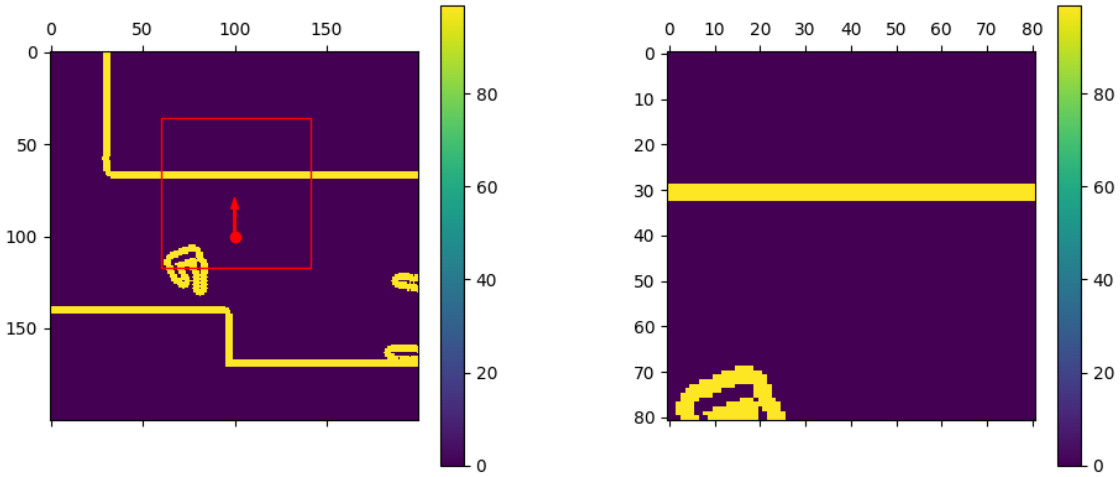
Figure A.1: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [-\frac{\pi}{8}, \frac{\pi}{8}[\text{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

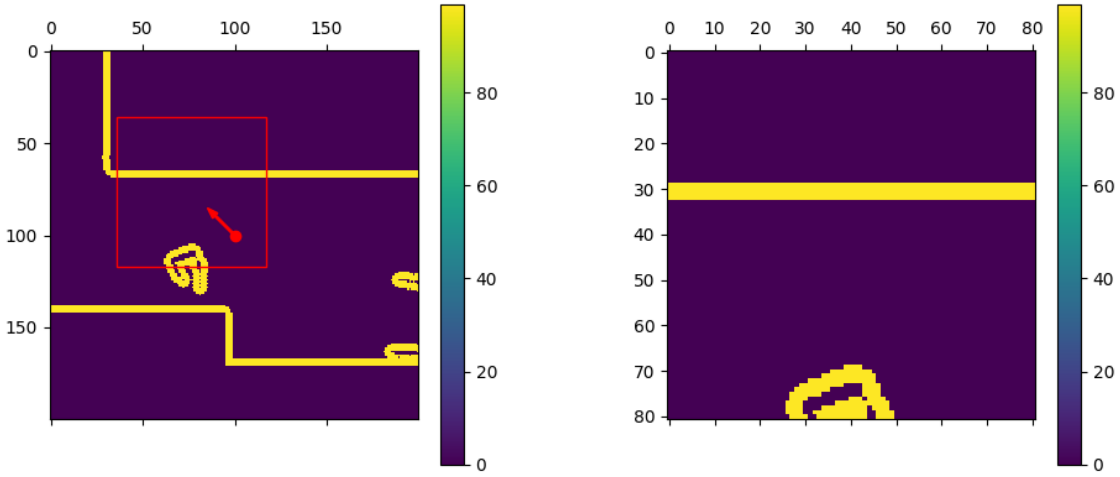
Figure A.2: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{\pi}{8}, \frac{3\pi}{8}[\text{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

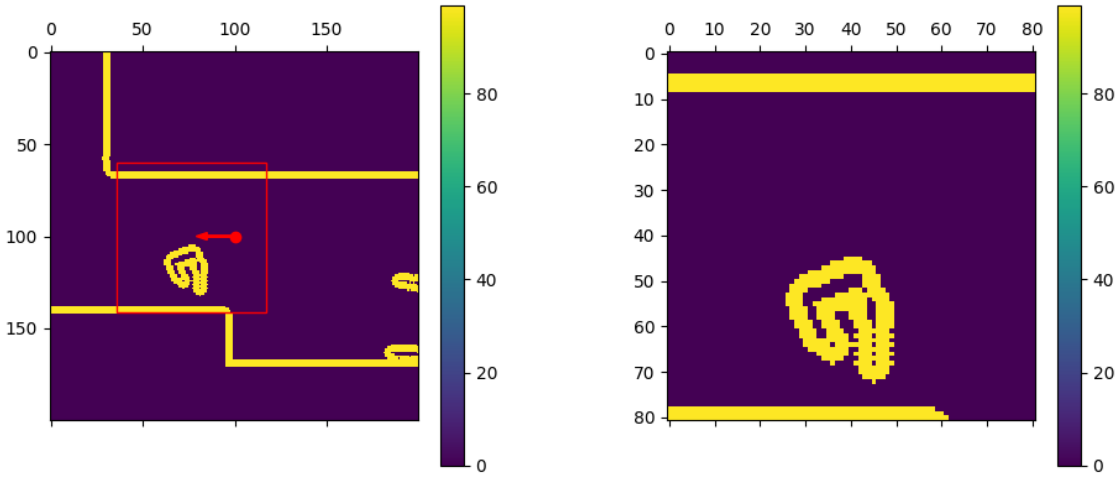
Figure A.3: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{3\pi}{8}, \frac{5\pi}{8}[\text{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

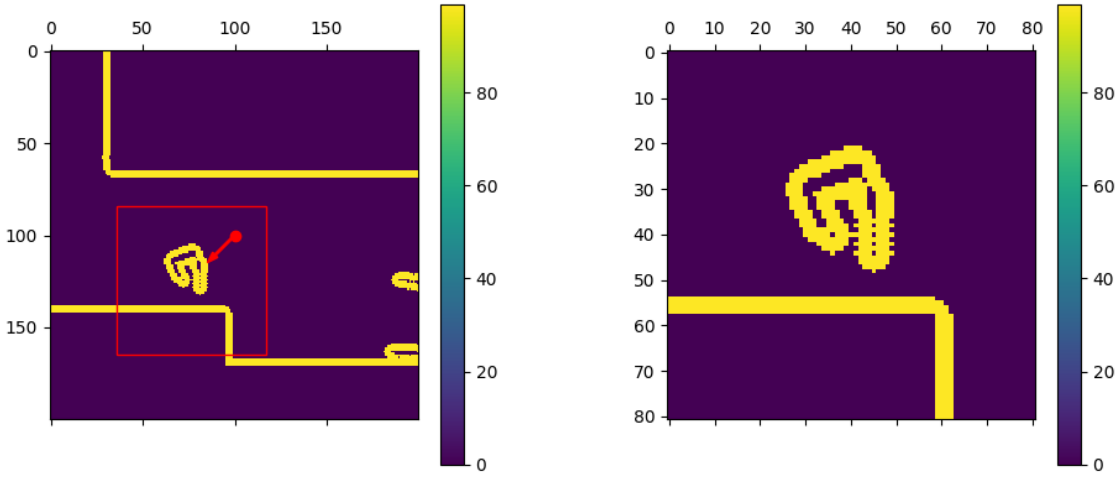
Figure A.4: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{5\pi}{8}, \frac{7\pi}{8}[\,\mathrm{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

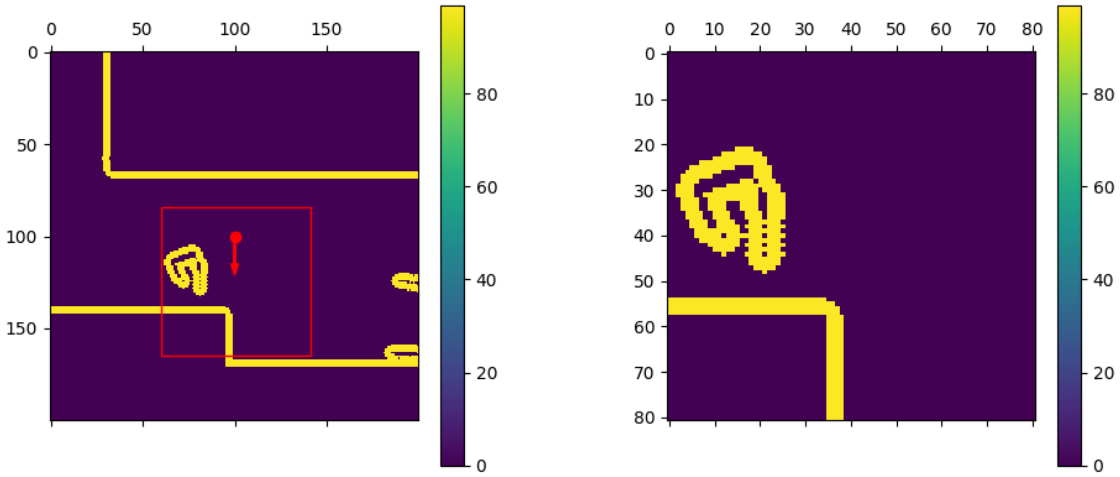
Figure A.5: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{7\pi}{8}, \frac{9\pi}{8}[\,\mathrm{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

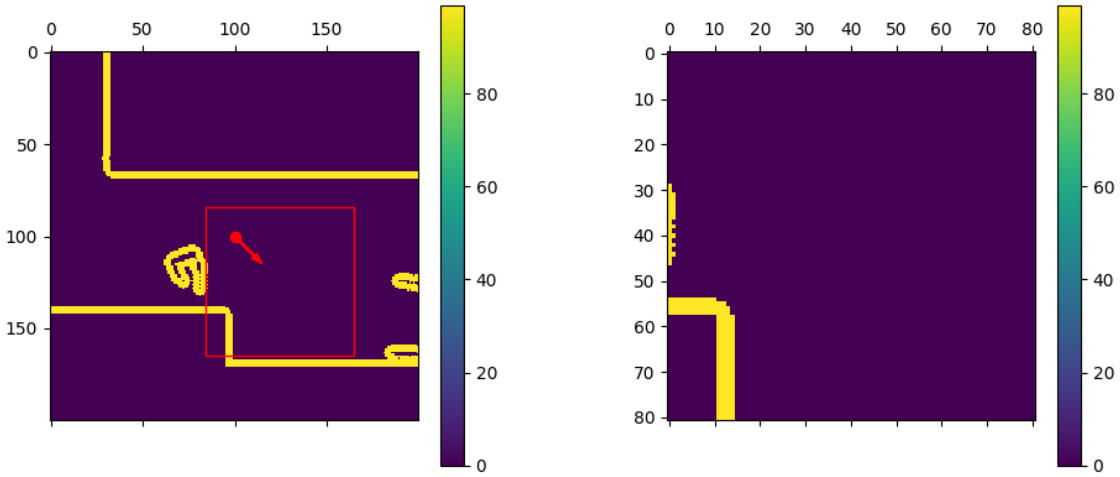
Figure A.6: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{9\pi}{8}, \frac{11\pi}{8}[\text{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

Figure A.7: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{11\pi}{8}, \frac{13\pi}{8}[\text{rad}$.



(a) Full occupancy map with the selection of the view-window.

(b) View-window considered.

Figure A.8: Robot's view-window of the occupancy map for a heading angle interval of $\theta \in [\frac{13\pi}{8}, \frac{15\pi}{8}[\,\mathrm{rad}$.

Appendix B

Parameters for experiments

In this appendix, the internal parameters are shown for both the baseline TEB and the AMCL positioning estimate. Note that these values are not the pre-configured parameters for each of the packages. TEB has catastrophic results when the pre-configured values are used, as it is unable to track a given reference path. The values presented were subject to tuning experiments over the course of the duration of the AGiLE project, for which this work is ultimately developed. These are the values that optimize performance for both TEB and AMCL at the start of this thesis and for the environment considered.

In Table B.1, there are the values for the internal parameters for the Adaptive Monte Carlo Localization (AMCL) positioning system used for experimenting in a real-world environment. The meaning of each of the parameters is outside the scope of this work, as was mentioned previously.

Table B.1: AMCL parameters used for experiments in a real-world scenario.

Parameter	**Value**
min_particles	600
max_particles	800
kld_err	0.01
kld_z	0.99
update_min_d	0.05
update_min_a	0.05
resample_interval	2
transform_tolerance	0.1
recovery_alpha_slow	0.0
recovery_alpha_fast	0.0
do_beamskip	False
beam_skip_distance	0.5
beam_skip_threshold	0.3
tf_broadcast	True

Parameter	Value
gui_publish_rate	10.0
save_pose_rate	0.5
use_map_topic	True
first_map_only	False
laser_min_range	-1.0
laser_max_range	-1.0
laser_max_beams	100
laser_z_hit	0.95
laser_z_short	0.1
laser_z_max	0.05
laser_z_rand	0.05
laser_sigma_hit	0.2
laser_lambda_short	0.1
laser_likelihood_max_dist	2.0
laser_model_type	beam_const (beam)
odom_model_type	diff_corrected_const (diff-corrected)
odom_alpha1	0.1
odom_alpha2	0.05
odom_alpha3	0.2
odom_alpha4	0.1
odom_alpha5	0.1
odom_frame_id	odom
base_frame_id	base_link
global_frame_id	map
restore_defaults	False

In Table B.2, there are the values for the internal parameters for the Timed-Elastic-Band (TEB) local planner used as a baseline for experimenting in a real-world environment. The meaning of each of the parameters is outside the scope of this work, as was mentioned previously.

Table B.2: TEB parameters used for experiments in real-world scenario.

Class	Parameter	Value
Goal tolerance	xy_goal_tolerance	0.1
	yaw_goal_tolerance	0.2
	free_goal_vel	False

Class	Parameter	Value
HCPlanning	enable_multithreading	True
	max_number_classes	2
	selection_cost_hysteresis	1.0
	selection_prefer_initial_plan	0.95
	selection_obst_cost_scale	1.0
	selection_viapoint_cost_scale	1.0
	selection_alternative_time_cost	True
	switching_blocking_period	0.0
	roadmap_graph_no_samples	15
	roadmap_graph_area_width	5.0
	roadmap_graph_area_length_scale	1.0
	h_signature_prescaler	0.5
	h_siganture_threshold	0.1
	obstacle_heading_threshold	0.45
	viapoints_all_candidates	False
	visualize_hc_graph	False
Obstacles	min_obstacle_dist	0.1
	inflation_dist	0.3
	dynamic_obstacle_inflation_dist	0.2
	include_dynamic_obstacles	True
	include_costmap_obstacles	False
	legacy_obstacle_association	False
	obstacle_association_force_inclusion_factor	1.5
	obstacles_association_cutoff_factor	5.0
	costmap_obstacles_behind_robot_dist	1.0
	obstacle_poses_affected	30
Optimization	no_inner_iterations	5
	no_outer_iterations	6
	optimization_activate	True
	optimization_verbose	False
	penalty_epsilon	0.1
	weight_max_vel_x	1.0
	weight_max_vel_y	2.0
	weight_max_vel_theta	1.0
	weight_acc_lim_x	1.0
	weight_acc_lim_y	0.0
	weight_acc_lim_theta	1.0

Class	Parameter	Value
Optimization	weight_kinematics_nh	10000.0
	weight_kinematics_forward_drive	1000.0
	weight_kinematics_turning_radius	1.0
	weight_optimaltime	0.1
	weight_shortest_path	0.1
	weight_obstacle	60.0
	weight_inflation	0.5
	weight_dynamic_obstacle	60.0
	weight_dynamic_obstacle_inflation	0.5
	weight_viapoint	2.0
	weight_adapt_factor	2.0
	obstacle_cost_exponent	1.0
Recovery	shrink_horizon_backup	True
	oscillation_recovery	True
Robot	max_vel_x	0.7
	max_vel_x_backwards	0.2
	max_vel_theta	0.9
	acc_lim_x	0.4
	acc_lim_theta	0.4
	is_footprint_dynamic	False
	min_turning_radius	0.0
	wheelbase	1.0
	cmd_angle_instead_rotvel	False
	max_vel_y	0.0
	acc_lim_y	0.5
Trajectory	teb_autosize	True
	dt_ref	0.4059
	dt_hysteresis	0.1
	global_plan_overwrite_orientation	True
	allow_init_with_backwards_motion	False
	max_global_plan_lookahead_dist	3.5
	force_reinit_new_goal_dist	1.0
	force_reinit_new_goal_angular	0.78
	feasibility_check_no_poses	5
	exact_arc_length	False
	publish_feedback	False

Class	Parameter	Value
Trajectory	visualize_with_time_as_z_axis_scale	0.2
ViaPoints	global_plan_viapoint_sep	-0.1
	via_points_ordered	False