

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Trajectory Control and Collision Avoidance for a Mobile Robot

Jorge Diogo Ferreira Ribeiro



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Dr. António Paulo Gomes Mendes Moreira

Second Supervisor: Dr. Héber Miguel Plácido Sobreira

July 22, 2024



# Sumário

A integração de robôs móveis autônomos em diversos contextos industriais e de serviços tem vindo a aumentar, o que tem levado a uma procura significativa por um melhor desempenho e controlo. Estes robôs enfrentam desafios para realizar manobras precisas a altas velocidades em ambientes dinâmicos, repletos de obstáculos móveis e estacionários. Assim, os controladores convencionais são muitas vezes insuficientes nestes cenários, exigindo soluções inovadoras.

Esta tese visa responder a estes desafios através do desenvolvimento de um módulo de planeamento e seguimento de trajetórias, para robôs móveis terrestres com rodas, que incorpore capacidades de desvio de obstáculos e prevenção de colisões. O objetivo principal é promover o desenvolvimento de controladores que possibilitem a navegação segura, rápida e precisa dos robôs em ambientes dinâmicos.

Em primeiro lugar, uma revisão da literatura é realizada para examinar o trabalho existente sobre controladores de robôs móveis, definição de trajetórias e estratégias de desvio de obstáculos. Esta revisão é complementada com informações de carácter geral sobre estes tópicos. Em seguida, relativamente ao seguimento da trajetória, é implementado um controlo preditivo de modelos não lineares para controlar um robô omnidirecional. Quanto à prevenção de colisões, são propostos dois métodos: um que utiliza zonas de segurança virtuais para parar o robô e outro que utiliza a geração online de splines para evitar obstáculos.

Os resultados do seguimento de trajetórias são obtidos a partir de simulações e experiências com robôs reais. O controlador é implementado utilizando a plataforma ROS (*Robot Operating System*), amplamente adoptada, com simulações realizadas no simulador Gazebo. Além disso, são também apresentadas comparações entre diferentes controladores do estado da arte. Para a prevenção de colisões, os resultados são obtidos a partir de simulações efectuadas em Matlab. Além disso, o algoritmo de desvio de obstáculos é testado em vários cenários, com diferentes configurações e formas de obstáculos.

De um modo geral, o controlador de seguimento de trajetórias proposto, com base em simulações e experiências reais, melhora a precisão tanto em posição como em orientação, sem aumentar significativamente o tempo de processamento. Quanto à prevenção de colisões, o algoritmo de desvio de obstáculos gera com sucesso trajetórias suaves em várias configurações de obstáculos, garantindo uma navegação segura em torno destes.

**Palavras-chave:** robô omnidirecional, controlador preditivo de modelos não lineares, splines, desvio de obstáculos, controlo de trajetórias

# Abstract

The integration of Autonomous Mobile Robots (AMRs) in various industrial and service settings has increased, driving significant demand for enhanced performance and control capabilities. These robots face challenges in achieving high-speed navigation and precise maneuvering in dynamic environments filled with both mobile and stationary obstacles. Conventional controllers often fall short in these scenarios, necessitating innovative solutions.

This thesis aims to address these challenges by developing a path planning and tracking module for wheeled ground mobile robots that incorporate obstacle avoidance capabilities. The primary objective is to advance the development of controllers that enable robots to navigate safely, quickly, and accurately in dynamic environments.

First, a comprehensive literature review is performed to examine existing work on mobile robot controllers, trajectory definition, and obstacle avoidance strategies. This review is supplemented with essential background information on these topics. Next, for trajectory tracking, a Nonlinear Model Predictive Control (NMPC) is implemented to control an omnidirectional robot. As for collision avoidance, two methods are proposed: one using virtual security zones to halt the robot, and another using online spline generation for obstacle avoidance.

The results for trajectory tracking are derived from both simulations and experiments with real robots. The controller is implemented within the widely adopted Robot Operating System (ROS) framework, with simulations conducted in the Gazebo simulator. Moreover, comparisons between different state-of-the-art controllers are also presented. For collision avoidance, results are obtained from simulations performed in Matlab. The obstacle avoidance algorithm is tested against various obstacle configurations and shapes.

Overall, the proposed trajectory tracking controller improves precision in both position and orientation without significantly increasing processing time, based on simulations and real-world experiments. As for collision avoidance, the obstacle avoidance algorithm successfully generates smooth trajectories across several obstacle configurations, ensuring safe navigation around obstacles.

**Keywords:** omnidirectional robot, Nonlinear Model Predictive Controller (NMPC), splines, obstacle avoidance, trajectory tracking

# Sustainable Development Goals (SDGs)

Integrating Autonomous Mobile Robots (AMRs) in industrial and service environments is a technological advancement and a significant contributor to the United Nations Sustainable Development Goals (SDGs). This chapter explores the contributions of the research conducted in my master's thesis to specific SDGs, highlighting the potential impact on economic growth, industrial innovation, and global partnerships.

**Table 1:** Contributions to SDGs and performance indicators

| SDG | Target | Contribution                      | Performance Indicators and Metrics                             |
|-----|--------|-----------------------------------|----------------------------------------------------------------|
| 8   | 8.2    | Enhancing Industrial Efficiency   | Productivity Increase;<br>Reduction in Downtime                |
|     | 8.3    | Innovation in Robotics            | Task Completion Rate;<br>Number of Industry Applications       |
| 9   | 9.4    | Sustainable Industrial Automation | Energy Efficiency;<br>Accuracy and Safety Metrics              |
| 17  | 17.6   | Collaborative Development         | Research Publications and Citations;<br>Collaborative Projects |

## SDG 8: Decent Work and Economic Growth

- **Enhancing Industrial Efficiency (Target 8.2):** The development of advanced control systems for AMRs enhances productivity in industrial settings by optimizing navigation and task execution. A report by Grand View Research [1] indicates that the global autonomous mobile robots market was valued at USD 2.2 billion in 2020 and is expected to grow at a Compound Annual Growth Rate (CAGR) of 20.3% from 2021 to 2028. Furthermore, a McKinsey report [2] indicates that using AMRs in warehouses can result in a productivity increase of up to 50%. Therefore, measuring the increase in tasks completed per hour by AMRs and the reduced downtime due to improved navigation and obstacle avoidance can be used as metrics.
- **Innovation in Robotics (Target 8.3):** Introducing innovative trajectory tracking and obstacle avoidance strategies promotes technological advancement in the robotics sector, driving economic growth. Target 8.3 promotes development-oriented policies that support productive activities, decent job creation, entrepreneurship, creativity, and innovation. Consequently, metrics involve assessing the task completion rate and the number of different industry applications for AMRs.

## SDG 9: Industry, Innovation, and Infrastructure

- **Sustainable Industrial Automation (Target 9.4):** The algorithms of this dissertation for AMRs contribute to the advancement of sustainable industrial automation by enhancing the efficiency and safety of robotic systems. A review on energy efficiency in autonomous mobile robots indicates that locomotion contributes over 50% of a mobile robot's total energy consumption, emphasizing the need for optimized control methods [3]. Consequently, the reduction in energy consumption of AMRs, the improvement in navigation accuracy, and the reduction in collision incidents in dynamic industrial environments can be employed to assess the impact of this contribution.

## SDG 17: Partnerships for the Goals

- **Collaborative Innovation (Target 17.6):** The integration of AMRs in dynamic environments is a topic of growing interest among academic, industrial, and technological partners who are engaged in collaborative research to advance this field. Target 17.6 aims to enhance international cooperation by supporting national plans to implement all the SDGs through knowledge sharing and collaboration. Therefore, direct contributions related to this thesis include the publication of the master's thesis and a conference paper. These publications contribute to the academic community and facilitate further research and development. Furthermore, additional metrics may be employed to assess the research's impact, including tracking publications' citations and measuring the number of joint projects or partnerships formed as a result of the research findings.

## Conclusion

The research presented in this thesis aligns with the United Nations Sustainable Development Goals by making possible economic growth, promoting industrial innovation, and encouraging global partnerships. By optimizing the performance and control capabilities of autonomous mobile robots, this work contributes to creating more efficient, safe, and resilient industrial environments. The contributions to SDGs 8, 9, and 17 demonstrate the transformative potential of advanced robotics technologies in achieving sustainable development.

# Acknowledgments

First and foremost, I extend my deepest gratitude to my supervisors, Professor Doctor António Paulo Moreira and Doctor Héber Miguel Sobreira, for their invaluable collaboration, assistance, patience, and support throughout the semester. Their guidance was essential for the successful completion of this dissertation. I have learned an immense amount about mobile robotics, particularly in the areas of control and collision avoidance, thanks to their mentorship.

Secondly, I would like to express my special thanks to Ricardo B. Sousa, a researcher at INESC TEC. His assistance over the past year has been invaluable, and I am especially grateful for his constant availability to answer my questions.

I am also deeply thankful to the Centre for Robotics in Industry and Intelligent Systems (CRIIS) at INESC TEC – Institute for Systems and Computer Engineering, Technology and Science, for their support and for providing all the resources necessary for this dissertation.

Additionally, I would like to extend my heartfelt thanks to all my friends, both from my personal and academic life, for their encouragement, companionship, and for always being there to offer motivation and support.

Lastly, my deepest thanks go to my family: my sister Cristiana Andreia, and my parents Joaquim and Maria Manuela. Their unwavering support, love, and encouragement throughout this journey have been fundamental, and I could not have achieved this without them.

Thank you all so much,  
Jorge Diogo Ferreira Ribeiro

*“Strive for perfection in everything you do.  
Take the best that exists and make it better.  
When it does not exist, design it.”*

Sir Henry Royce

# Contents

|          |                                                 |           |
|----------|-------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                             | <b>1</b>  |
| 1.1      | Context . . . . .                               | 1         |
| 1.2      | Motivation . . . . .                            | 2         |
| 1.3      | Goals . . . . .                                 | 3         |
| 1.4      | Contributions . . . . .                         | 3         |
| 1.5      | Structure of the Document . . . . .             | 4         |
| <b>2</b> | <b>State of the Art</b>                         | <b>5</b>  |
| 2.1      | Trajectory Tracking . . . . .                   | 5         |
| 2.1.1    | Reactive controllers . . . . .                  | 6         |
| 2.1.2    | Predictive controllers . . . . .                | 8         |
| 2.2      | Trajectory Definition . . . . .                 | 11        |
| 2.3      | Obstacle Avoidance . . . . .                    | 16        |
| 2.3.1    | Sensors . . . . .                               | 16        |
| 2.3.2    | Algorithms . . . . .                            | 18        |
| 2.4      | Summary . . . . .                               | 19        |
| <b>3</b> | <b>Trajectory Tracking</b>                      | <b>22</b> |
| 3.1      | Controller Architecture . . . . .               | 22        |
| 3.2      | Robot Model . . . . .                           | 23        |
| 3.2.1    | Robot kinematics . . . . .                      | 24        |
| 3.2.2    | Robot dynamics . . . . .                        | 27        |
| 3.3      | Optimization Problem Formulation . . . . .      | 29        |
| 3.3.1    | Optimal control problem . . . . .               | 29        |
| 3.3.2    | Nonlinear programming problem . . . . .         | 29        |
| 3.3.3    | Gradient-based methods . . . . .                | 30        |
| 3.4      | Velocity and Acceleration Constraints . . . . . | 31        |
| 3.4.1    | Velocity scalling . . . . .                     | 31        |
| 3.4.2    | Motion velocity profile . . . . .               | 32        |
| 3.5      | Simulation Results . . . . .                    | 34        |
| 3.5.1    | PD-only controller . . . . .                    | 36        |
| 3.5.2    | Motion velocity profile . . . . .               | 40        |
| 3.6      | Experimental Results . . . . .                  | 41        |
| 3.6.1    | PD-only controller . . . . .                    | 42        |
| 3.6.2    | Feedforward controller . . . . .                | 46        |
| 3.6.3    | MPC with optimization constraints . . . . .     | 47        |
| 3.7      | Discussion . . . . .                            | 50        |

|          |                                                               |           |
|----------|---------------------------------------------------------------|-----------|
| <b>4</b> | <b>Collision Avoidance</b>                                    | <b>51</b> |
| 4.1      | Collision Avoidance Overview . . . . .                        | 51        |
| 4.2      | Virtual Security Zone Method . . . . .                        | 52        |
| 4.3      | Obstacle Avoidance Method . . . . .                           | 55        |
| 4.3.1    | Obstacle detection . . . . .                                  | 55        |
| 4.3.2    | Path generation . . . . .                                     | 58        |
| 4.4      | Simulation Results . . . . .                                  | 61        |
| 4.5      | Discussion . . . . .                                          | 66        |
| <b>5</b> | <b>Conclusion</b>                                             | <b>67</b> |
| 5.1      | Future Work . . . . .                                         | 68        |
|          | <b>References</b>                                             | <b>69</b> |
| <b>A</b> | <b>Gradient Descent Methods</b>                               | <b>74</b> |
| A.1      | Resilient Propagation (Rprop) . . . . .                       | 74        |
| A.1.1    | Rprop with Weight-Backtracking . . . . .                      | 74        |
| A.1.2    | Rprop without Weight-Backtracking . . . . .                   | 76        |
| A.1.3    | Improved Rprop with and without Weight-Backtracking . . . . . | 76        |
| A.2      | Steepest Descent . . . . .                                    | 78        |
| A.3      | Conjugate Gradient Methods . . . . .                          | 79        |
| A.4      | Performance . . . . .                                         | 80        |
| <b>B</b> | <b>Motion Velocity Profiles</b>                               | <b>83</b> |
| B.1      | Trapezoidal . . . . .                                         | 84        |
| B.2      | S-Curve . . . . .                                             | 85        |
| <b>C</b> | <b>Splines</b>                                                | <b>88</b> |
| C.1      | Formulation . . . . .                                         | 88        |
| C.2      | Numeric Algorithms . . . . .                                  | 91        |

# List of Figures

|      |                                                                                                                                                                                      |    |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1  | Feed-Forward PID-based controller for omnidirectional robots with predictive characteristics, proposed by Sousa et al. . . . .                                                       | 8  |
| 2.2  | Schematic illustration of the functioning of an Model Predictive Control . . . . .                                                                                                   | 9  |
| 2.3  | Scheme of the control strategy proposed by Conceição et al. . . . .                                                                                                                  | 10 |
| 2.4  | Classical NMPC approach (left) and Nascimento et al. approach (right) . . . . .                                                                                                      | 11 |
| 2.5  | Bézier spline of 3rd-order . . . . .                                                                                                                                                 | 13 |
| 3.1  | Proposed trajectory tracking controller for omnidirectional mobile robots . . . . .                                                                                                  | 23 |
| 3.2  | Four-wheeled omnidirectional robot . . . . .                                                                                                                                         | 23 |
| 3.3  | Comparison of differential equations discretization with a 0.1 seconds step size . . . . .                                                                                           | 26 |
| 3.4  | Comparison of differential equations discretization with a 0.025 seconds step size . . . . .                                                                                         | 26 |
| 3.5  | First order approximation of the robot velocity dynamics for robot codename “Adam” . . . . .                                                                                         | 28 |
| 3.6  | Reference trajectory discretization . . . . .                                                                                                                                        | 32 |
| 3.7  | Comparison of the motion profiles . . . . .                                                                                                                                          | 33 |
| 3.8  | Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of 0.5 m/s; Time constants of 0.1 s . . . . .            | 37 |
| 3.9  | Evolution of the velocities and errors over time of the proposed NMPC for a pulse trajectory: Simulation; Steady-state linear velocity of 0.5 m/s; Time constants of 0.1 s . . . . . | 37 |
| 3.10 | Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of 1.0 m/s; Time constants of 0.1 s . . . . .            | 38 |
| 3.11 | Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of 0.5 m/s; Time constants of 0.3 s . . . . .            | 39 |
| 3.12 | Evolution of the velocities over time of the proposed NMPC for a pulse trajectory: Simulation; Steady-state linear velocity of 0.5 m/s; Time constants of 0.1 s . . . . .            | 40 |
| 3.13 | Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of 0.5 m/s; Time constants of 0.1 s . . . . .          | 43 |
| 3.14 | Robot codename “Friday” . . . . .                                                                                                                                                    | 44 |
| 3.15 | First order approximation of the robot velocity dynamics for robot codename “Friday” . . . . .                                                                                       | 44 |
| 3.16 | First order approximation of the delayed robot velocity dynamics for robot codename “Adam” . . . . .                                                                                 | 45 |
| 3.17 | Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of 0.5 m/s; Time constants of 0.3 s . . . . .          | 46 |
| 3.18 | Comparison of the proposed controller with a feedforward controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of 0.5 m/s; Time constants of 0.1 s . . . . . | 47 |

|      |                                                                                                                                                                                                                                                    |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.19 | Comparison of the proposed controller with an NMPC controller using the open-source libraries CasADi and Ipopt for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.1\text{ s}$ . . . . . | 49 |
| 4.1  | Graph representation of the navigation paths used by CRIIS robots at INESC TEC                                                                                                                                                                     | 52 |
| 4.2  | Collision Avoidance: Virtual security zone method . . . . .                                                                                                                                                                                        | 53 |
| 4.3  | Obstacle detection: point clouds . . . . .                                                                                                                                                                                                         | 56 |
| 4.4  | Obstacle detection: convex hulls . . . . .                                                                                                                                                                                                         | 57 |
| 4.5  | Obstacle detection: filtered convex hulls . . . . .                                                                                                                                                                                                | 57 |
| 4.6  | Path generation . . . . .                                                                                                                                                                                                                          | 59 |
| 4.7  | Obstacle Avoidance: Setup 1 . . . . .                                                                                                                                                                                                              | 63 |
| 4.8  | Obstacle Avoidance: Setup 2 . . . . .                                                                                                                                                                                                              | 64 |
| 4.9  | Obstacle Avoidance: Setup 3 . . . . .                                                                                                                                                                                                              | 65 |
| A.1  | Comparison of gradient descent methods: Datasets 1, 2, 3, 4, 5, and 6 . . . . .                                                                                                                                                                    | 81 |
| A.2  | Comparison of gradient descent methods in several cycles . . . . .                                                                                                                                                                                 | 82 |
| B.1  | Comparison of the motion profiles . . . . .                                                                                                                                                                                                        | 83 |
| B.2  | Trapezoidal motion velocity profile UML state machine diagram . . . . .                                                                                                                                                                            | 84 |
| B.3  | S-Curve motion velocity profile UML state machine diagram . . . . .                                                                                                                                                                                | 85 |

# List of Tables

|     |                                                                                                                                                                                                                                                            |     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 1   | Contributions to SDGs and performance indicators . . . . .                                                                                                                                                                                                 | iii |
| 2.1 | Key properties of cubic spline families . . . . .                                                                                                                                                                                                          | 15  |
| 3.1 | Pulse trajectory . . . . .                                                                                                                                                                                                                                 | 34  |
| 3.2 | Comparison of the proposed controller with a PD controller for a pulse trajectory:<br>Simulation; Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.1\text{ s}$ . . . .                                                              | 36  |
| 3.3 | Comparison of the proposed controller with a PD controller for a pulse trajectory:<br>Simulation; Steady-state linear velocity of $1.0\text{ m/s}$ ; Time constants of $0.1\text{ s}$ . . . .                                                              | 38  |
| 3.4 | Comparison of the proposed controller with a PD controller for a pulse trajectory:<br>Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.3\text{ s}$ . . . . .                                                                        | 39  |
| 3.5 | Comparison of the proposed controller with a PD controller for a pulse trajectory:<br>Robot “Adam”; Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.1\text{ s}$ . .                                                                | 43  |
| 3.6 | Comparison of the proposed controller with a PD controller for a pulse trajectory:<br>Robot “Adam”; Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.3\text{ s}$ . .                                                                | 45  |
| 3.7 | Comparison of the proposed controller with a feedforward controller for a pulse<br>trajectory: Robot “Adam”; Steady-state linear velocity of $0.5\text{ m/s}$ ; Time constants<br>of $0.1\text{ s}$ . . . . .                                              | 47  |
| 3.8 | Comparison of the proposed controller with an NMPC controller using the open-<br>source libraries CasADi and Ipopt for a pulse trajectory: Robot “Adam”; Steady-<br>state linear velocity of $0.5\text{ m/s}$ ; Time constants of $0.1\text{ s}$ . . . . . | 49  |

# Abbreviations and Symbols

|           |                                                                        |
|-----------|------------------------------------------------------------------------|
| AI        | Artificial Intelligence                                                |
| AMRs      | Autonomous Mobile Robots                                               |
| APF       | Artificial Potential Field                                             |
| CRIIS     | Centre for Robotics in Industry and Intelligent Systems                |
| DoFs      | Degrees of Freedom                                                     |
| FF        | Feed Forward                                                           |
| INESC TEC | Institute for Systems and Computer Engineering, Technology and Science |
| LiDAR     | Light Detection and Ranging                                            |
| MIMO      | Multi-Input Multi-Output                                               |
| MPC       | Model Predictive Control                                               |
| NMPC      | Nonlinear Model Predictive Control                                     |
| NN        | Neural Network                                                         |
| NLP       | Nonlinear Programming                                                  |
| OCP       | Optimal Control Problem                                                |
| PD        | Proportional-Derivative                                                |
| PI        | Proportional-Integral                                                  |
| PID       | Proportional-Integral-Derivative                                       |
| RDP       | Ramer-Douglas-Peucker                                                  |
| RMSE      | Root Mean Square Error                                                 |
| ROS       | Robot Operating System                                                 |
| Rprop     | Residual Propagation                                                   |

# Chapter 1

## Introduction

This chapter is divided into five sections. Section 1.1 provides the context of the dissertation by highlighting problems associated with the trajectory tracking of mobile robots. Section 1.2 highlights the motivations for this thesis, emphasizing the benefits of enhancing controllers and integrating obstacle avoidance capabilities. Section 1.3 details the objectives of the thesis. In section 1.4, the contributions to the scientific community are presented. Lastly, section 1.5 defines the structure of this document.

### 1.1 Context

Nowadays, mobile robots are employed in diverse fields, including space exploration, underwater research, and industrial and military applications. In all these mentioned applications, some tasks must be carried out in dynamic and unpredictable environments. Furthermore, for a robot to navigate autonomously and perform designated tasks, four subsystems are required: cognition (the robot's decision-making on how to act to achieve its goals); motion control (managing the robot's motors to follow the desired trajectory); perception (interpreting sensor data to extract relevant information); and localization (determining the robot's position within its environment) [4].

Advances in trajectory tracking have been crucial in improving the robustness and precision of mobile robots in the motion control subsystem. This subsystem involves the use of control strategies that enable robots to navigate through complex and dynamic environments. The integration of advanced sensors, such as Light Detection and Ranging (LiDAR) and cameras, allows motion control systems to adapt to changes in the environment in real time. Recent research has focused on developing robust motion control algorithms that can handle uncertainties and disturbances, ensuring mobile robots' reliable and safe operation in challenging environments [4, 5].

On the other hand, the cognition and perception subsystems play a crucial role in enabling robots to sense and interact intelligently with the environment, enabling collision detection and obstacle avoidance. The cognition subsystem involves high-level decision-making, where robots evaluate their goals, plan actions, and make strategic choices to achieve goals efficiently, for example, by minimizing energy consumption or path length. The perception subsystem focuses on

interpreting sensor data to extract meaningful environmental information, such as the obstacle's location. Advances in computer vision and sensor technologies have given robots the ability to recognize objects and navigate through dynamic environments. Integrating these subsystems is crucial to achieving smooth and efficient autonomous operation of mobile robots, making it possible to avoid collisions with objects and people, thus preventing accidents [4, 6].

## 1.2 Motivation

In industrial settings, mobile robots typically follow predetermined paths and rely on safety systems to halt operations when obstacles are detected. While safety measures are essential to prevent collisions, they can also result in unwanted stops, even when minor obstructions only block a small part of the robot's path. In such circumstances, a more appropriate approach is required. This approach would permit robots to navigate around obstacles in a timely and secure manner, eliminating the necessity for human intervention.

One limitation of conventional controllers is their predominantly reactive nature, which hinders their ability to use path information to enhance precision. In contrast, predictive controllers have a greater potential for trajectory tracking, allowing robots to move at higher speeds while maintaining or improving accuracy and safety.

Moreover, implementing sophisticated control mechanisms becomes crucial in scenarios where the transported loads are of high value or sensitive to abrupt movements. This underscores the importance of maintaining control over acceleration limits to mitigate the risk of the robot causing the transported loads to fall. Consequently, trajectories with a smooth design and extremely precise control over the evolution of the robot's speed and position are required. This level of control ensures the safety of transported loads and the overall reliability and effectiveness of the mobile robot in an industrial environment.

Another challenge currently faced is the lack of robust obstacle avoidance algorithms. These algorithms would enable mobile robots to make real-time decisions and dynamically adjust their paths in response to unexpected obstacles. The motivation to address this issue is the potential for considerable gains in process efficiency. Robots with advanced obstacle avoidance controllers can avoid obstacles smoothly, minimizing workflow interruptions and optimizing productivity. Integrating a controller that can navigate obstacles quickly and safely improves the autonomy of mobile robots and reduces the need for manual intervention. This enhancement is significant in industrial environments where time-sensitive processes require uninterrupted operation.

Finally, this thesis was produced in collaboration with the Centre for Robotics in Industry and Intelligent Systems (CRIIS) at INESC TEC – Institute for Systems and Computer Engineering, Technology and Science. CRIIS specializes in developing advanced solutions in the fields of automation and industrial robotics. A substantial part of CRIIS's research is in the field of mobile robot navigation. Therefore, improving navigation systems at CRIIS depends heavily on the progress made in both trajectory control and trajectory planning.

## 1.3 Goals

The dissertation has two main objectives. The first is to develop a controller that enables precise trajectory tracking at high speeds with limited accelerations. In industrial applications, tricycle robots are commonly used due to their predictable motion and because they are mechanically simpler. However, in recent years, omnidirectional robots have become more prevalent in this field due to their complete maneuverability, highlighting the need for new controllers suited to this steering geometry. As such, the omnidirectional steering geometry has been chosen as a reference to allow better specialization of the controller. Secondly, accurate obstacle detection is also essential for effectively employing this control module for obstacle avoidance. Furthermore, the robot must be capable of real-time obstacle detection and avoidance to prevent collisions and select the appropriate trajectory.

As such, this dissertation aims to achieve the following objectives:

1. Study the literature on trajectory tracking control, trajectory definition, and collision avoidance;
2. Develop a predictive controller for omnidirectional robots;
3. Develop an obstacle avoidance algorithm able to contour stationary obstacles;
4. Test the developed controller in real omnidirectional robots and compare it with other controllers from the literature;
5. Test the developed algorithm for obstacle avoidance with several obstacle configurations.

## 1.4 Contributions

This work had several contributions to the fields of trajectory tracking and collision avoidance, which are summarized as follows:

- A comprehensive literature review focusing on trajectory tracking, trajectory definition, and collision avoidance;
- Development of a Nonlinear Model Predictive Control (NMPC) framework for omnidirectional ground mobile robots, with potential application to different robot steering geometries in future developments.
- Creation of an obstacle avoidance algorithm featuring real-time path generation based on splines to enable collision avoidance for mobile robots.

Furthermore, the second contribution, centered on trajectory tracking control, resulted in the publication of a conference paper:

- Conference paper accepted for publication in 16th APCA International Conference on Automatic Control and Soft Computing (CONTROLO 2024) with the title “A Nonlinear Model Predictive Control Strategy for Trajectory Tracking of Omnidirectional Robots” [7].

## 1.5 Structure of the Document

The dissertation is organized into five chapters. Thus, following the introduction, the document is structured as follows:

- Chapter 2 provides background information and a literature review on trajectory tracking and collision avoidance for mobile robots;
- Chapter 3 details the controller architecture developed for trajectory tracking. Additionally, a model of an omnidirectional mobile robot is derived, taking into account the goals of this thesis. Ultimately, the controller's performance is evaluated using results from both simulation and real-world environments, and compared with controllers from the literature;
- Chapter 4 presents the proposed collision avoidance algorithm. It also includes the algorithms used for trajectory definition. The chapter concludes with an analysis of the collision avoidance algorithm's performance in simulation;
- Chapter 5 lists the conclusions of this dissertation, describing the goals achieved and the problems encountered during the dissertation. Furthermore, recommendations for future research are provided.

## Chapter 2

# State of the Art

This chapter provides a literature review on trajectory tracking and obstacle avoidance for mobile robots. Firstly, section 2.1 examines the primary controllers utilized for the autonomous navigation of mobile robots. Subsequently, section 2.2 investigates various trajectory definition techniques, including a set of points, lines and arcs, and parametric curves. Finally, section 2.3 covers the sensors that can be employed to identify obstacles and the algorithms utilized to avoid collisions. Subsequently, in section 2.4, the literature review is summarised, and the conclusions drawn from the analyses carried out are presented.

### 2.1 Trajectory Tracking

Trajectory tracking is defined as “the problem concerned with the design of control laws that force a vehicle to reach and follow a time-parameterized reference” [8]. In a wheeled ground mobile robot, the trajectory tracking problem involves determining the wheel velocities required to move the robot to a reference position with a given orientation within a specified time.

In order to achieve trajectory tracking for a mobile robot, a variety of controllers have been employed up to the present day. Consequently, these controllers can be categorized into two primary groups: reactive controllers and predictive controllers [4]. On the one hand, reactive controllers are distinguished by their retrospective approach to robot position control. In essence, reactive controllers calculate errors in the robot’s position and orientation relative to reference values during an initial phase. Subsequently, adjustments are made to the robot’s speed, considering the magnitude and direction of these errors. Moreover, the most common reactive controllers that have been developed until now are the Proportional-Integral-Derivative (PID) controllers and fuzzy logic controllers. Additionally, classical controllers such as Lyapunov-based controllers and state feedback controllers also possess reactive characteristics [5].

On the other hand, predictive controllers, also known as anticipation controllers, take into account the future trajectory to determine the robot’s velocity. Some examples of predictive controllers include Feed Forward (FF) control, and Model Predictive Control (MPC) [9].

In the following subsections, two separate literature analyses will be conducted along with the necessary background information due to the significant differences between the two groups. Additionally, more attention will be given to predictive controllers.

### 2.1.1 Reactive controllers

Reactive controllers were the first developed solution to the challenge of trajectory tracking. They present a straightforward approach by using the real-time error in the robot's position and orientation to modify its speed and direction. An early example of a reactive controller is the work by Watanabe et al. [10], which developed and tested a feedback loop using a combination of Proportional-Derivative (PD) and Proportional-Integral (PI) controllers for mobile robots. This controller was designed specifically for an omnidirectional robot with three wheels. It employs a set of control laws based on the robot's kinematics and dynamics.

The control strategy outlined by Watanabe et al. [10], faced challenges when applied to Multi-Input Multi-Output (MIMO) systems. To address this limitation, they simplified the equations by decoupling the position and orientation components ( $x$ ,  $y$ ,  $\theta$ ). This adaptation allowed for effective trajectory tracking within the system's constraints using a reactive control approach.

In contrast to PID controllers, fuzzy logic controllers do not rely on precise mathematical models. Instead, they use linguistic rules to handle uncertainties within the system. These controllers employ linguistic variables such as “close”, “far”, “fast”, and “slow” to inform decisions about adjustments to the robot's velocity. Fuzzy logic controllers have been successful in managing nonlinear and complex systems, making them suitable for specific applications in mobile robotics.

Li et al. [11] and Liang et al. [12] have developed fuzzy logic controllers for different robotic configurations. Li et al. [11] designed a fuzzy logic controller for an Ackerman robot. The controller received inputs in the form of distance and orientation errors of the robot, relative to a reference. It produced outputs that determined the wheel velocities and their orientation. The implementation involved two separate rule bases for controlling wheel speed and direction. Two distinct controllers were utilized, each equipped with five membership functions for every input variable. The use of a dual-controller approach enabled precise control over both wheel speed and orientation. By utilizing linguistic variables and a rule-based system, the controller was able to handle uncertainties present in the robot's environment, resulting in an adaptive and effective trajectory-tracking system.

On the other hand, Lyapunov-based controllers utilize a Lyapunov function to evaluate a system's dynamics and maintain its stability. The control laws are formulated based on the kinematic and dynamic model of the robot to determine its velocity at each instant, ensuring that it follows a specified reference trajectory [5]. While Lyapunov-based controllers ensure stability, the generated control laws lack a clear and intuitive physical interpretation. This limitation restricts the application of these controllers in dynamic and unpredictable environments. Moreover, the trajectory defined by the Lyapunov-based controller is not well-known, until it converges to the reference trajectory, which limits its frequent use in such contexts.

Benaoumeur et al. [13] present a backstepping controller designed for a differential robot, utilizing Lyapunov-based stability analysis principles. Similarly, Mu et al. [14] adopt a comparable strategy, but with the implementation of a sliding mode controller. One significant benefit of these controllers is their effectiveness in managing nonlinear systems. Therefore, both approaches utilize the robot's dynamic model without the need for linearisation. However, it is important to note that these controllers do not explicitly address uncertainties related to localization and parameter constants within the model. Therefore, all tests were conducted at low speeds to mitigate the effects of slippage between the wheels and the ground, as well as other disturbances.

Later on, Jianbin et al. [15] introduce an adaptive sliding mode controller designed for an omnidirectional robot. This new approach demonstrates the ability to manage input constraints, model uncertainties, and external disturbances. Despite its improved robustness, certain limitations still exist. When applied to the control of large robots in dynamic and unpredictable environments, the trajectory must still be constrained and predictable. Not satisfying these constraints could lead to accidents that could potentially harm people.

While reactive controllers have proven effective in many scenarios, their dependence on current error information and retrospective adjustments can limit their performance in dynamic and unpredictable environments. So, research on trajectory tracking controllers has shifted towards predictive approaches by incorporating predictive features into conventional reactive controllers.

Feedforward controllers are an example of this approach, aiming to counteract the anticipated effects of disturbances and uncertainties on the robot's motion. These controllers use a feedforward control signal, computed based on the anticipated disturbances or reference trajectory, to compensate for these anticipated disturbances and achieve better tracking performance. Feedforward controllers are frequently combined with feedback controllers, such as PID controllers, to enhance the overall performance of control systems.

The work by Sousa et al., [16] introduces a trajectory controller for omnidirectional robots. This algorithm incorporates PI controllers to regulate the angular speed of the wheels, PD controllers for precise positioning control, and FF controllers to anticipate future reference signals. This is achieved by using the first two derivatives of the future trajectory and an approximate dynamic model of the system to be controlled. The structure of the controller is presented in figure 2.1, where some additional features like the anti-windup and the Hammerstein algorithms are presented. These features are important to model motor nonlinearities, with the anti-windup algorithm addressing issues related to actuator saturation, and the Hammerstein algorithm compensating for dead zones, thus improving the controller's overall performance.

The feedforward component enables the addition of predictive control by directly modifying the velocity reference signals. To achieve this, the feedforward block must be designed as the inverse of the model, resulting in the calculation of reference signal derivatives. Additionally, the model transfer function can be simplified to a first-order system, which is identifiable by its time constant and gain. Therefore, Sousa et al. [16] use the first and second derivatives of the references to improve the alignment accuracy between the actual pose of the robot and the reference pose, defined by the trajectory.

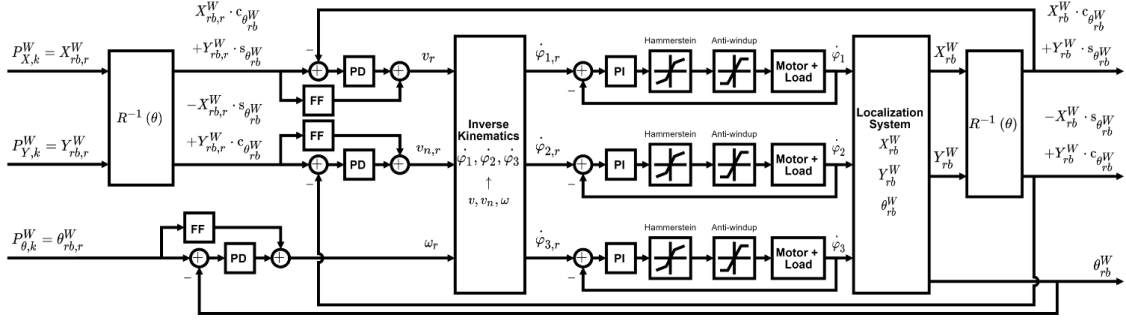


**Figure 2.1:** Feed-Forward PID-based controller for omnidirectional robots with predictive characteristics, proposed by Sousa et al. [16]

Although the addition of the feedforward component improved the performance of the controller, certain challenges remain. In particular, difficulties arise in dealing with MIMO systems, requiring the decoupling of pose components and the implementation of individual control loops. Additionally, the feedforward component requires a precise model, which is a challenging task in highly nonlinear problems such as the trajectory tracking of mobile robots. Moreover, the performance of feedforward controllers is substantially better in approximately linear systems.

Another significant challenge is the difficulty in applying these controllers to non-differentiable trajectories, such as the corner of a square trajectory. This necessitates approximating the future trajectory with differentiable equations, which may not always provide an accurate representation. Such approximations can lead to performance degradation and may not fully capture the system's dynamics, especially in highly complex or abrupt trajectory changes.

### 2.1.2 Predictive controllers

In recent years, predictive controllers have gained significant attention due to advancements in computational power. This has enabled the implementation of increasingly sophisticated predictive control algorithms. Additionally, the increased computational efficiency has allowed for the use of predictive controllers in real-time systems, specifically in trajectory-tracking problems.

Model Predictive Control, also referred to as receding horizon control, is a frequently employed control strategy. It is characterized by the iterative solution of an Optimal Control Problem (OCP) at discrete points in time. The current control action is determined by minimizing an objective function, considering a dynamic model of the plant and constraints imposed on the system. Based on the given dynamics, the system's behavior is predicted over a moving horizon. However, due to perturbations affecting the real plant and the inevitable mismatch between the modeled and real dynamical systems, the computed prediction differs from the real behavior [17].

To overcome this challenge, the model predictive controller recalculates the OCP at the next time step, typically at the next sampling instance. The current measured or estimated state of the system is used as the new initial condition for the problem formulation. Thus, the prediction horizon is adjusted, and a new control action is computed. Simultaneously, the system's prediction

is modified to match the current conditions, reducing the impact of external disturbances and model inaccuracies. Therefore, the principle of MPC for time-continuous processes is illustrated in figure 2.2, where on the left side, the evolution of the system states is represented, and on the right side, the control signals are represented.

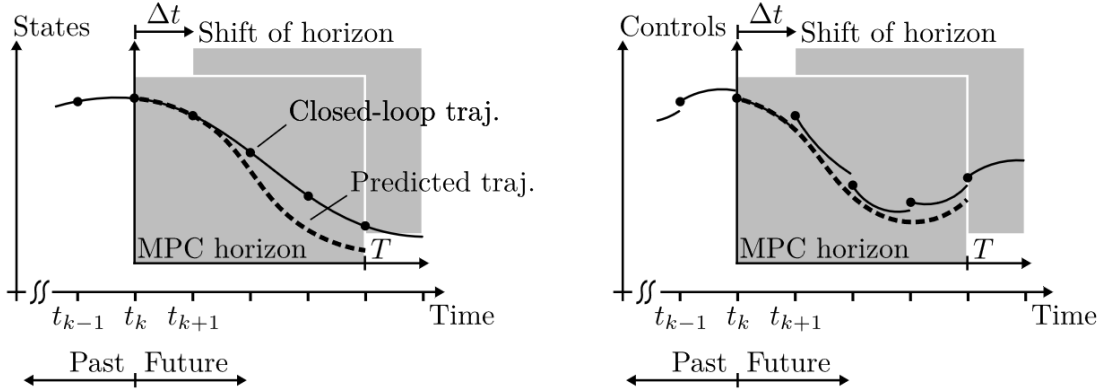


**Figure 2.2:** Schematic illustration of the functioning of an Model Predictive Control [17]

Nonlinear Model Predictive Control (NMPC) encounters a unique challenge compared to linear MPC, as it generally cannot solve convex quadratic programs precisely at each sampling time. This difficulty arises from the need to balance performing a nonlinear iteration process until a specified convergence criterion is achieved and stopping the process sooner to obtain an approximate solution within a predetermined computational time limit. Significant progress in the past decade has reduced both computational time and approximation errors [18].

As described by Käpernick in [17], there are two primary approaches for numerically solving an OCP: direct and indirect methods. Direct methods involve discretizing the problem formulation before optimizing it, while indirect methods consist of performing the optimization before the discretization stage. Additionally, because robots are controlled by discrete reference values in motor speeds, direct methods are more commonly used. Their simpler implementation also makes direct methods more appealing.

Therefore, as explained in [18] and [17], direct methods are effective in managing complex nonlinear systems and constraints and are computationally efficient for specific issues. However, their computational complexity tends to increase, especially with a longer prediction horizon. Additionally, these methods may lack sensitivity information, which is critical for certain applications. On the other hand, indirect methods use analytical techniques to derive solutions and provide sensitivity analysis. They can be computationally efficient for certain types of systems and problems. However, they face challenges when dealing with complex, high-dimensional, or poorly behaved systems. Furthermore, the need for symbolic differentiation and the difficulty in handling constraints may be limiting factors.

Conceição et al. [19] and Ferreira et al. [20] have introduced similar nonlinear MPC approaches for omnidirectional robots. They adopt a direct strategy combining low-level PI controllers to regulate the wheel velocities and a high-level nonlinear MPC for managing the robot's

position, as shown in figure 2.3. The cost function in their controller has been designed to minimize the difference between the predicted pose and the reference pose within a predicted horizon ( $N_p$ ). In addition, the controller minimizes the variation of the control input ( $V$ ,  $V_n$ , and  $W$ ) over a specified control horizon ( $N_u$ ). By minimizing the variation of the control input rather than its absolute value, the controller effectively mitigates potential steady-state errors. This approach also allows for control over the aggressiveness or smoothness of the control signal, which is critical in mechanical systems. Abrupt variations in aggressive control signals can induce mechanical issues, making the management of signal dynamics essential for system integrity.

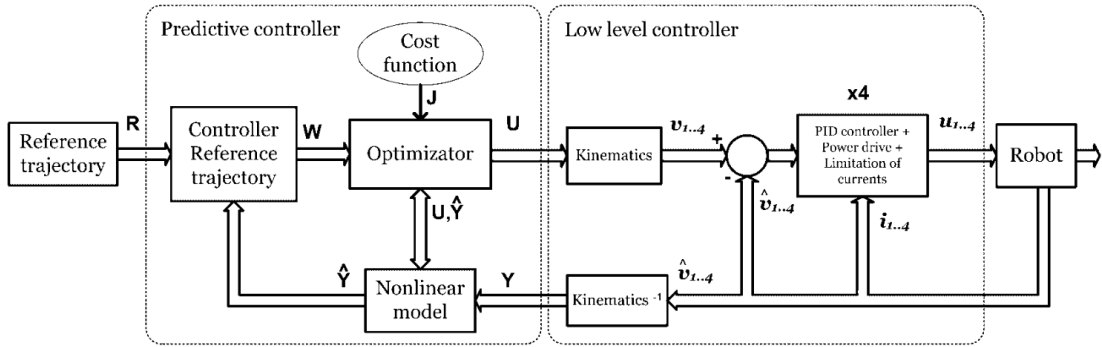


**Figure 2.3:** Scheme of the control strategy proposed by Conceição et al. [19]

Additionally, their controller was also tested using three gradient descent methods: the steepest descent and two nonlinear conjugate gradient methods – Fletcher-Reeves and Polak-Ribière. Moreover, the optimizer does not directly take into account the maximum speed constraints; instead, the reference speeds of each wheel are scaled at the end to ensure that the robot's movement complies with the imposed maximum speed constraints. This scaling of the reference velocities ensures that the robot's motion remains unchanged at a lower speed.

Alternatively, Pei et al. [21] proposed an indirect method by using the Continuation method in combination with the Generalized Minimum Residual (C/GMRES) [22]. They presented a nonlinear MPC for a differential traction robot using the C/GMRES algorithm, in which a differential equation is obtained by the continuation method to update the control sequence, and the generalized minimum residual (GMRES) method is used to solve the linear equation efficiently. The C/GMRES method has the advantage of determining the control sequence increment at each time interval without iteratively solving the nonlinear optimization problem, leading to a significant reduction in computation time.

Another distinction arises from the direct use of both equality and inequality constraints in the optimization algorithm. This involves deriving the associated optimality conditions and incorporating them into a vector function, creating a system of equations that depend on the sequences of optimal control and Lagrange multipliers.

Besides the optimization algorithm, some researchers also have proposed changes in the cost function. Nascimento et al. [23] uses the distance between the robot pose and the reference coordinate, as seen in figure 2.4 (right), instead of the difference between the robot's pose and the

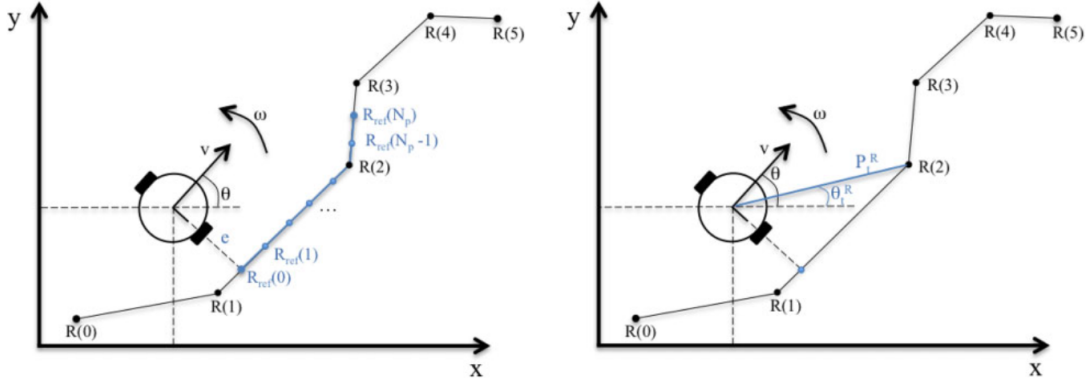


**Figure 2.4:** Classical NMPC approach (left) and Nascimento et al. [23] approach (right)

predicted trajectory coordinate as seen in figure 2.4 (left). Besides that, they also penalize the difference between the angle of the robot in the world frame (the orientation of the robot in the world frame) and the angle between the robot and the “virtual leader” (trajectory position points). Given that these modifications increase the nonlinearity of the system, they use the 1-norm in the control effort term of the modified cost function, to maintain stability. In addition, they adopted the residual propagation (Rprop) optimization algorithm [24], which can handle the nonlinearities introduced by the use of the 1-norm.

As highlighted by Nascimento et al. [23], the common approach in some works is to use different gains for different trajectories, to achieve a balance between position and orientation errors. In contrast, their methodology uses a single set of gains, which is possible due to the elimination of dependency on trajectory equations.

Moreover, their controller was developed and tested for differential traction robots, where the use of the angle between the robot position and the target position improves the convergence to the trajectory. However, for an omnidirectional robot, this information is less relevant since they can move in all directions regardless of their orientation.

## 2.2 Trajectory Definition

When defining a trajectory, it is important to define two key concepts. Firstly, path planning describes the geometric and mathematical route from the starting point to the destination point, while avoiding obstacles. Secondly, trajectory planning defines this path as a function of time, specifying the robot’s position at each point in time [4].

Therefore, the trajectory definition can be approached in two different ways. In the first approach, the path is defined without taking into account the constraints on the robot velocities, which may result in suboptimal paths. In a second approach, the path and trajectory are defined simultaneously to achieve a specific time goal or reduce the robot’s energy consumption [4, 6].

Additionally, to define the paths for the robot, there are several alternatives to consider, including sequences of points, lines, arcs, and parametric curves. The simplest alternative involves using

a sequence of points since any trajectory defined by any other alternative can be translated into this one through discretization. However, this method has some drawbacks, such as the loss of information and the amount of data required to define a path. These two disadvantages are interrelated, to mitigate one, the other often needs to be amplified. For instance, increasing the distance between points will reduce the number of points required to define the trajectory, but it will also decrease the accuracy of curved sections. Additionally, the loss of information has consequences, such as the necessity of a numerical approach when calculating the robot's closest point to the path, regardless of its shape.

In the study by Sousa et al. [16], a trajectory  $T$  is defined as a set of points specified in both spatial (position and orientation) and temporal dimensions:  $T = \{P_0(t_0), P_1(t_1), \dots, P_{N-1}(t_{N-1})\}$ , where  $P_k(t_k) = \{P_{X,k}, P_{Y,k}, P_{\theta,k}\}(t_k)$  represents a trajectory pose, composed by its position ( $P_{X,k}$  and  $P_{Y,k}$ ) and orientation ( $P_{\theta,k}$ ), at a certain time instant ( $t_k$ ). To achieve a constant linear velocity, they imposed constraints on the positions of the points and their corresponding times, ensuring that the distance and time interval between consecutive points remained constant.

In contrast, lines and arcs offer the advantage of representing a wide range of trajectories while requiring fewer parameters than point sequences. Additionally, these representations enable the analytical determination of the closest point on the trajectory to the robot. For example, the closest point to a line is determined by the intersection point between the line and a perpendicular line passing through the robot's position. Similarly, the closest point to an arc or circle can be determined by the closest intersection between the circle and a line connecting the robot's position and the center of the circle. However, defining smooth paths using only lines and arcs can be quite complex due to the need for continuous transitions between segments, maintaining curvature continuity, and ensuring that the path adheres to the robot's kinematic constraints.

Ouach et al. [25] present a method for defining trajectories using a combination of lines and arcs. They start by establishing a path consisting of lines connecting the robot to the goal, generated through a Probabilistic Roadmap Method (PRM). Then, arcs are introduced using the fillet technique to smooth the transition between two consecutive lines. The curvature of these fillets is precisely defined to ensure that the circle describing each arc is tangent to both lines. Furthermore, the points where the arcs tangentially intersect with the lines are positioned equidistantly from the intersection point of the two lines. This distance is set to half the shorter line's length, contributing to a smoother and more predictable trajectory. Although this approach creates apparently smooth paths, it also introduces curvature discontinuities. This occurs because the curvature of a line segment is constant and equal to zero, whereas the curvature of an arc is constant and different from zero. Therefore, these abrupt changes in curvature can be problematic when the robot moves at high speeds.

Finally, parametric curves provide a more versatile way to define trajectories, allowing for smoother and more continuous paths by seamlessly integrating varying degrees of curvature. These curves are characterized by mathematical equations, often involving polynomials and trigonometric functions. Among the most well-known are splines, generally described as piecewise polynomial parametric curves of order  $N$ , defined by control points, as illustrated in figure 2.5.

Equation 2.1 represents a generic way to define a spline with one polynomial of order  $N$ , where  $\mathbf{P}$  is a column vector containing  $N+1$  control points of the spline, and  $\mathbf{M}$  is an  $N+1$  by  $N+1$  matrix containing specific parameters for each spline. For instance, in the case of a Bézier spline of 3rd-order, as shown in figure 2.5, the matrix  $\mathbf{M}$  is equivalent to the one in equation 2.2. Additionally, the dimensions of the control points are not limited or interdependent, enabling each dimension (such as  $x$  and  $y$ ) to be considered independently [26, 27]. To provide a comprehensive understanding of splines, appendix C presents alternative notations for representing splines as piecewise polynomial parametric curves, offering further insight into their mathematical formulation.

$$\mathbf{Q}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 & \dots & t^N \end{bmatrix} \mathbf{M} \mathbf{P}, \quad t \in [0, 1] \quad (2.1)$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad (2.2)$$

Therefore, the primary benefit of splines lies in their ability to generate smooth trajectories with various shapes. Furthermore, their polynomial representation allows for the analytical derivation of essential information, including curvature, velocity, and acceleration, based solely on the knowledge of control points and the spline matrix.

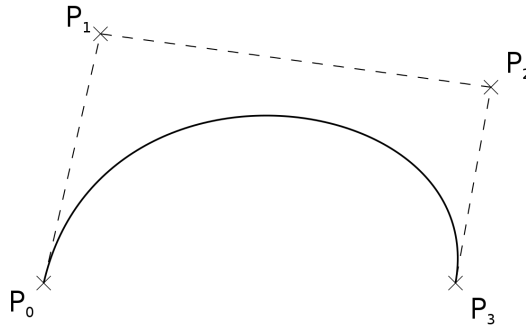


**Figure 2.5:** Bézier spline of 3rd-order

Furthermore, in the work by Sprunk et al. [28], various characteristics are presented for evaluating and selecting the appropriate type of spline for defining trajectories in mobile robots. The spline is defined as a piecewise polynomial parametric curve in the two-dimensional plane, represented as  $\mathbf{Q}(t) \in \mathbb{R}^2$ , and is composed of  $n$  spline segments  $S_i(t) \in \mathbb{R}^2$ . To understand the continuity between two spline segments, two fundamental concepts are introduced:

- **Parametric Continuity:** A parametric curve exhibits  $n$ th-degree parametric continuity in the parameter  $t$  if its  $n$ th derivative  $\frac{d^n}{dt^n} \mathbf{Q}(t)$  is continuous. This property is known as  $C^n$  continuity.

- **Geometric Continuity:** Nth-degree geometric continuity, referred to as  $G^n$  continuity, does not require the left-hand and right-hand limits of the nth derivative  $\frac{d^n}{dt^n}Q(t)$  to be equal. Instead, it requires the directions of these vectors to align, meaning that the derivative at the intersection point between two spline segments just needs to have the same direction and not the same magnitude. It is important to note that while a spline with  $C^n$  continuity also satisfies  $G^n$  continuity, the reverse is not always true.

Since the derivatives of a polynomial are well-defined and continuous, parametric and geometric continuities are only violated at the intersection points between two consecutive spline segments. Therefore, taking into account the parametric and geometric continuities, the following characteristics are defined to evaluate the performance of each spline in trajectory definition for mobile robots:

- **Continuity in velocity:** Besides inherent  $C^0$  continuity (position continuity), demanding  $C^1$  continuity (velocity continuity) ensures a smooth transition in the robot's heading, eliminating the need for infinite rotational accelerations;
- **Continuity in curvature:** Trajectories for wheeled robots should exhibit  $G^2$  continuity (continuity in curvature), ensuring a consistent and smooth angular velocity.
- **Stitching:** Continuities are ensured during the transition between two consecutive polynomials of a spline, being inherent for some and requiring manual intervention for others;
- **Localism:** Localism ensures that changes made to a single segment of a multi-segment spline only affect a bounded neighborhood of that segment. It allows local modifications without global reshaping and potentially reduces computational costs during updates;
- **Correlation between shape and parameters:** This property facilitates trajectory planning, establishing direct correlations between a spline's parameters and its intuitive shape. It simplifies the task of defining splines passing through specific points or avoiding designated areas in trajectory planning;
- **First and second derivative at start:** To update a trajectory that is already being executed by a robot, it is necessary to generate trajectories for a moving robot that can smoothly match the required heading and orientation/steering angle. This requires the ability to match the first and second derivatives.

In addition, when selecting the order of a spline and the number of constituent polynomials, it is important to consider various factors carefully. Both a single high-order polynomial spline and a spline composed of multiple low-order polynomials can represent the same arbitrary path. However, choosing a single high-order polynomial ensures all the continuities but sacrifices localism. On the other hand, using multiple low-order polynomials allows for localism, but may not satisfy all the necessary continuities. As a result, cubic splines (3rd-order) are frequently used

in literature because their low-order polynomials allow for localism while still providing enough parameters to maintain essential continuities. [26, 27].

Table 2.1, retrieved from [28], presents a comparison of the four most common uniform rational splines. Moreover, the evaluation of each spline's properties, previously explained, is conducted in their cubic version. Some additional notes provided by [28] explain that Hermite and Bézier splines operate on a per-segment basis, meaning that manual stitching is necessary to create splines comprising multiple segments (multiple polynomials). In contrast, Catmull-Rom Splines and B-Splines have an inherent stitching process. Additionally, it is possible to achieve  $C^2$  continuity in Bézier and Hermite splines by sacrificing localism during the stitching process. However, this can cause the resulting splines to be highly unstable and exhibit pronounced oscillations in response to single parameter changes.

Therefore, although cubic splines are effective in defining smooth predetermined paths for mobile robots, they have limitations when making real-time trajectory adjustments, such as for obstacle avoidance. This limitation is directly linked to the Degrees of Freedom (DoFs) inherent in cubic splines. To meet all the requirements for real-time trajectory definition, in controlling mobile robot trajectories, it is essential to increase the polynomial order. This modification involves increasing the number of control points in each polynomial that makes up the spline, increasing the overall degrees of freedom. Furthermore, Sprunk [28] successfully meets all the outlined requirements by employing Quintic Bézier splines (5th-order) for real-time trajectory definition.

**Table 2.1:** Key properties of cubic spline families [28]

| Property                                | Hermite | Bézier | Catmull-Rom | B-Spline |
|-----------------------------------------|---------|--------|-------------|----------|
| Stitching                               | manual  | manual | inherent    | inherent |
| $C^1$ continuity                        | X       | X      | X           | X        |
| $C^2$ continuity                        | -       | -      | -           | X        |
| Strong correlation                      | -       | X      | X           | -        |
| Localism                                | X       | X      | X           | X        |
| Freely set 1 <sup>st</sup> deriv. start | X       | X      | X           | X        |
| Freely set 2 <sup>nd</sup> deriv. start | -       | -      | -           | -        |

Additionally, although the parameter  $t$  is commonly interpreted as time, a decoupling between position and velocity is introduced in several literature papers, such as [28, 29]. This separation allows for precise velocity definition within each spline segment, enabling segments of varying sizes where the robot can maintain a constant speed.

Finally, Munoz et al. [29] propose a real-time path-planning algorithm that considers both kinematic and dynamic constraints. Moreover, the authors combine all three alternatives – parametric curves, lines, arcs, and sets of points – to create smooth trajectories. They start by using lines and arcs to determine a direct path between the robot position and the goal position. Next, they restrict the map, taking into account the constraints imposed by the kinematic and dynamic models. Due to the close-form equations of the lines and arcs, the computational effort required is substantially reduced. The authors then define the control points for a  $\beta$ -spline [30], using the

constraints of the lines and arcs to create a smooth path without discontinuities. Finally, the path is discretized into a set of poses (position and orientation) over time to precisely define the velocity of the robot along the trajectory.

Furthermore, the final discretization method employed in [29] is also utilized to address trajectory tracking issues when implementing parametric curves or splines. Due to the arbitrary shape of the path described by the parametric curve, an analytical approach to finding the nearest point on the spline for controlling the robot's position is not feasible. Therefore, a real-time numerical approximation is required to achieve an accurate estimation. In their paper, Wang et al. [31] proposed an algorithm that combines a quadratic minimization method with a Newton method to improve convergence and reduce the number of divergences. However, their approach only achieved good results when certain restrictions on the spline shape were assumed. Chen et al. [32] presented a more robust approach, translating the point projection into a root-finding problem. The authors start by isolating the real roots using the Descartes method. They then prune invalid intervals with a simple classification method. Finally, they combine the bisection method with Newton's method, similar to the approach taken by [31], to find the nearest point to the spline. This approach divides the problem into segments, solving the divergence of the Newton algorithm and making the algorithm more robust.

## 2.3 Obstacle Avoidance

Obstacle avoidance is a critical aspect of robotics, especially in the context of autonomous navigation and control frameworks. It involves a robot's or independent system's ability to identify and navigate around obstacles that obstruct its designated path. Obstacle avoidance is essential for the safe and efficient operation of robots in complex and dynamic environments, as it reduces the risk of unintended collisions and damage. In order to successfully navigate obstacles, a robot or autonomous system must be able to detect them. The conventional method for robot navigation involves using sensors to interpret the robot's surroundings, making decisions on how to navigate around obstacles, and executing those decisions using its actuators [33].

Therefore, an obstacle-avoidance system should address two key aspects. Firstly, the selection of sensors that will be responsible for acquiring information about the obstacles' positions, and their respective distances from the robot. Secondly, the formulation of an algorithm that will determine the necessary adjustments to the robot's path to navigate around these obstacles. The following subsections will explore these two main points, providing background information and citing examples from existing literature that explore different options within each domain.

### 2.3.1 Sensors

As described in [33, 34], a range of sensors can be utilized for obstacle detection to enhance a robot's perception and navigation abilities. Infrared sensors, LiDAR sensors, and camera-based vision systems are among the most frequently used sensors due to their efficiency and reliability in detecting obstacles in various environments. Their versatility enables them to be used in a wide

range of situations, from outdoor environments to complex indoor spaces. While radar sensors have gained attention as an alternative to these sensors [35], it is still a costly technology with limited detection resolution. Although they can measure the velocity of obstacles, their use is less justifiable in indoor environments where weather resistance is a negligible factor.

In contrast, the use of ultrasonic and sonar sensors in obstacle avoidance is limited due to their dependence on the speed of sound. They are suitable for non-critical scenarios where the robot's speed is not a decisive factor [36]. However, their effectiveness is limited in high-speed situations. Therefore, their application is often restricted to specific use cases where their limitations do not affect overall performance. Additionally, the sensors' detection cone is typically wide, resulting in low precision when determining the position of obstacles.

In certain scenarios, contact switches, bumpers, and comparable sensors are utilized as safety measures to prevent collisions when the laser scanner fails to detect obstacles within its scanning range. These mechanisms are reactive measures that ensure the robot stops before colliding with an obstacle. Although effective for collision avoidance, they are not suited for the more intricate task of obstacle avoidance, which involves navigating around obstacles rather than merely stopping before collision.

Furthermore, the dimensionality of the sensors is also important. Although ground mobile robots primarily operate in a 2D plane in industrial environments, collisions with objects at different heights are still possible. Thus, relying solely on two-dimensional (2D) sensors may prove insufficient for robust and accurate obstacle detection. In contrast, 3D sensors can identify obstacles at any height, although this comes at the cost of generating more data. Managing the increased data volume requires more processing time, which can pose challenges, especially at higher speeds. This necessitates the use of additional techniques, such as parallel processing, advanced algorithms, or hardware acceleration, to address this issue [34]. Moreover, for safety applications, sensors must be certified according to relevant standards to ensure their reliability in detecting obstacles and preventing collisions.

Additionally, employing these sensors, whether integrated into the robot structure or in the environment, allows for different algorithms. The commonly used strategy for obstacle avoidance involves integrating various sensors, such as ultrasonic, LiDAR, radar, sonar, and cameras on the robot. These sensors enable an autonomous robot to perform a three-step process: sense, think, and act. The robot's sensors take in distance measurements of surrounding objects, providing data that enables it to detect obstacles and calculate their distances. Using this information, the robot can adjust its trajectory to navigate around obstacles while staying on its intended path. This process occurs in real-time and applies to most obstacle avoidance scenarios [33, 34]. While this method is generally effective, there are situations where more advanced techniques are necessary to efficiently reach an endpoint. For example, the optimal path between the robot's current position and the destination goal cannot be achieved with this approach due to the limitations of the robot's sensors. As the sensors are located within the robot structure, they cannot observe all the information from the obstacles present in the environment at a given time. This limitation prevents the sensors from obtaining precise dimensions of obstacles, which is essential for the robot

to navigate around them effectively.

However, 2D laser scanners, commonly referred to as 2D LiDAR sensors, are particularly prevalent in indoor environments. These sensors provide a high-resolution, two-dimensional map of the surrounding area by emitting laser beams and measuring the time it takes for the reflections to return. The precision and speed of 2D laser scanners make them ideal for applications where accurate, real-time obstacle detection and navigation are critical. Due to their capacity to effectively detect and map obstacles on the same plane as the robot, 2D laser scanners are widely employed in AMRs. Their relatively lower cost in comparison to 3D sensors renders them a cost-effective choice for numerous indoor applications.

### 2.3.2 Algorithms

To achieve more accurate and efficient obstacle avoidance, it is necessary to use advanced algorithms. These algorithms can be either classical, known as path-planning algorithms, or they may use machine learning techniques for enhanced performance.

A literature review on path planning and obstacle avoidance algorithms is available in [37] and [38]. In their review, they describe and explain the main algorithms that have been used to date, including the A\* algorithm [39], Artificial Potential Field (APF) [40], Neural Networks (NN)-based algorithms, and reinforcement learning algorithms.

On the one hand, path-planning algorithms are essential for calculating optimal, collision-free routes by taking into account the robot's current position, destination, and the positions of obstacles in the environment. By using this information to map out the area, the fastest route to a specific destination can be determined. Path-planning algorithms, such as A\* [39], Dijkstra's algorithm [41], and APF, are commonly used in scenarios such as maze navigation and autonomous vehicles. These algorithms enable real-time decision-making for the robot, ensuring it reaches its goal quickly while avoiding collisions [42].

Moreover, several modified versions of classical algorithms, such as A\* and APF, have been proposed in the literature to overcome their limitations. For instance, Sudhakara et al. [43] modified the A\* algorithm to minimize the number of turns made by the robot during its planned path. This enhances the robot's motion efficiency and reduces the required effort. On the other hand, Santos et al. [44] introduce a temporal optimization approach for the Time-Enhanced A\* (TEA\*) algorithm. This algorithm includes the time component as an extra dimension to address the A\* algorithm, specifically for coordination challenges within multi-robot systems. This enhancement allows for the simultaneous execution of multiple tasks, preventing collisions and deadlock scenarios, resulting in a reduction in the average execution time. Rostami et al. [45] present another example, addressing the improvement of the artificial potential field algorithm by focusing on mitigating local minimum situations, a significant drawback in its original form.

On the other hand, the use of machine learning expands the range of possibilities for obstacle avoidance. With the help of Artificial Intelligence (AI), an autonomous robot can not only find a path to its destination but also learn to adapt to a rapidly changing environment. This is accomplished through multiple testing stages where the robot is exposed to obstacles and environmental

changes. Assigning a task and providing a reward for correct completion can enable an AI to efficiently and effectively perform the task over time. This allows the robot to identify obstacles and devise efficient solutions to overcome them. The use of AI enables autonomous robots to respond to a wide range of situations, both expected and unexpected. This method of obstacle avoidance is particularly effective in autonomous vehicles as it eliminates potential human errors [46].

However, the use of machine learning in industrial settings is limited due to its inherent unpredictability. Given the potential for a single algorithmic error to cause significant damage, the algorithms employed must be consistent and predictable, taking into account the dimensions of the robot. To address this concern, it is important to incorporate additional safety measures, such as proximity sensors, certified for safety applications, to provide fail-safe protection.

Various studies exemplify the use of machine learning for obstacle avoidance algorithms, including those discussed in [47] and [48]. Duguleana et al. [47] introduced an algorithm that uses Q-learning and a neural network planner to address path-planning challenges. In contrast, Macek et al. [48] proposed a reinforcement learning approach where the learning agent receives only sparse reward information, including both temporal and structural aspects. It is important to note that these studies, among others, conducted tests and training in controlled environments. This means that there is a lack of insight into the robustness of the algorithms in industrial environments. Furthermore, the paths generated by the algorithms in these controlled settings showed poor trajectories with unnecessary curves.

To address the need for smoother trajectories, researchers have explored the fusion of machine-learning techniques with classical path-planning algorithms. For example, Yangmin and Xin [49] combined adaptive neural networks with particle swarm optimization to guide mobile robots. Particle swarm optimization served as a path planner, providing a smooth path for the adaptive neural network controller. Similarly, in [50], the author proposed a hybrid approach that integrates fuzzy logic algorithms with neural networks to navigate mobile robots effectively in chaotic conditions.

## 2.4 Summary

This chapter focused on the background and literature review of three key domains: trajectory tracking, trajectory definition, and obstacle avoidance. In environments with multiple robots, trajectory tracking is essential to improve overall performance and mitigate problems such as deadlocks and potential delays. On the other hand, the definition of different types of trajectories has the potential to optimize the performance of a robot by minimizing effort and energy consumption through the inclusion of smooth trajectories that take into account the inherent limitations of the robots. Furthermore, obstacle avoidance is crucial in industrial environments, which are typically unpredictable and dynamic. Thus, robust collision and obstacle avoidance algorithms are required in industrial environments to prevent potential accidents caused by the use of heavy industrial robots.

In Section 2.1, two types of trajectory tracking controllers have been introduced: reactive and predictive controllers. Reactive controllers, such as PID [10], fuzzy-logic [11, 12], and Lyapunov-based approaches [13, 14, 15], are known for their simplicity and low computational requirements. However, each type has its limitations. PID controllers, for example, struggle with multi-input multi-output systems. Lyapunov-based controllers, such as backstepping [13] and sliding mode [14, 15], ensure system stability. However, they lack a clear and predictable convergence trajectory. Fuzzy-logic controllers, on the other hand, face interpretability challenges and exhibit limited performance or computational demands, depending on the number of rules and membership functions employed. Moreover, all of them are characterized by a reactive response, which means that the control change is only applied after an error relative to a reference value is observed.

On the other hand, predictive controllers like feedforward controllers [16] and model predictive controllers [21, 19, 20, 23] use the dynamic and kinematic model of the robot in addition to the future trajectory reference, to anticipate the errors and correct the actuator signals before the errors even appear. While feedforward controllers are substantially simpler and have a low computational cost, MPCs demand a precise system model and come with significantly higher computational costs. Nevertheless, as FF controllers are a simple addition to a normal feedback controller, like PID, they have practically the same limitations, being not optimized to nonlinear MIMO systems.

In order to achieve high-speed and high-precision trajectory tracking, Model Predictive Control is a strong alternative. Despite its complexity and significant computational requirements, its nonlinear variant (NMPC) has proven to be highly effective in accurately handling nonlinear systems, such as mobile robot trajectory tracking. However, it requires a precise system model to predict future states accurately in alignment with real-world dynamics. The literature recognizes two families of NMPC controllers: direct methods [19, 20, 23] and indirect methods [21]. Direct methods are effective in managing complex nonlinear systems and constraints while maintaining computational efficiency. However, their computational complexity tends to increase, particularly with an extended prediction horizon. In contrast, indirect methods use analytical techniques to obtain solutions and provide sensitivity analysis. However, they face difficulties when dealing with complex, high-dimensional systems.

In Section 2.2, three alternatives to define a trajectory were presented: sequence of points, lines and arcs, and parametric curves. The most straightforward method is to use a sequence of points [16], as any other trajectory can be converted into this form through discretization. However, this method has some drawbacks, including the loss of information and the large amount of data required to define a path. In addition, it requires the use of a numerical approach when calculating the closest point of the robot to the path. In contrast, lines and arcs [25] offer the advantage of representing a wide range of trajectories while requiring fewer parameters than point sequences. Additionally, these representations enable the analytical determination of the closest point on the trajectory to the robot. However, the format of the trajectory presents limitations, as defining smooth paths can be quite complex.

Finally, when defining a trajectory, parametric curves, particularly splines [28], offer an attractive option. They are valued for their ability to define smooth trajectories with a limited number of parameters. However, since controllers are typically implemented discretely, it is essential to use point sequences, requiring a local discretization of parametric curves [28, 29]. Furthermore, to determine the section of the spline that requires discretization, it is necessary to identify the nearest point to the spline in real-time. This is a significant challenge when dealing with parametric curves, as it is not possible to identify the nearest point analytically. Therefore, numerical methods [31, 32] must be employed, which can result in computational delays.

In Section 2.3, the sensors and algorithms employed in obstacle avoidance are explored. Commonly used sensors include infrared sensors, LiDAR, and cameras [33, 34]. Additionally, radar sensors [35] and sonar sensors [36] find applications in some systems, although each comes with its limitations. Radar sensors, despite their effectiveness, remain a costly technology with limited detection resolution. On the other hand, sonar sensors are constrained by their dependence on the speed of sound, making them suitable for non-critical scenarios where the robot's speed is not a decisive factor. Finally, a careful evaluation is essential when choosing between 2D and 3D sensor versions, with considerations for either robustness or efficiency.

In the literature, algorithms for obstacle avoidance are presented in three different approaches. The first approach uses the information provided by the sensors directly, allowing the robot to maintain a safe distance from obstacles while following a predetermined path. Alternatively, more advanced algorithms such as classic path-planning algorithms [43, 44, 45] or those relying on machine learning techniques [47, 48] can be employed. Path-planning algorithms aim to determine the most efficient and safe route for a robot within its environment by evaluating potential paths and selecting the one that minimizes obstacles and aligns with a predefined criterion. In contrast, machine learning-based algorithms enable the system to adapt its behavior by learning from interactions with sensor inputs and the environment. Moreover, the use of machine learning in industrial settings is restricted due to its inherent unpredictability. In environments where algorithmic errors could cause significant damage, such as industrial settings, maintaining consistent and predictable performance is essential. Therefore, implementing safety measures like proximity sensors is crucial to ensure fail-safe protection.

## Chapter 3

# Trajectory Tracking

This chapter introduces the proposed architecture for trajectory tracking of an omnidirectional mobile robot, which employs Nonlinear Model Predictive Control (NMPC). Additionally, the performance of this controller is compared with various other controllers from existing literature. Section 3.1 outlines the controller architecture, detailing all necessary components. Section 3.2 covers the robot model equations, encompassing both kinematic and dynamic aspects. Section 3.3 focuses on formulating the optimization problem for NMPC. Section 3.4 discusses how the architecture manages acceleration and velocity constraints. Additionally, simulation results are shown in section 3.5, while section 3.6 presents experimental results conducted with real robots. Finally, Section 3.7 analyzes the outcomes from sections 3.5 and 3.6, forming conclusions on the evaluated trajectory tracking controllers.

### 3.1 Controller Architecture

The control strategy employs a hierarchical approach, depicted in figure 3.1. At the lower level, each wheel is equipped with a PI controller responsible for regulating the angular velocity of the motors ( $\varphi_i \mid i \in \{1 \dots 4\}$ ). At the higher level, a trajectory tracking controller based on a NMPC is composed of a trajectory discretization, a NMPC computation, and a scaling module.

During each control cycle, a discretized trajectory ( $\mathbf{x}_{ref}$ ) for a predefined path ( $P$ ) is generated, following a motion velocity profile ( $V$ ) regarding the robot's linear velocity. Subsequently, the nonlinear optimization problem is solved by the NMPC solver using the reference trajectory ( $\mathbf{x}_{ref}$ ) and the robot's current state ( $\mathbf{x} = [x, y, \theta]^\top$ ), obtained by the localization system. This computation determines the reference velocities for the robot ( $\mathbf{u} = [v_{ref}, v_{n,ref}, w_{ref}]^\top$ ). Then, the reference velocities of the robot's wheels ( $\varphi_{i,ref} \mid i \in \{1 \dots 4\}$ ) are computed by applying inverse kinematics to these velocities, followed by uniform scaling with a common factor to ensure they remain within specified limits without deforming the robot's movement. Finally, each PI controller utilizes these wheel reference velocities to determine the corresponding voltage ( $u_i \mid i \in \{1 \dots 4\}$ ) to be applied to each motor. It is noted that the diagram considers a robot with four wheels, as this

is the configuration of the omnidirectional robot discussed in this chapter. However, this architecture can be easily adapted for other robots with a different number of wheels or different steering geometries.

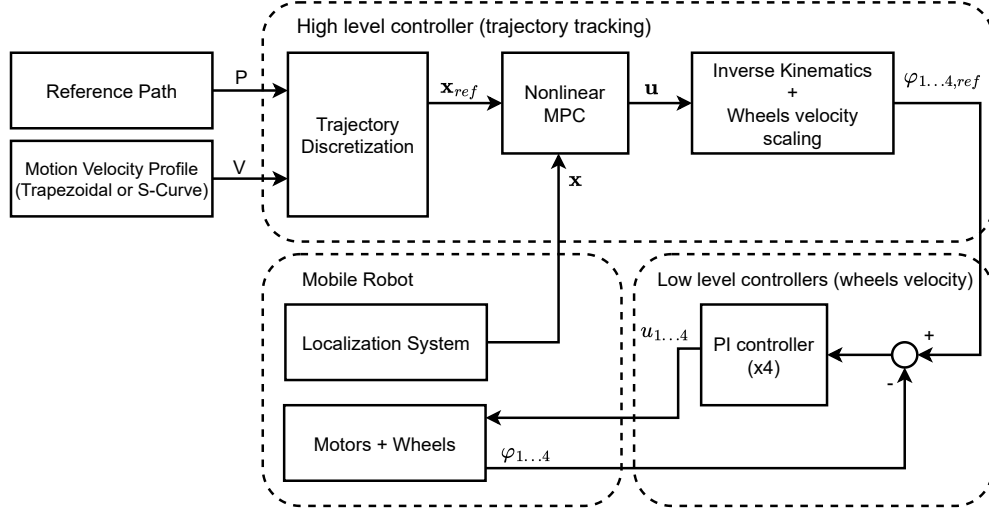


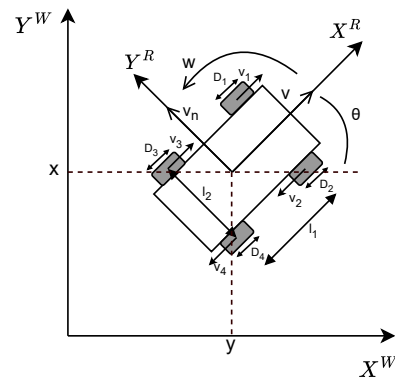
**Figure 3.1:** Proposed trajectory tracking controller for omnidirectional mobile robots

## 3.2 Robot Model

In this thesis, the four-wheeled omnidirectional robot illustrated in figure 3.2a was utilized. This robot was adapted by the CRIIS team using the Discovery Q2 robot platform from the Chengdu Hangfa Robotics<sup>1</sup>. Moreover, figure 3.2b presents the robot schematic in the world coordinate frame. Here, all the robot velocity vectors are presented, namely the robot's linear ( $v$ ,  $v_n$ ) and angular ( $w$ ) velocities, and the wheel's linear velocities ( $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$ ). Moreover, it also illustrates the relevant parameters such as the distances between the front and back wheels ( $l_1$ ) and between the right and left wheels ( $l_2$ ), and the diameter ( $D_i$ ) of each wheel ( $i \in \{1 \dots 4\}$ ).



(a) Hangfa Q2 robot:  
Codename "Adam"



(b) Robot Schematic

**Figure 3.2:** Four-wheeled omnidirectional robot

<sup>1</sup><http://hangfa.com/EN/robot/DiscoveryQ2.html>

### 3.2.1 Robot kinematics

Regarding the robot kinematics, two aspects are important. First, the transformation between the robot velocities ( $v$ ,  $v_n$ , and  $w$ ) and the wheel's linear velocities ( $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$ ) can be performed using equation 3.1, as described by Sousa et al. in [51, 52]. Additionally, the angular velocities for each motor wheel ( $\phi_1$ ,  $\phi_2$ ,  $\phi_3$ , and  $\phi_4$ ) can be determined by equation 3.2, where  $n$  represents the motor gear reduction ratio. This transformation is necessary to establish the reference angular velocities for each motor wheel, which are used by the wheels' PI controllers.

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -\frac{1}{2} \cdot (l_1 + l_2) \\ -1 & -1 & -\frac{1}{2} \cdot (l_1 + l_2) \\ 1 & 1 & -\frac{1}{2} \cdot (l_1 + l_2) \\ -1 & 1 & -\frac{1}{2} \cdot (l_1 + l_2) \end{bmatrix} \cdot \begin{bmatrix} v \\ v_n \\ w \end{bmatrix} \quad (3.1)$$

$$\phi_i = \frac{2n}{D_i} \cdot v_i, \quad i \in \{1 \dots 4\} \quad (3.2)$$

On the other hand, a transformation between the world coordinate frame ( $X^W$ ,  $Y^W$ ) and the robot coordinate frame ( $X^R$ ,  $Y^R$ ) is also necessary, since both the trajectory and the robot pose ( $x$ ,  $y$ ,  $\theta$ ) are expressed in the world coordinate frame. This transformation is defined by the differential equations 3.3, corresponding to a rotation around the z-axis of the robot orientation ( $\theta$ ).

$$\begin{aligned} \frac{dx(t)}{dt} &= v(t) \cdot \cos \theta(t) - v_n(t) \cdot \sin \theta(t) \\ \frac{dy(t)}{dt} &= v(t) \cdot \sin \theta(t) + v_n(t) \cdot \cos \theta(t) \\ \frac{d\theta(t)}{dt} &= w(t) \end{aligned} \quad (3.3)$$

Furthermore, by considering the robot reference velocities as the system input  $\mathbf{u} = [v, v_n, w]^\top = [v, v_n, w]^\top$  and the robot pose as the system state  $\mathbf{x} = [x, y, \theta]^\top$ , a simple nonlinear kinematic model for the omnidirectional robot can be derived using the state equation  $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ . However, the differential equations must be discretized to apply the model for a controller in discrete time. This can be done using several methods, such as the Euler method (see equation 3.4) or the 4th-order Runge-Kutta method (see equation 3.5), where  $dt$  represents the discretization step size. Alternatively, the discretization can be performed using the method described in equation 3.6, which is commonly used for odometry in omnidirectional mobile robots, as mentioned by Sousa et al. in [51, 52]. In this method,  $\Delta d_k = v_k \cdot dt$ ,  $\Delta d_{n,k} = v_{n,k} \cdot dt$ , and  $\Delta \theta_k = w_k \cdot dt$ . Additionally, for simplification, let  $c_{\theta_{k-1}}$  and  $s_{\theta_{k-1}}$  represent  $\cos(\theta_{k-1})$  and  $\sin(\theta_{k-1})$ , respectively.

$$\mathbf{x}(k+1) = \mathbf{x}(k) + dt \cdot f(\mathbf{x}(k), \mathbf{u}(k)) \quad (3.4)$$

$$\begin{aligned}
\mathbf{k}_1 &= f(\mathbf{x}(k), \mathbf{u}(k)), \\
\mathbf{k}_2 &= f\left(\mathbf{x}(k) + \frac{dt}{2} \cdot \mathbf{k}_1, \mathbf{u}(k) + \frac{dt}{2}\right) \\
\mathbf{k}_3 &= f\left(\mathbf{x}(k) + \frac{dt}{2} \cdot \mathbf{k}_2, \mathbf{u}(k) + \frac{dt}{2}\right) \\
\mathbf{k}_4 &= f(\mathbf{x}(k) + dt \cdot \mathbf{k}_3, \mathbf{u}(k) + dt) \\
\mathbf{x}(k+1) &= \mathbf{x}(k) + \frac{dt}{6} \cdot (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned} \tag{3.5}$$

if  $w = 0$  :

$$\begin{cases}
x_k = x_{k-1} + \Delta d_k \cdot c_{\theta_{k-1}} - \Delta d_{n,k} \cdot s_{\theta_{k-1}} \\
y_k = y_{k-1} + \Delta d_k \cdot s_{\theta_{k-1}} + \Delta d_{n,k} \cdot c_{\theta_{k-1}} \\
\theta_k = \theta_{k-1}
\end{cases}$$

else :

$$\begin{cases}
x_k = x_{k-1} + (\Delta d_k \cdot s_{\Delta\theta_k} + \Delta d_{n,k} \cdot (c_{\Delta\theta_k} - 1)) \cdot \frac{c_{\theta_{k-1} + \Delta\theta_k/2}}{\Delta\theta_k} \\
\quad - (\Delta d_k \cdot (1 - c_{\Delta\theta_k}) + \Delta d_{n,k} \cdot s_{\Delta\theta_k}) \cdot \frac{s_{\theta_{k-1} + \Delta\theta_k/2}}{\Delta\theta_k} \\
y_k = y_{k-1} + (\Delta d_k \cdot s_{\Delta\theta_k} + \Delta d_{n,k} \cdot (c_{\Delta\theta_k} - 1)) \cdot \frac{s_{\theta_{k-1} + \Delta\theta_k/2}}{\Delta\theta_k} \\
\quad + (\Delta d_k \cdot (1 - c_{\Delta\theta_k}) + \Delta d_{n,k} \cdot s_{\Delta\theta_k}) \cdot \frac{c_{\theta_{k-1} + \Delta\theta_k/2}}{\Delta\theta_k} \\
\theta_k = \theta_{k-1} + \Delta\theta_k
\end{cases} \tag{3.6}$$

On the other hand, since the optimization process of an NMPC requires the derivatives of the model, using a simple method like the Euler method is preferable as it significantly reduces computational time. Additionally, as shown in figures 3.3 and 3.4, reducing the discretization step ( $dt$ ) allows the Euler method to achieve a Root Mean Square Error (RMSE) smaller than  $0.005 \text{ m/s}$  for both  $x$  and  $y$ , and less than  $0.02 \text{ rad/s}$  for  $\theta$ . In these simulations, the Euler method with a discretization step of  $0.001$  seconds was used as the ground truth. Additionally, the input velocity functions used were  $v(t) = 0.5 + \sin(0.5t)$ ,  $v_n(t) = 0.5 + \sin(0.2t)$ , and  $w(t) = 0.5 + \sin(0.3t)$ .

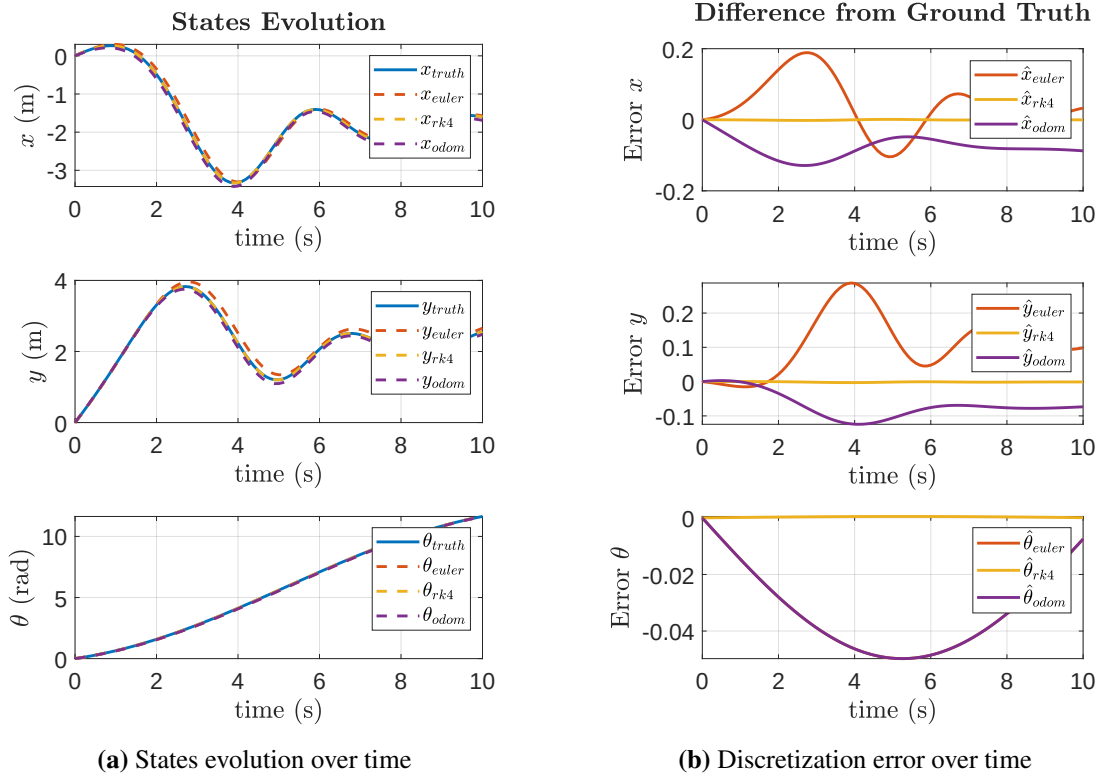


**Figure 3.3:** Comparison of differential equations discretization with a 0.1 seconds step size

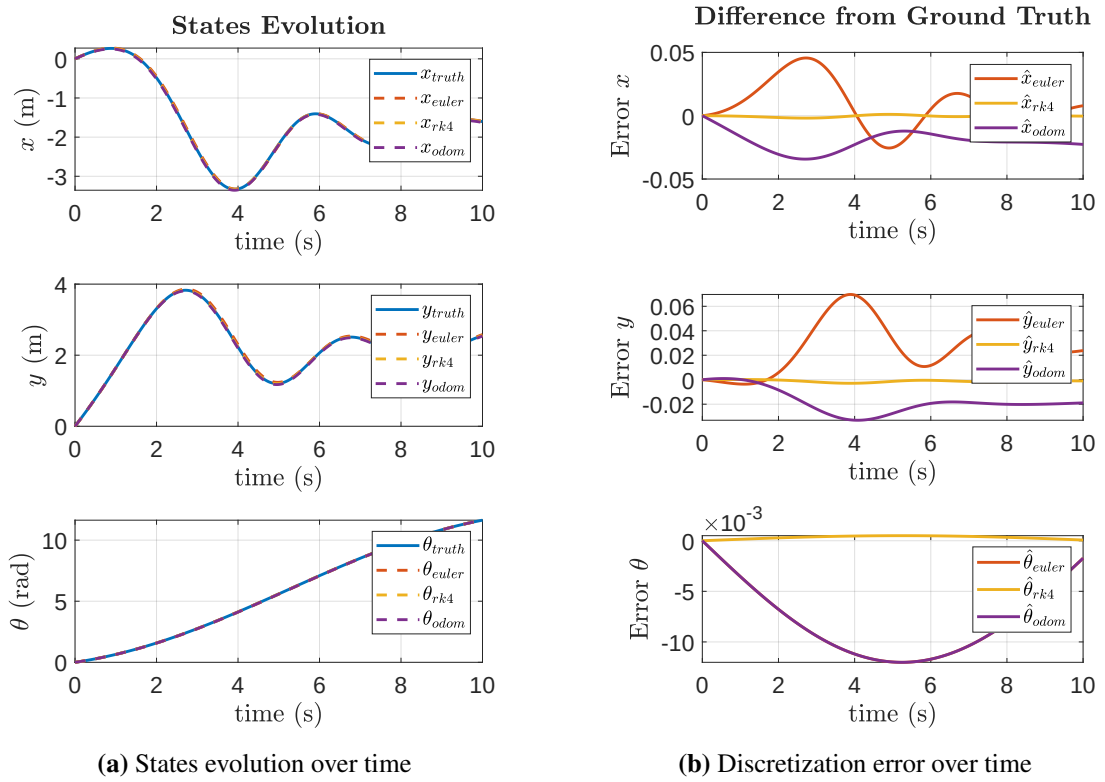


**Figure 3.4:** Comparison of differential equations discretization with a 0.025 seconds step size

### 3.2.2 Robot dynamics

Furthermore, the robot model can be improved by incorporating its dynamics, specifically accounting for the actuation forces during motion. Equation 3.7 thus describes the dynamic differential equations of the robot based on Newton's laws, where  $M$  denotes the mass of the robot and  $J$  represents its moment of inertia. Additionally,  $F_{B_v}$  and  $F_{B_{v_n}}$  denote viscous friction forces along  $v$  and  $v_n$ , respectively, while  $T_{B_w}$  represents the viscous friction torque along  $w$ , all of which are linearly proportional to their respective velocities (see equation 3.8). Moreover,  $F_{C_v}$  and  $F_{C_{v_n}}$  denote coulomb friction forces along  $v$  and  $v_n$ , and  $T_{C_w}$  represents the coulomb friction torque along  $w$ , both of which are proportional to the sign of their respective velocities (see equation 3.9). Finally,  $\sum F_v$  and  $\sum F_{v_n}$  correspond to the traction forces on the robot along  $v$  and  $v_n$ , and  $\sum T_w$  represents the rotational torque on the robot along  $w$ , that can be determined by combining the equations regarding the robot inverse kinematics (see equation 3.1) and the equations regarding the force and torque produced by the motor of each wheel (see equation 3.10). All of these equations were retrieved from the work by Oliveira et al. [53].

$$\begin{aligned} M \cdot \frac{dv(t)}{dt} &= \sum F_v(t) - F_{B_v}(t) - F_{C_v}(t) \\ M \cdot \frac{dv_n(t)}{dt} &= \sum F_{v_n}(t) - F_{B_{v_n}}(t) - F_{C_{v_n}}(t) \\ J \cdot \frac{dw(t)}{dt} &= \sum T_w(t) - T_{B_w}(t) - T_{C_w}(t) \end{aligned} \quad (3.7)$$

$$\begin{aligned} F_{B_v}(t) &= K_{B_v} \cdot v(t) \\ F_{B_{v_n}}(t) &= K_{B_{v_n}} \cdot v_n(t) \\ T_{B_w}(t) &= K_{B_w} \cdot w(t) \end{aligned} \quad (3.8)$$

$$\begin{aligned} F_{C_v}(t) &= K_{C_v} \cdot \frac{v(t)}{|v(t)|} \\ F_{C_{v_n}}(t) &= K_{C_{v_n}} \cdot \frac{v_n(t)}{|v_n(t)|} \\ T_{C_w}(t) &= K_{C_w} \cdot \frac{w(t)}{|w(t)|} \end{aligned} \quad (3.9)$$

Equation 3.10 describe the relationship, in a DC motor  $j$ , between the motor current ( $i_{m,j}$ ), motor angular velocity ( $\omega_{m,j}$ ), motor voltage ( $u_{m,j}$ ), motor torque ( $T_{m,j}$ ), and the resulting wheel force ( $f_j$ ) and torque ( $T_j$ ). Here,  $r$  denotes the wheel radius,  $n$  represents the gearbox reduction ratio,  $K_t$  denotes the motor torque constant,  $k_v$  represents the motor electromotive force constant, and  $L$  and  $R$  symbolizes the motor inductance and the motor resistance, respectively.

$$\begin{aligned}
f_j(t) &= \frac{T_j(t)}{r} \\
T_j(t) &= n \cdot K_t \cdot i_{m,j}(t) \\
u_{m,j}(t) &= L \cdot \frac{di_{m,j}(t)}{dt} + R \cdot i_{m,j}(t) + K_v \cdot \omega_{m,j}(t) \\
T_{m,j}(t) &= K_t \cdot i_{m,j}(t)
\end{aligned} \tag{3.10}$$

However, since the controller architecture includes low-level PI controllers for the wheel velocities, the robot's dynamic equations can be approximated by a first-order system, as shown in equation 3.11. Consequently, the measurement and estimation of all parameters from the previous equations can be replaced by only estimating the robot's time constants:  $\tau_v$ ,  $\tau_{v_n}$ , and  $\tau_w$ . Therefore, a straightforward experiment can be conducted with the real robot, as depicted in figure 3.5, to determine these time constants. This experiment involves applying a step input to each velocity component of the real robot, starting from a non-zero initial velocity. Then, using odometry, the actual velocities of the robot's wheels can be measured, and consequently, the robot's velocities can be determined. Finally, the time constants may be determined through a first-order approximation employing the least squares method, resulting in an RMSE less than  $0.005 \text{ m/s}$  for both  $v$  and  $v_n$ , and less than  $0.02 \text{ rad/s}$  for  $w$ .

$$\begin{aligned}
\frac{dv(t)}{dt} &= -\frac{1}{\tau_v} \cdot v(t) + \frac{1}{\tau_v} \cdot v_{ref}(t) \\
\frac{dv_n(t)}{dt} &= -\frac{1}{\tau_{v_n}} \cdot v_n(t) + \frac{1}{\tau_{v_n}} \cdot v_{n,ref}(t) \\
\frac{dw(t)}{dt} &= -\frac{1}{\tau_w} \cdot w(t) + \frac{1}{\tau_w} \cdot w_{ref}(t)
\end{aligned} \tag{3.11}$$

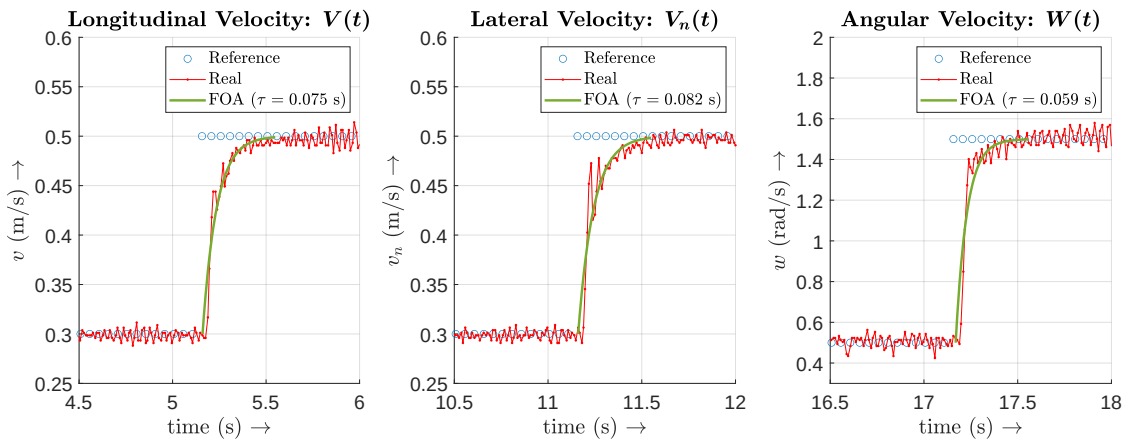


**Figure 3.5:** First Order Approximation (FOA) of the robot velocity dynamics for the robot with codename "Adam"; the symbol  $\tau$  denotes the robot's time constants

Furthermore, by utilizing both differential equations 3.3 and 3.11, a dynamic model for the

omnidirectional robot can be derived in the form of the state equation  $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ . In this model, the system input is now given by  $\mathbf{u} = [v_{ref}, v_{n,ref}, w_{ref}]^\top$  and the system state is represented as  $\mathbf{x} = [x, y, \theta, v, v_n, w]^\top$ , now including both the robot's pose and real velocities.

### 3.3 Optimization Problem Formulation

In an MPC framework, the solution of an Optimal Control Problem (OCP) is necessary. There are two main approaches to solving such an OCP numerically: direct and indirect. A problem formulation is either first discretized and then optimized (direct method) or first optimized and then discretized (indirect method). In robotics, however, the direct methods are much more common as they are simpler and easier to solve, as already mentioned in section 2.1.

#### 3.3.1 Optimal control problem

Therefore, using a direct method, a common optimization problem for trajectory tracking is described in equation 3.12. The objective is to find the set of system inputs  $\mathbf{u}$  that minimizes the cost function  $J(\cdot)$  while respecting the system dynamics, the initial state  $\mathbf{x}_0$ , and the constraints on velocity and acceleration ( $\mathbf{u}_{max}, \Delta\mathbf{u}_{max}$ ). The cost function penalizes the position and orientation errors relative to the reference trajectory  $\mathbf{x}_{ref}$  over a discrete prediction horizon, denoted as  $N_p$ , and weighs these penalties using a diagonal matrix  $Q$ . Additionally, the cost function also penalizes variations in the system input  $\Delta\mathbf{u}$  over a discrete control horizon  $N_c$ , with penalties also weighted by a diagonal matrix  $R$ . The penalty of  $\Delta\mathbf{u}$  ( $\mathbf{u}(k+1) - \mathbf{u}(k)$ ) instead of just  $\mathbf{u}$ , results in smoother transitions and a smaller steady-state error.

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & J(\mathbf{x}_0, \mathbf{x}_{ref}(\cdot), \mathbf{u}(\cdot)) = \sum_{k=1}^{N_p} \|\mathbf{x}(k) - \mathbf{x}_{ref}(k)\|_Q^2 + \sum_{k=0}^{N_c-1} \|\Delta\mathbf{u}(k)\|_R^2 \\ \text{s.t.} \quad & \mathbf{x}(k+1) = h(\mathbf{x}(k), \mathbf{u}(k)), \quad \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{u}_{min} \leq \mathbf{u}(\cdot) \leq \mathbf{u}_{max} \\ & \Delta\mathbf{u}_{min} \leq \Delta\mathbf{u}(\cdot) \leq \Delta\mathbf{u}_{max} \end{aligned} \tag{3.12}$$

#### 3.3.2 Nonlinear programming problem

However, solving the OCP while considering the inequality constraints is complex and typically requires techniques like Sequential Quadratic Programming (SQP) or interior-point methods [18]. Alternatively, the OCP can be simplified by first addressing the velocity and acceleration inequality constraints outside the optimization process, which enables the use of simpler and more efficient optimization solvers, such as gradient-based methods. Additionally, by utilizing a single shooting approach [17] for the system dynamics, the Nonlinear Programming (NLP) problem in equation 3.13 can be formulated. In the single shooting strategy, only the system inputs are treated as optimization variables. Moreover, the states across the entire prediction horizon,  $N_p$ , are determined by forward propagating,  $\phi(k, \mathbf{x}_0, \mathbf{u}(\cdot))$ , the initial state,  $\mathbf{x}_0$ , along with the system inputs,

$\mathbf{u}(\cdot)$ , as demonstrated in equation 3.14. An inherent drawback of this method is its tendency to amplify nonlinearity with each iteration, potentially slowing convergence, particularly for longer prediction horizons. On the other hand, for shorter prediction horizons, this method can be effective since the number of optimization variables is smaller.

$$\min_{\mathbf{u}(\cdot)} J(\mathbf{x}_0, \mathbf{x}_{ref}(\cdot), \mathbf{u}(\cdot)) = \sum_{k=1}^{N_p} \|\phi(k, \mathbf{x}_0, \mathbf{u}(\cdot)) - \mathbf{x}_{ref}(k)\|_Q^2 + \sum_{k=0}^{N_c-1} \|\Delta \mathbf{u}(k)\|_R^2 \quad (3.13)$$

$$\begin{aligned} \mathbf{x}(0) &= \phi(0, \mathbf{x}_0, \mathbf{u}(\cdot)) = \mathbf{x}_0 \\ \mathbf{x}(1) &= \phi(1, \mathbf{x}_0, \mathbf{u}(\cdot)) = h(\mathbf{x}_0, \mathbf{u}(0)) \\ \mathbf{x}(2) &= \phi(2, \mathbf{x}_0, \mathbf{u}(\cdot)) = h(h(\mathbf{x}_0, \mathbf{u}(0)), \mathbf{u}(1)) \\ &\vdots \\ \mathbf{x}(k) &= \phi(k, \mathbf{x}_0, \mathbf{u}(\cdot)) = h(h(\dots), \mathbf{u}(k-1)) \end{aligned} \quad (3.14)$$

Lastly, the function  $h(\mathbf{x}(k), \mathbf{u}(k))$  denotes the discretization of the ordinary differential equations governing the system dynamics. The Euler method, elaborated in equation 3.4, was employed for this purpose in the thesis.

### 3.3.3 Gradient-based methods

In order to solve the NLP problem defined in equation 3.13, a gradient-based method is utilized. The optimal solution  $\mathbf{u}_{sol}$ , which includes the reference robot velocities ( $v_{ref}$ ,  $v_{n,ref}$ , and  $w_{ref}$ ), is determined iteratively by updating the optimization variables  $\mathbf{u}(\cdot) = [\mathbf{u}(1), \mathbf{u}(2), \dots, \mathbf{u}(N_c)]$  using the gradient of the cost function with respect to  $\mathbf{u}$ , denoted as  $\nabla_{\mathbf{u}} J(\mathbf{x}_0, \mathbf{x}_{ref}(\cdot), \mathbf{u}_{i+1}(\cdot))$ , as shown in algorithm 1. The optimal solution corresponds to the optimal system input within the first control horizon  $\mathbf{u}(1)$ . This algorithm requires the reference trajectory  $\mathbf{x}_{ref}(\cdot)$  for the entire prediction horizon  $N_p$  and the initial state of the system  $\mathbf{x}_0$  to compute the gradient. The optimization process terminates either when the norm of the cost function gradient is sufficiently small, indicating proximity to a local minimum, or when the maximum number of iterations is reached. While limiting the maximum number of iterations does not guarantee an acceptable solution, it helps maintain the control cycle time. Nevertheless, since the optimization is performed at each control cycle and initialized with the solution from the previous cycle, the new solution is typically very close to the previous one, allowing convergence to a point very close to the solution in just a few iterations.

Additionally, to solve the NLP, various gradient methods were evaluated, including Rprop [54], Steepest Descent, and the Fletcher–Reeves [55] and Polak–Ribière [56] conjugate gradient methods. A detailed description of each evaluated algorithm is provided in appendix A. The appendix also presents the results, which show that the Fletcher–Reeves conjugate gradient method performed the best, typically converging within 4 to 8 iterations. The results also helped define the terminal conditions for algorithm 1, concluding that  $\varepsilon = 1e-3$  is sufficient for achieving a very

close solution. A very close solution is achieved when, after reaching the terminal condition  $\varepsilon$  (minimum absolute value for the cost function gradient), subsequent iterations of the optimization process do not significantly improve the solution. Additionally, for a controller running at 40 Hz, a maximum of 20 iterations is appropriate for the LattePanda 3 Delta<sup>2</sup> used in the robot with the codename “Adam”. The LattePanda 3 Delta used is powered by the Intel Celeron N5105 processor, which operates at a frequency range of 2.0-2.9 GHz.

---

**Algorithm 1:** Optimization cycle

---

**Input:**  $\mathbf{x}_0, \mathbf{x}_{ref}$   
**Data:**  $\varepsilon, itr_{max}$   
**Output:**  $\mathbf{u}_{sol}$

```

1 begin
2    $i \leftarrow 0$ 
3   repeat
4      $\nabla_{\mathbf{u}} J(\mathbf{x}_0, \mathbf{x}_{ref}(\cdot), \mathbf{u}_i(\cdot)) \leftarrow$  cost function gradient computation
5      $\mathbf{u}_{i+1}(\cdot) \leftarrow$  determined by a gradient descent method
6      $i \leftarrow i + 1$ 
7   until  $\|\nabla_{\mathbf{u}} J(\mathbf{x}_0, \mathbf{x}_{ref}(\cdot), \mathbf{u}_{i+1}(\cdot))\| < \varepsilon$  or  $i \geq itr_{max}$ 
8    $\mathbf{u}_{sol} \leftarrow \mathbf{u}_i(1)$ 
9 end
```

---

### 3.4 Velocity and Acceleration Constraints

As previously mentioned, moving the velocity and acceleration constraints outside the optimization process reduces its complexity and, consequently, decreases the required processing time. Therefore each of the constraints is handled in a different way.

#### 3.4.1 Velocity scaling

The velocity inequality constraints are managed by scaling the wheel’s reference linear velocities ( $v_{i,ref} \mid i \in \{1 \dots 4\}$ ) to ensure they do not surpass a maximum velocity value  $v_{max}$ . Therefore, the scaling factor is derived from equation 3.15. Subsequently, each reference motor wheel angular velocity ( $\phi_{i,ref} \mid i \in \{1 \dots 4\}$ ) is computed using the scaling factor and the transformation between angular and linear velocities (see equation 3.2), resulting in equation 3.16.

$$\sigma = \min \left( \frac{v_{max}}{\max(|v_{1,ref}|, |v_{2,ref}|, |v_{3,ref}|, |v_{4,ref}|)}, 1.0 \right) \quad (3.15)$$

$$\phi_{i,ref} = \frac{2n}{D_i} \cdot \sigma \cdot v_{i,ref}, \quad i \in \{1 \dots 4\} \quad (3.16)$$

---

<sup>2</sup><https://www.lattepanda.com/lattepanda-3-delta>

This scaling strategy ensures that the robot adheres to the trajectory specified by the NMPC, albeit at a reduced speed, in cases where any of the wheels' reference velocities exceed their maximum allowable limits. By uniformly scaling all velocities based on the wheel with the highest reference velocity, the proportionality between the individual wheel velocities is preserved. Consequently, the resultant linear and angular velocities ( $v$ ,  $v_n$ , and  $w$ ) maintain their relative ratios, and the direction of the velocity vector remains unchanged. This preservation of proportionality and direction ensures that the robot continues to follow the intended trajectory without any deformation of its motion.

### 3.4.2 Motion velocity profile

On the other hand, the acceleration inequality constraints are managed at the robot level, particularly during transitions in linear velocity. Since the nonlinear MPC requires a discretized trajectory at each cycle, this set of reference points (see figure 3.6) must be spaced proportionally to the robot's linear velocity and the discretization step size ( $dt$ ). Normally, if the acceleration constraints are not considered during path discretization, the points will be evenly spaced along the path based on a constant steady-state linear velocity. Additionally, at the end of the path, the final point is often repeated to signify the path's conclusion. As a result, critical points typically occur at the beginning and end of the trajectory, where the distance between reference points varies from 0 to a constant value determined by the steady-state linear velocity. This variance creates a velocity discontinuity, which can be partially mitigated by minimizing the variance of the system input ( $\Delta \mathbf{u}$ ) in the NMPC cost function. However, these discontinuities are not entirely eliminated and require precise tuning of the weights in matrix  $R$  within the cost function (see equation 3.13). Furthermore, different steady-state linear velocities will necessitate different tuning of these weights, possibly leading to overshoots at the end of the trajectory.

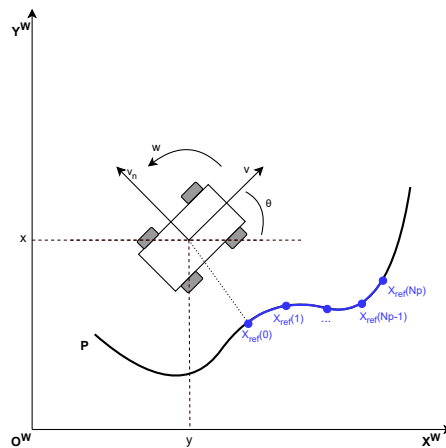


**Figure 3.6:** Reference trajectory discretization

In this thesis, these critical points are mitigated by implementing a motion velocity profile such as the trapezoidal profile (see figure 3.7a) or an s-curve profile (see figure 3.7b). The primary

distinction lies in the presence of instantaneous changes in acceleration in the trapezoidal profile versus a linear variation in the s-curve profile, defined by a maximum jerk.

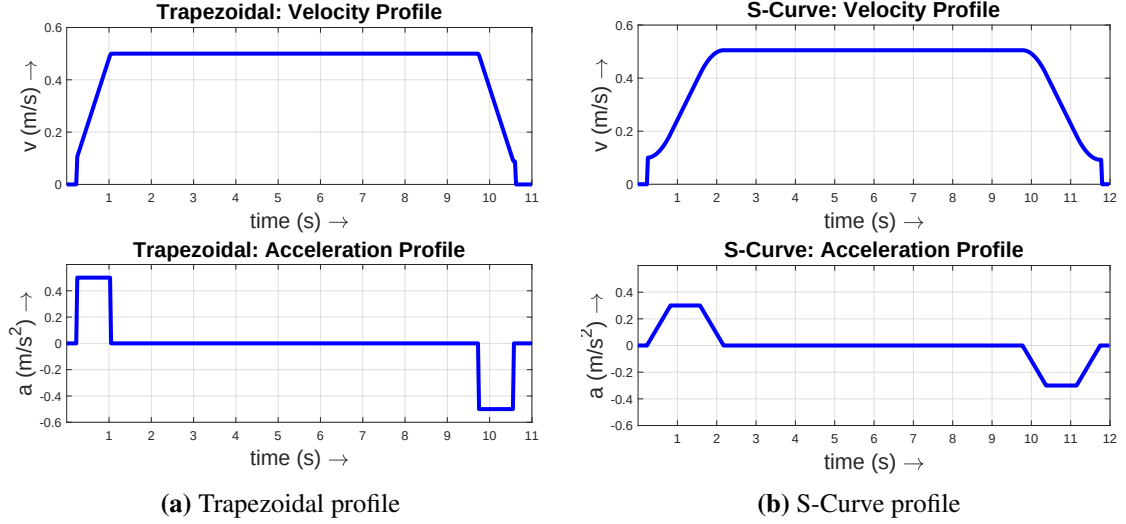


**Figure 3.7:** Comparison of the motion profiles

The motion velocity profiles are derived from the one-dimensional equations of motion, as shown in equation 3.17. In these equations,  $x_0$ ,  $v_0$ , and  $a_0$  denote the initial position, velocity, and acceleration, respectively, while  $j$  represents the jerk.

$$\begin{aligned}
 x(t) &= x_0 + v_0 \cdot t + \frac{1}{2} \cdot a_0 \cdot t^2 + \frac{1}{6} \cdot j \cdot t^3 \\
 v(t) &= v_0 + a_0 \cdot t + \frac{1}{2} \cdot j \cdot t^2 \\
 a(t) &= a_0 + j \cdot t
 \end{aligned} \tag{3.17}$$

Appendix B presents a possible implementation of the motion velocity profiles using finite state machines. Additionally, the appendix also explains all the equations related to the transitions between different states, given the desired steady-state linear velocity and the distance to the end of the trajectory. It is important to note that since the controller operates at a specific frequency and is subject to disturbances, identifying the exact cycle that the final transition must start to reach the final position at 0 m/s is challenging. Furthermore, the robot motors' dead zone makes it difficult for the robot to move at very low speeds. Consequently, the motion velocity profiles account for a minimum velocity outside the dead zone to facilitate transition identification and improve the accuracy at the final point.

### 3.5 Simulation Results

In order to validate the proposed control architecture, several simulations were initially performed. The trajectory controller, developed in C++, was integrated into the navigation stack of the mobile robot team from the Centre for Robotics in Industry and Intelligent Systems (CRIIS) at INESC TEC, using the ROS framework, specifically the Noetic distribution<sup>3</sup>. The Gazebo simulator<sup>4</sup>, an open-source 2D/3D robotics simulator with ROS interfaces, was used for the simulations. However, Gazebo does not support the simulation of omnidirectional robot wheels. To address this, the “planar move plugin”<sup>5</sup> was employed to move the robot in a 2D plane based on the robot’s reference velocities  $v_{ref}$ ,  $v_{n,ref}$ , and  $w_{ref}$ . Additionally, to incorporate the system dynamics, particularly the dynamics of the PI wheel controllers, a first-order dynamic system for each wheel was used (see equation 3.18) instead of directly applying the reference velocities produced by the NMPC. This system shared the same time constant ( $\tau$ ) as the PI controllers. The dynamic system was discretized using the 4th-order Runge-Kutta method (see equation 3.5) with a time step ( $dt$ ) of 0.001 seconds. Thus, the robot simulation was performed at 100 Hz using the “planar move plugin”, accounting for the new reference velocities ( $v_{new}$ ,  $v_{n,new}$ , and  $w_{new}$ ), calculated using the inverse kinematic matrix in equation 3.19 and the wheel velocities ( $v_i \mid i \in \{1 \dots 4\}$ ) determined by the first-order dynamic system.

$$\frac{dv_i(t)}{dt} = -\frac{1}{\tau} \cdot v_i(t) + \frac{1}{\tau} \cdot v_{i,ref}(t), \quad i \in \{1 \dots 4\} \quad (3.18)$$

$$\begin{bmatrix} v_{new} \\ v_{n,new} \\ w_{new} \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{2 \cdot (l_1 + l_2)} & -\frac{1}{2 \cdot (l_1 + l_2)} & -\frac{1}{2 \cdot (l_1 + l_2)} & -\frac{1}{2 \cdot (l_1 + l_2)} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \quad (3.19)$$

In the following sections, the proposed controller will be compared with a PD controller for a pulse trajectory with orientation variation defined with the line segments presented in the table 3.1.

**Table 3.1:** Pulse trajectory

| Index | $x$ (m) | $y$ (m) | $\theta$ (rad) |
|-------|---------|---------|----------------|
| 1     | 0       | 0       | 0              |
| 2     | 2       | 0       | $\pi/4$        |
| 3     | 2       | 2       | $\pi/2$        |
| 4     | 4       | 2       | $3\pi/4$       |
| 5     | 4       | 0       | $\pi$          |
| 6     | 6       | 0       | $5\pi/4$       |

<sup>3</sup><http://wiki.ros.org/noetic>

<sup>4</sup><https://gazebo.org/home>

<sup>5</sup>[https://github.com/Boeing/gazebo\\_planar\\_move\\_plugin](https://github.com/Boeing/gazebo_planar_move_plugin)

Moreover, in the analysis of the results, both the maximum (max) and the average (avg) values of the following three quality measures are considered. Additionally, the position errors ( $\epsilon_d$ ) and orientation errors ( $\epsilon_\theta$ ) are determined with respect to the closest point on the trajectory at each control cycle.

- $\epsilon_d$ : distance error over time;
- $\epsilon_\theta$ : absolute orientation error over time;
- $t_{proc}$ : control cycle processing time.

Finally, simulations for both the proposed controller and the PD controller were conducted using the specified parameters. These simulations were performed on a laptop with an Intel Core i7-7700HQ processor running at 2.8 GHz, offering performance comparable to the processor used in the actual robot.

- Simulation conditions:
  - Processor: Intel Core i7-7700HQ 2.80 GHz
  - Trajectory tracking controller frequency: 40 Hz
  - Simulation / Localization frequency: 100 Hz
  - Robot wheels time constant ( $\tau$ ): 0.1 s or 0.3 s
- Proposed controller:
  - Cost function parameters:  $dt = 0.025$  s,  $N_p = 20$ ,  $N_c = 1$  to 4,  
 $Q = \text{diag}\{1, 1, 1, 0, 0, 0\}$ ,  $R = \text{diag}\{1, 1, 1\}$
  - Optimization solver: Fletcher–Reeves
  - Optimization parameters:  $\epsilon = 0.001$ ,  $itr_{max} = 20$
  - Dynamic model parameters:  $\tau_v = 0.1$  s or 0.3 s,  $\tau_{v_n} = 0.1$  s or 0.3 s,  $\tau_w = 0.1$  s or 0.3 s
  - Motion velocity profile (S-Curve):  $v_{max} = 0.6$  m/s or 1.2 m/s,  $a_{max} = 2.0$  m/s<sup>2</sup>,  $j_{max} = 0.8$  m/s<sup>3</sup>,  $v_{min} = 0.1$  m/s
  - Velocity scaling:  $v_{max} = 0.6$  m/s or 1.2 m/s
- PD controller:
  - Proportional gains:  $K_p = [1.5, 1.5, 9.0]$
  - Derivative gains:  $K_d = [0.1, 0.1, 0.3]$

Additionally, the optimization parameters were determined through a set of tests, and the most relevant results are presented in appendix A. The time interval ( $dt$ ) was set equivalent to the inverse of the controller frequency, and the prediction horizon was fixed at 0.5 seconds, resulting in a discrete-time prediction horizon ( $N_p = 20$ ). From the tests conducted, a prediction horizon of 0.5 seconds was found to be optimal. Furthermore, in the upcoming simulations, the controllers will be evaluated using different steady-state velocities (0.5 m/s and 1.0 m/s) as well as varying robot time constants ( $\tau = 0.1$  s and  $\tau = 0.3$  s). Moreover, multiple simulations with the proposed controller were conducted for each combination of steady-state velocity and robot time constants, varying the control horizon ( $N_c$ ) from 1 to 4, for both the kinematic and dynamic models. Additionally, the parameters of the PD controller were fine-tuned through manual search to achieve optimal calibration, aiming to minimize overshoot at trajectory corners.

### 3.5.1 PD-only controller

First, a set of simulations were conducted for a steady-state linear velocity of  $0.5 \text{ m/s}$  and robot time constants of  $0.1 \text{ s}$ . These parameters were chosen based on the real robot, codenamed “Adam” (see figure 3.2a), which is equipped with motors allowing a maximum velocity of  $0.6 \text{ m/s}$  for each wheel. Additionally, the estimated time constants for the robot’s velocities are  $\tau_v = 0.075 \text{ s}$ ,  $\tau_{v_n} = 0.082 \text{ s}$ , and  $\tau_w = 0.059 \text{ s}$  (see figure 3.5).

The results of this set of simulations are shown in figures 3.8 and 3.9 and summarized in table 3.2. From table 3.2, it is evident that the PD controller exhibited larger errors in both position and orientation compared to the proposed NMPC controller. This outcome is expected since the PD controller lacks predictive capabilities, leading to overshoots at each corner, as depicted in figure 3.8. However, the PD controller is simpler and does not require an optimization process, resulting in shorter processing times. Regarding the proposed controller, the kinematic and dynamic models produced very similar errors, with the kinematic model performing slightly better. These results indicate that when the time constants are very small ( $0.1 \text{ s}$ ), using a first-order dynamic model does not enhance controller performance. Additionally, a longer control horizon led to smaller errors in both position and orientation, though this came at the cost of increased processing times. Conversely, the variation in position and orientation errors was not significant. For instance, the maximum distance error over time ( $\epsilon_{max,d}$ ) decreased from  $0.049 \text{ m}$  to  $0.034 \text{ m}$  in the kinematic model and from  $0.053 \text{ m}$  to  $0.045 \text{ m}$  in the dynamic model.

Given that the results for the proposed controller were very similar across different settings, only the dynamic model with a control horizon of 4 is illustrated in figure 3.8. Additionally, figure 3.9 depicts the evolution of the linear and angular robot velocities (see figure 3.9a) as well as the position and orientation errors (see figure 3.9b) for the proposed controller with this settings.

**Table 3.2:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

| Metric                                | PD          | NMPC Kinematic Model |              |           |              | NMPC Dynamic Model |           |           |              |
|---------------------------------------|-------------|----------------------|--------------|-----------|--------------|--------------------|-----------|-----------|--------------|
|                                       |             | $N_c = 1$            | $N_c = 2$    | $N_c = 3$ | $N_c = 4$    | $N_c = 1$          | $N_c = 2$ | $N_c = 3$ | $N_c = 4$    |
| $\epsilon_{avg,d} \text{ (m)}$        | 0.011       | 0.005                | 0.004        | 0.004     | <u>0.003</u> | 0.007              | 0.006     | 0.005     | 0.004        |
| $\epsilon_{max,d} \text{ (m)}$        | 0.067       | 0.049                | 0.045        | 0.038     | <u>0.034</u> | 0.053              | 0.053     | 0.048     | 0.045        |
| $\epsilon_{avg,\theta} \text{ (rad)}$ | 0.045       | 0.027                | 0.026        | 0.024     | <u>0.022</u> | 0.029              | 0.029     | 0.028     | 0.027        |
| $\epsilon_{max,\theta} \text{ (rad)}$ | 0.112       | 0.085                | <u>0.081</u> | 0.083     | <u>0.081</u> | 0.084              | 0.083     | 0.083     | <u>0.081</u> |
| $t_{avg,proc} \text{ (ms)}$           | <u>0.35</u> | 0.89                 | 0.98         | 1.18      | 1.59         | 0.94               | 1.08      | 1.24      | 1.39         |
| $t_{max,proc} \text{ (ms)}$           | <u>1.10</u> | 2.29                 | 2.25         | 5.01      | 5.49         | 2.05               | 2.94      | 3.27      | 4.10         |

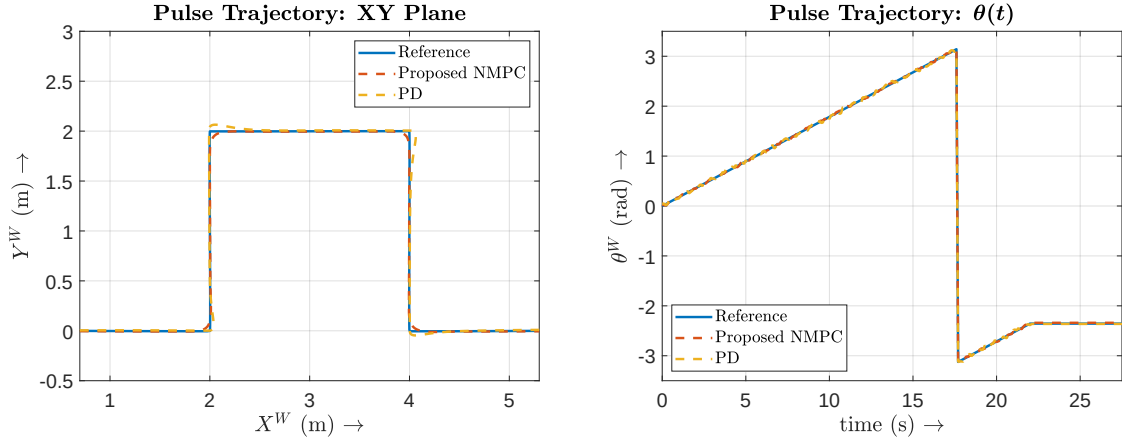


**Figure 3.8:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

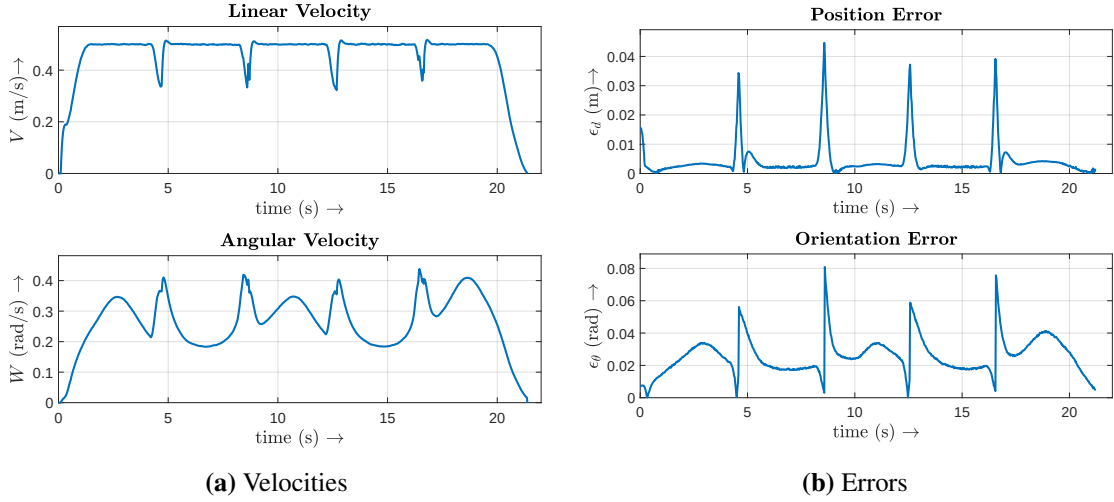


**Figure 3.9:** Evolution of the velocities and errors over time of the proposed NMPC for a pulse trajectory: Simulation; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

In figure 3.9b, both the position and orientation errors show four peaks throughout the trajectory. These peaks correspond to the four corners of the trajectory, where there is a discontinuity in the curvature. Additionally, in figure 3.9a, the linear velocity remains at the steady-state value for most of the trajectory, except at the beginning and end, where it follows an s-curve velocity profile, and at each corner, where the minimization of velocity variation in the cost function produces a brief deceleration.

Furthermore, to evaluate the performance of the controller at higher velocities, a second set of simulations were conducted for a steady-state linear velocity of  $1.0 \text{ m/s}$ , keeping the same robot time constants and changing the robot maximum velocity from  $0.6 \text{ m/s}$  to  $1.2 \text{ m/s}$ . The results from this set of simulations are illustrated in the figure 3.10 and summarized in the table 3.3.

The analysis from the previous results remains consistent, with the PD controller producing higher errors but achieving better processing times, as shown in table 3.3. Additionally, the differences between the kinematic and dynamic models are still minimal, indicating that the impact of the dynamic model on the controller is more related to the time constants than to the robot velocity. However, with an increased steady-state velocity, the performance disparity between the PD controller and the proposed NMPC controller was more evident. This is illustrated in figure 3.10, where the PD controller's positional overshoot is more clear, resulting in a  $\epsilon_{max,d} = 0.166\text{ m}$ , compared to the proposed controller's  $\epsilon_{max,d} = 0.100\text{ m}$ , which performs a cut inside in each corner of the trajectory. Regarding orientation errors, the proposed controller still outperformed the PD controller, although the difference is smaller because the reference orientation changes linearly, lacking discontinuities that a predictive controller can exploit. Finally, the processing time for the proposed controller remains proportional to the number of optimization variables. Thus, the processing time increases with the control horizon and is higher for the dynamic model, which contains six states ( $\mathbf{x} = [x, y, \theta, v, v_n, w]^\top$ ), compared to only three in the kinematic model ( $\mathbf{x} = [x, y, \theta]^\top$ ).

**Table 3.3:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of  $1.0\text{ m/s}$ ; Time constants of  $0.1\text{ s}$

| Metric                        | PD          | NMPC Kinematic Model |           |              |              | NMPC Dynamic Model |           |           |           |
|-------------------------------|-------------|----------------------|-----------|--------------|--------------|--------------------|-----------|-----------|-----------|
|                               |             | $N_c = 1$            | $N_c = 2$ | $N_c = 3$    | $N_c = 4$    | $N_c = 1$          | $N_c = 2$ | $N_c = 3$ | $N_c = 4$ |
| $\epsilon_{avg,d} (m)$        | 0.040       | 0.019                | 0.014     | <u>0.012</u> | <u>0.012</u> | 0.024              | 0.021     | 0.018     | 0.016     |
| $\epsilon_{max,d} (m)$        | 0.166       | 0.107                | 0.091     | 0.084        | <u>0.076</u> | 0.120              | 0.107     | 0.096     | 0.100     |
| $\epsilon_{avg,\theta} (rad)$ | 0.085       | 0.051                | 0.049     | 0.045        | <u>0.042</u> | 0.056              | 0.055     | 0.053     | 0.051     |
| $\epsilon_{max,\theta} (rad)$ | 0.164       | 0.125                | 0.128     | <u>0.115</u> | 0.116        | 0.129              | 0.132     | 0.129     | 0.125     |
| $t_{avg,proc} (ms)$           | <u>0.35</u> | 0.90                 | 1.09      | 1.37         | 2.13         | 1.03               | 1.20      | 1.43      | 1.66      |
| $t_{max,proc} (ms)$           | <u>0.99</u> | 1.66                 | 2.49      | 4.03         | 4.87         | 3.02               | 2.52      | 3.45      | 4.82      |

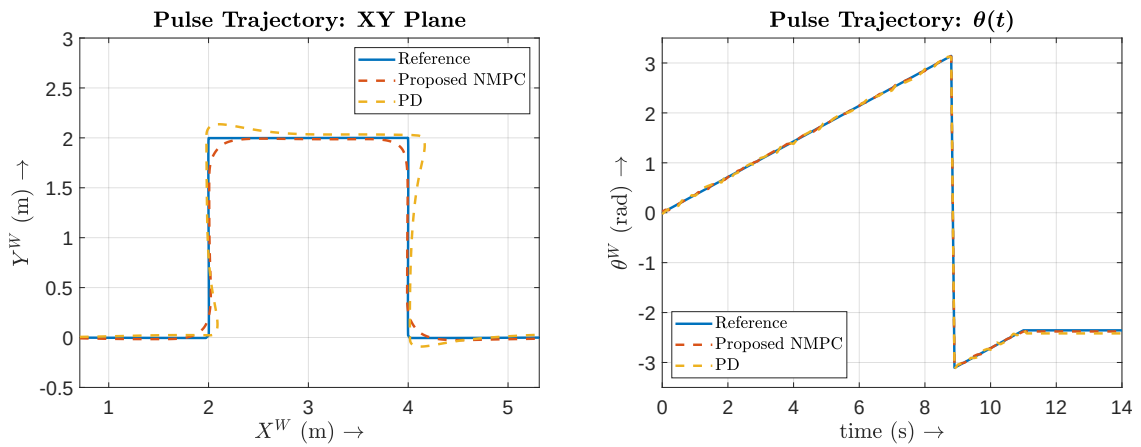


**Figure 3.10:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of  $1.0\text{ m/s}$ ; Time constants of  $0.1\text{ s}$

Finally, a third set of simulations was performed with a steady-state linear velocity of  $0.5 \text{ m/s}$  and robot time constants of  $0.3 \text{ s}$ . These simulations aim to evaluate the performance of the proposed controller for robots with higher time constants and, consequently, slower dynamics, such as heavier and larger robots. The results from this set of simulations are illustrated in figure 3.11 and summarized in table 3.4.

**Table 3.4:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.3 \text{ s}$

| Metric                                | PD          | NMPC Kinematic Model |           |           |              | NMPC Dynamic Model |           |              |              |
|---------------------------------------|-------------|----------------------|-----------|-----------|--------------|--------------------|-----------|--------------|--------------|
|                                       |             | $N_c = 1$            | $N_c = 2$ | $N_c = 3$ | $N_c = 4$    | $N_c = 1$          | $N_c = 2$ | $N_c = 3$    | $N_c = 4$    |
| $\epsilon_{avg,d} \text{ (m)}$        | 0.029       | 0.010                | 0.012     | 0.014     | 0.015        | 0.004              | 0.004     | 0.004        | <u>0.003</u> |
| $\epsilon_{max,d} \text{ (m)}$        | 0.135       | 0.052                | 0.058     | 0.065     | 0.071        | 0.042              | 0.043     | 0.040        | <u>0.033</u> |
| $\epsilon_{avg,\theta} \text{ (rad)}$ | 0.043       | 0.024                | 0.023     | 0.022     | <u>0.021</u> | 0.029              | 0.028     | 0.028        | 0.027        |
| $\epsilon_{max,\theta} \text{ (rad)}$ | 0.120       | 0.090                | 0.088     | 0.093     | 0.091        | 0.089              | 0.086     | <u>0.084</u> | 0.086        |
| $t_{avg,proc} \text{ (ms)}$           | <u>0.43</u> | 1.04                 | 1.23      | 1.50      | 2.08         | 1.18               | 1.47      | 1.79         | 1.99         |
| $t_{max,proc} \text{ (ms)}$           | <u>1.02</u> | 2.09                 | 2.77      | 3.58      | 4.44         | 2.94               | 3.39      | 3.58         | 4.57         |

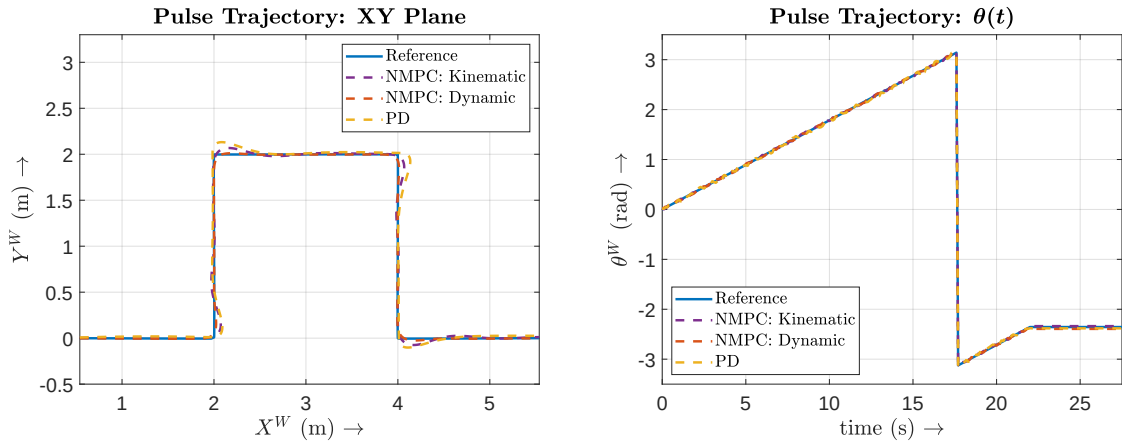


**Figure 3.11:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Simulation; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.3 \text{ s}$

In this new set of results, the proposed controller with a dynamic model outperformed both the PD controller and the proposed controller with the kinematic model. It had an  $\epsilon_{max,d}$  that was 2 times smaller than the kinematic model and 4 times smaller than the PD controller, as indicated in table 3.4. This difference is illustrated in figure 3.11, where the PD controller is compared with the proposed controller using a control horizon of 4 for both the kinematic and dynamic models. These results confirm that using a first-order dynamic model is justified even at lower velocities ( $0.5 \text{ m/s}$ ) when the time constants are high ( $\geq 0.3 \text{ s}$ ). Moreover, although the kinematic model produces an overshoot, this overshoot is still smaller than that of the PD controller, mainly because it is a predictive controller that can still anticipate the corners, unlike the PD controller. Finally, regarding the processing time of all the controllers, even though the proposed NMPC controller has

higher processing times than the PD controller, the difference is minimal. The controller operates at 40 Hz, meaning it only requires at most 20% of the total control cycle time ( $1/40 = 25 \text{ ms}$ ).

### 3.5.2 Motion velocity profile

In order to evaluate the influence of the motion velocity profile on the proposed controller, two simulations were performed: one using an s-curve motion velocity profile and another without it. For the simulation without the s-curve profile, the path was discretized with equal distances between reference positions proportional to the steady-state velocity, and the final point was repeated at the end of the path. In these simulations, the proposed controller with the dynamic model and a control horizon of 4 was used, considering a steady-state linear velocity of  $0.5 \text{ m/s}$  and robot time constants of  $0.1 \text{ s}$ .

A comparison of the reference linear and angular velocities is presented in figure 3.12, where the difference at the beginning and end of the motion is evident. When using the s-curve motion profile (see figure 3.12a), the velocity variation is smooth, adhering to the motion profile and resulting in a maximum slope equivalent to the maximum acceleration defined in the motion profile ( $a_{max}$ ). In contrast, when the s-curve motion profile is not used (see figure 3.12b), the velocity variation is abrupt, adjustable only by the weights on the velocity variation in the cost function ( $R$ ). By increasing the weights in matrix  $R$ , the maximum slope of the velocities will be reduced; however, it remains impossible to precisely define the desired maximum acceleration.

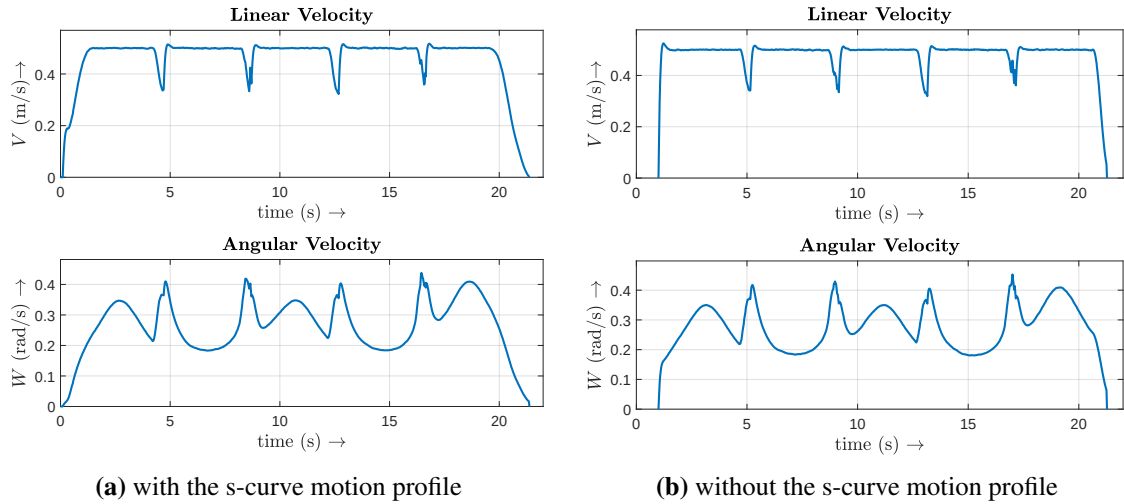


**Figure 3.12:** Evolution of the velocities over time of the proposed NMPC for a pulse trajectory: Simulation; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

Moreover, as the optimization process does not account for a maximum velocity, an initial overshoot in the linear reference velocity is shown in figure 3.12b. This overshoot occurs due to the first-order dynamic model, which implies that for a given reference velocity, the actual velocity will reach 63.2% of that value in  $\tau$  seconds, where  $\tau$  represents the time constant. Consequently, the optimal solution of the optimization process results in a higher reference velocity to reach the

desired velocity in the shortest possible time. However, this behavior is undesirable as it leads to even higher slopes and, consequently, higher accelerations. In addition, this approach is also problematic due to the increased risk of slippage inherent to omnidirectional wheels compared to other wheel geometries.

### 3.6 Experimental Results

After evaluating the performance of the proposed controller in a simulation environment, experimental tests were conducted with a real omnidirectional robot to validate the results. The proposed controller was compared to several controllers from the literature, including a PD controller currently used in the navigation stack of the CRIIS robots, a feedforward controller proposed by Sousa et al. [16], and an NMPC controller with optimization constraints developed using the open-source libraries CasADi [57] and Ipopt [58]. Furthermore, the reference trajectory used in the experiments was the same as in the simulation tests. Additionally, some of the experiments were recorded and are available in this video <sup>6</sup>. Additionally, the robot used in the experiments was the omnidirectional robot codenamed “Adam”, illustrated in figure 3.2a. A list of relevant parameters for this robot is presented below.

- Processor: Intel Celeron N5105 2.00 *GHz* - 2.90 *GHz* (LattePanda 3 Delta processor)
- Gear reduction ratio ( $n$ ): 64
- Wheel diameter ( $D$ ): 0.102 *m*
- Front-to-back wheel distance ( $l_1$ ): 0.265 *m*
- Left-to-right wheel distance ( $l_2$ ): 0.270 *m*
- Trajectory tracking controller frequency: 40 *Hz*
- Odometry localization frequency: 100 *Hz*
- Absolute localization system frequency: 40 *Hz*
- Wheels maximum linear velocity ( $v_{max}$ ): 0.6 *m/s*
- Estimated time constants:  $\tau_v = 0.075$  *s*,  $\tau_{v_n} = 0.082$  *s*,  $\tau_w = 0.059$  *s* (see figure 3.5)

It is important to note that the absolute localization system employed in the following experiments relies on matching laser scanner point clouds through an optimization process developed by the mobile robot team at CRIIS [59]. With a steady-state linear velocity of 0.5 *m/s* and an up-to-date map of the environment, this system achieves a localization error of less than 1 *cm* (0.01 *m*) in position and 1° (0.017 *rad*) in orientation. An up-to-date map refers to one obtained on the same day as the experiments, thereby ensuring minimal discrepancies between the map and the actual environment due to any moved objects. Nevertheless, all results were obtained using the same robot and localization system, ensuring the validity of relative comparisons in the following sections.

---

<sup>6</sup><https://youtu.be/msU3yxTHz3s>

### 3.6.1 PD-only controller

In order to validate the simulation results, the proposed controller, using both the kinematic and the dynamic model for different control horizons, was compared with a PD controller. As such, the following parameters were used for each controller.

- Proposed controller:
  - Cost function parameters:  $dt = 0.025\text{ s}$ ,  $N_p = 20$ ,  $N_c = 1$  to  $4$ ,  
 $Q = \text{diag}\{1, 1, 1, 0, 0, 0\}$ ,  $R = \text{diag}\{1, 1, 1\}$
  - Optimization solver: Fletcher–Reeves
  - Optimization parameters:  $\varepsilon = 0.001$ ,  $itr_{max} = 20$
  - Dynamic model parameters:  $\tau_v = 0.1\text{ s}$  or  $0.3\text{ s}$ ,  $\tau_{v_n} = 0.1\text{ s}$  or  $0.3\text{ s}$ ,  $\tau_w = 0.1\text{ s}$  or  $0.3\text{ s}$
  - Motion velocity profile (S-Curve):  $v_{max} = 0.6\text{ m/s}$ ,  $a_{max} = 2.0\text{ m/s}^2$ ,  $j_{max} = 0.8\text{ m/s}^3$ ,  
 $v_{min} = 0.1\text{ m/s}$
  - Velocity scaling:  $v_{max} = 0.6\text{ m/s}$
- PD controller:
  - Proportional gains:  $K_p = [1.5, 1.5, 9.0]$
  - Derivative gains:  $K_d = [0.1, 0.1, 0.3]$

Figure 3.13 and table 3.5 present the results obtained for the experiments using a steady-state linear velocity of  $0.5\text{ m/s}$  and the robot time constants of  $0.1\text{ s}$ . The results are coherent with the simulation results, namely the proposed controller resulted in smaller errors both in position and orientation when compared with the PD controller, as indicated in table 3.5. Furthermore, the processing times of the proposed controller were for both the kinematic and dynamic models always less than  $6.71\text{ ms}$ . Moreover, similarly to the simulations, considering the smaller time constants ( $0.1\text{ s}$ ), the use of a dynamic model didn't achieve smaller errors than the kinematic model. Additionally, the increase in the control horizon didn't produce significant decreases in the errors. However, it also didn't result in a significant increase in processing time.

Figure 3.13 and table 3.5 present the results from experiments conducted with a steady-state linear velocity of  $0.5\text{ m/s}$  and robot time constants of  $0.1\text{ s}$ . The experimental results align with the simulation results, demonstrating that the proposed controller produced smaller errors in both position and orientation compared to the PD controller, as shown in table 3.5. Additionally, the processing times for the proposed controller were minimal, consistently remaining under  $6.71\text{ ms}$  for both kinematic and dynamic models. Similar to the simulations, the dynamic model did not produce smaller errors than the kinematic model, when considering small time constants ( $0.1\text{ s}$ ). Furthermore, increasing the control horizon did not significantly reduce the errors. However, the processing time cost was also insignificant, increasing only from  $1.43\text{ ms}$  to  $2.33\text{ ms}$  in the dynamic model.

**Table 3.5:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

| Metric                                | PD          | NMPC Kinematic Model |           |              |              | NMPC Dynamic Model |           |           |           |
|---------------------------------------|-------------|----------------------|-----------|--------------|--------------|--------------------|-----------|-----------|-----------|
|                                       |             | $N_c = 1$            | $N_c = 2$ | $N_c = 3$    | $N_c = 4$    | $N_c = 1$          | $N_c = 2$ | $N_c = 3$ | $N_c = 4$ |
| $\epsilon_{avg,d} \text{ (m)}$        | 0.017       | 0.007                | 0.007     | 0.007        | <u>0.006</u> | 0.008              | 0.008     | 0.007     | 0.007     |
| $\epsilon_{max,d} \text{ (m)}$        | 0.093       | 0.049                | 0.042     | 0.036        | <u>0.032</u> | 0.055              | 0.050     | 0.050     | 0.040     |
| $\epsilon_{avg,\theta} \text{ (rad)}$ | 0.030       | 0.024                | 0.024     | 0.021        | <u>0.020</u> | 0.026              | 0.025     | 0.024     | 0.025     |
| $\epsilon_{max,\theta} \text{ (rad)}$ | 0.113       | 0.067                | 0.064     | <u>0.063</u> | 0.069        | 0.073              | 0.066     | 0.065     | 0.069     |
| $t_{avg,proc} \text{ (ms)}$           | <u>0.64</u> | 1.31                 | 1.48      | 1.74         | 2.00         | 1.43               | 1.67      | 1.81      | 2.33      |
| $t_{max,proc} \text{ (ms)}$           | <u>1.98</u> | 3.35                 | 3.46      | 4.64         | 5.58         | 3.67               | 4.73      | 5.26      | 6.71      |

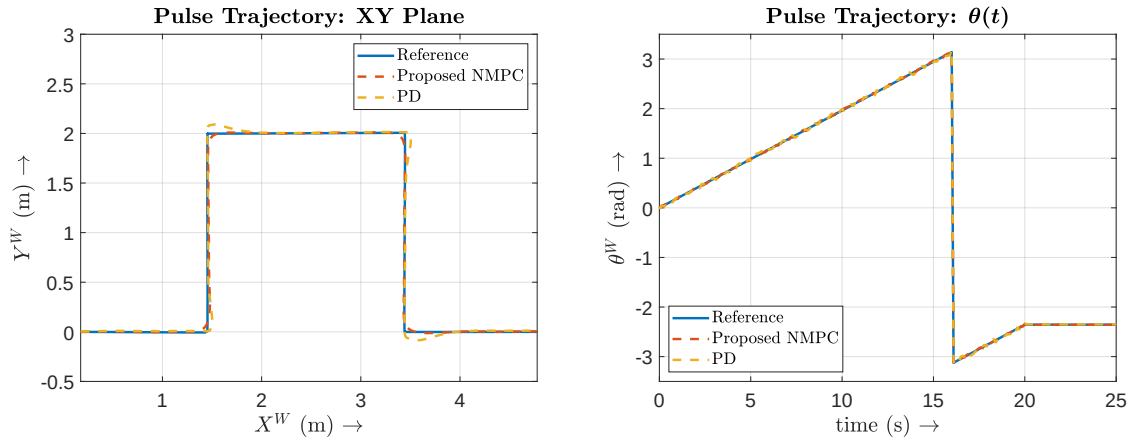


**Figure 3.13:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

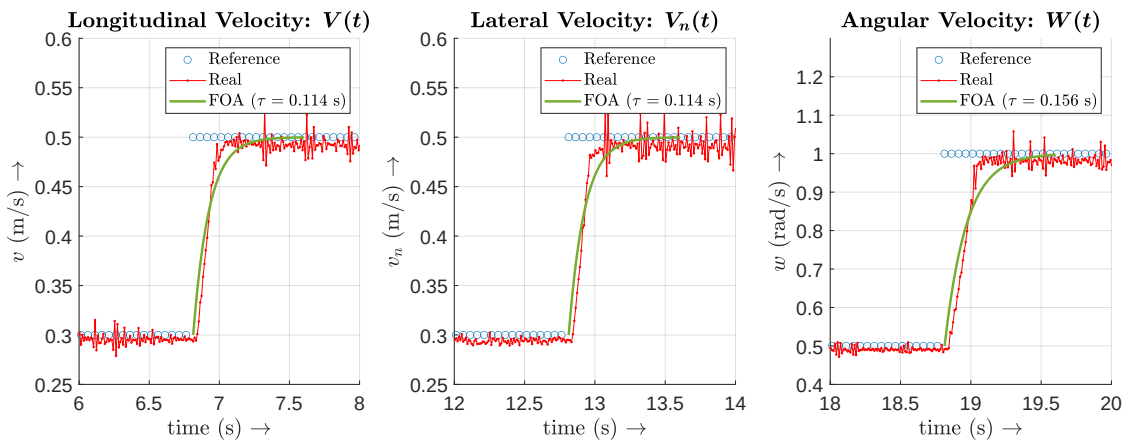
Then, in order to validate the results for higher robot time constants, the time constant estimation was performed on a different robot that was also developed at CRIIS. This new robot, codenamed “Friday”, is significantly larger and heavier than the initial one (see figure 3.14). However, the resulting time constants were not significantly different from those of the initial robot, as illustrated in figure 3.15. This similarity is due to the more powerful motors in the larger robot, allowing the PI controllers of the wheels to achieve the same dynamics despite the increased size and weight. As such, the estimated time constants, for the mobile robot “Friday”, were not high enough in order to produce different results.



(a) left view



(b) front view

**Figure 3.14:** Robot codename “Friday”**Figure 3.15:** First Order Approximation (FOA) of the robot velocity dynamics for the robot with codename “Friday”; the symbol  $\tau$  denotes the robot’s time constants

Therefore, a different approach was taken by tuning the PI controllers of the mobile robot “Adam” to achieve higher time constants ( $\geq 0.3$  s). The newly estimated time constants are presented in figure 3.16.

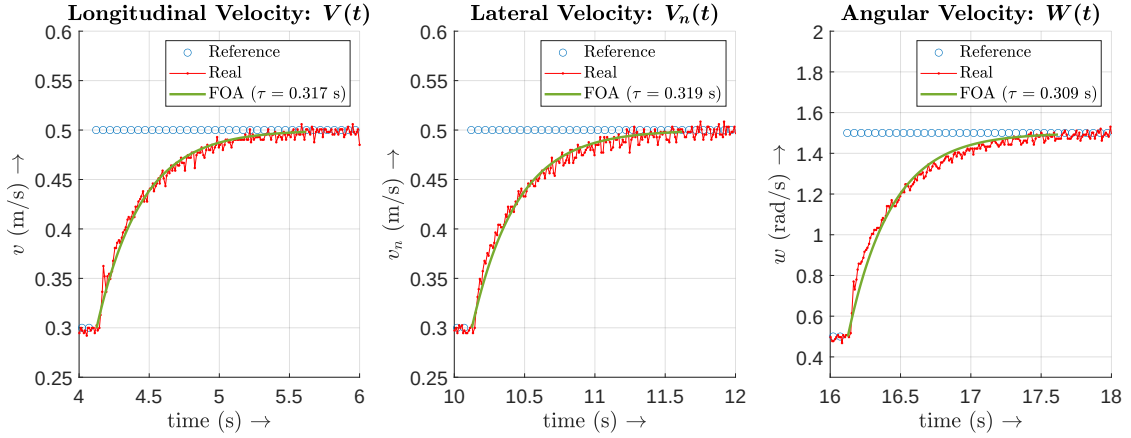


**Figure 3.16:** First Order Approximation (FOA) of the delayed robot velocity dynamics for the robot with codename “Adam”; the symbol  $\tau$  denotes the robot’s time constants

A new set of experiments was conducted with the new delayed robot dynamics, in which the dynamic model time constants were adjusted to  $0.3\text{ s}$ . Figure 3.17 illustrates the results of these experiments, which are also summarized in table 3.6. Once again, the experimental results align with simulation results, confirming that the dynamic model proves superior to a kinematic model-only approach for higher time constants ( $\geq 0.3\text{ s}$ ). Figure 3.17 clearly demonstrates the disparity between the two models: the kinematic model shows overshoot followed by minor oscillations at each trajectory corner, contrasting with the dynamic model’s performance. Additionally, employing a control horizon of 4 minimized errors in both position and orientation ( $\epsilon_{avg,d} = 0.009\text{ m}$ ,  $\epsilon_{max,d} = 0.037\text{ m}$ ,  $\epsilon_{avg,\theta} = 0.025\text{ rad}$ ,  $\epsilon_{max,\theta} = 0.091\text{ rad}$ ) without significant computational costs ( $t_{avg,proc} = 2.42\text{ ms}$ ,  $t_{max,proc} = 6.51\text{ ms}$ ), when compared to the PD controller ( $t_{avg,proc} = 0.65\text{ ms}$ ,  $t_{max,proc} = 2.16\text{ ms}$ ).

**Table 3.6:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5\text{ m/s}$ ; Time constants of  $0.3\text{ s}$

| Metric                        | PD          | NMPC Kinematic Model |           |              |           | NMPC Dynamic Model |              |           |              |
|-------------------------------|-------------|----------------------|-----------|--------------|-----------|--------------------|--------------|-----------|--------------|
|                               |             | $N_c = 1$            | $N_c = 2$ | $N_c = 3$    | $N_c = 4$ | $N_c = 1$          | $N_c = 2$    | $N_c = 3$ | $N_c = 4$    |
| $\epsilon_{avg,d} (m)$        | 0.027       | 0.016                | 0.017     | 0.018        | 0.019     | 0.010              | 0.010        | 0.010     | <u>0.009</u> |
| $\epsilon_{max,d} (m)$        | 0.162       | 0.066                | 0.073     | 0.080        | 0.090     | 0.040              | <u>0.037</u> | 0.038     | <u>0.037</u> |
| $\epsilon_{avg,\theta} (rad)$ | 0.031       | 0.026                | 0.026     | <u>0.025</u> | 0.026     | 0.029              | 0.027        | 0.028     | <u>0.025</u> |
| $\epsilon_{max,\theta} (rad)$ | 0.141       | 0.111                | 0.115     | 0.106        | 0.111     | 0.096              | <u>0.090</u> | 0.092     | 0.091        |
| $t_{avg,proc} (ms)$           | <u>0.65</u> | 1.37                 | 1.55      | 1.85         | 2.04      | 1.49               | 1.83         | 2.16      | 2.42         |
| $t_{max,proc} (ms)$           | <u>2.16</u> | 3.65                 | 3.79      | 5.07         | 6.23      | 4.08               | 5.46         | 6.02      | 6.51         |

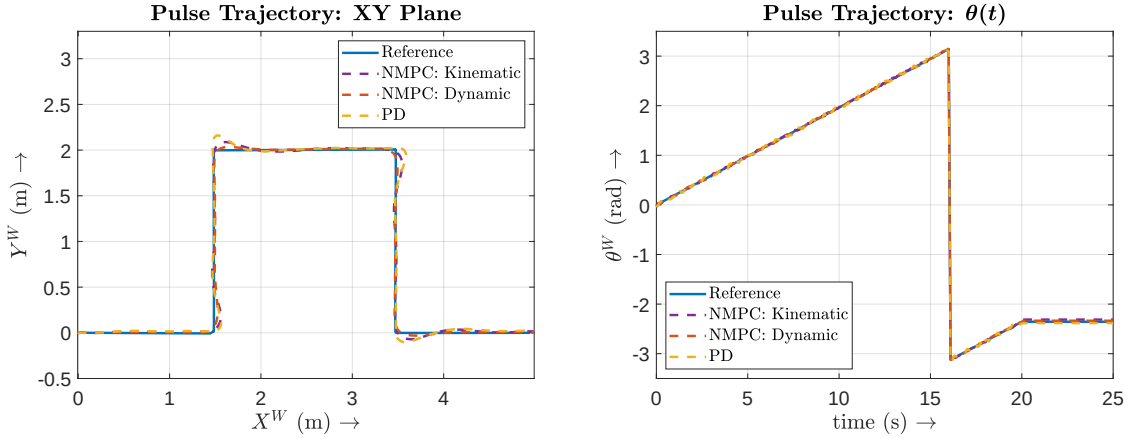


**Figure 3.17:** Comparison of the proposed controller with a PD controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.3 \text{ s}$

### 3.6.2 Feedforward controller

In order to compare the proposed controller with another controller featuring prediction capabilities, the feedforward controller proposed by Sousa et al. [16] was implemented. This controller utilizes  $M$  equidistant points along the future trajectory. These points start from the robot’s current position, specifically from the closest position on the path. During each control cycle, they are used to compute a feedforward component for each of the robot’s velocities ( $v$ ,  $v_n$ , and  $w$ ). These points, combined with the least squares method, determine the first and second derivatives needed for the feedforward component. Additionally, the controller includes a PD controller as the feedback component. As such, the following parameters were utilized for the feedforward controller:

- Feedforward component:
  - Number of trajectory points ( $M$ ): 10, 15 or 20
  - Time constants:  $\tau_v = 0.1 \text{ s}$ ,  $\tau_{v_n} = 0.1 \text{ s}$ ,  $\tau_w = 0.1 \text{ s}$
- Feedback component (PD controller):
  - Proportional gains:  $K_p = [1.5, 1.5, 9.0]$
  - Derivative gains:  $K_d = [0.1, 0.1, 0.3]$

Figure 3.18 and table 3.7 present the results obtained from experiments conducted with a steady-state linear velocity of  $0.5 \text{ m/s}$  and robot time constants of  $0.1 \text{ s}$ . In these experiments, the number of points used in the feedforward controller was varied to assess its impact on controller performance. As shown in table 3.5, the proposed controller achieved smaller position errors than the feedforward controller, although the difference was not significant. However, the feedforward controller had a shorter processing time, positioning it between the PD-only controller and the proposed NMPC controller. Additionally, the feedforward controller outperformed the proposed controller regarding average orientation error ( $t_{avg,proc}$ ) for all values of  $M$ . However, the proposed controller’s maximum orientation error ( $t_{max,proc}$ ) was smaller. This can be attributed to the

feedforward controller using an independent feedforward component for each robot reference velocity, unlike the proposed NMPC controller. This approach improves performance along straight segments but causes overshoots at the trajectory corners. This happens as only the reference orientations are used to determine the reference angular velocity ( $w$ ), without considering the reference positions of the trajectory. Furthermore, the feedforward controller demonstrated significant sensitivity to the parameter  $M$ . For instance, it produced overshoots when  $M = 10$  and a cut inside the path when  $M = 20$ , as illustrated in figure 3.13. Additionally, the cut inside had a non-symmetric shape, likely because the least squares method applies the same weight to all the reference points.

**Table 3.7:** Comparison of the proposed controller with a feedforward controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

| Metric                                | NMPC Dynamic Model | PD          | PD + Feedforward |          |          |
|---------------------------------------|--------------------|-------------|------------------|----------|----------|
|                                       | $N_c = 4$          |             | $M = 10$         | $M = 15$ | $M = 20$ |
| $\epsilon_{avg,d} \text{ (m)}$        | <u>0.007</u>       | 0.017       | 0.009            | 0.012    | 0.019    |
| $\epsilon_{max,d} \text{ (m)}$        | <u>0.040</u>       | 0.093       | 0.045            | 0.080    | 0.107    |
| $\epsilon_{avg,\theta} \text{ (rad)}$ | 0.025              | 0.030       | <u>0.009</u>     | 0.013    | 0.019    |
| $\epsilon_{max,\theta} \text{ (rad)}$ | <u>0.069</u>       | 0.113       | 0.074            | 0.105    | 0.147    |
| $t_{avg,proc} \text{ (ms)}$           | 2.33               | <u>0.64</u> | 0.80             | 0.87     | 0.93     |
| $t_{max,proc} \text{ (ms)}$           | 6.71               | <u>1.98</u> | 3.39             | 2.46     | 3.83     |

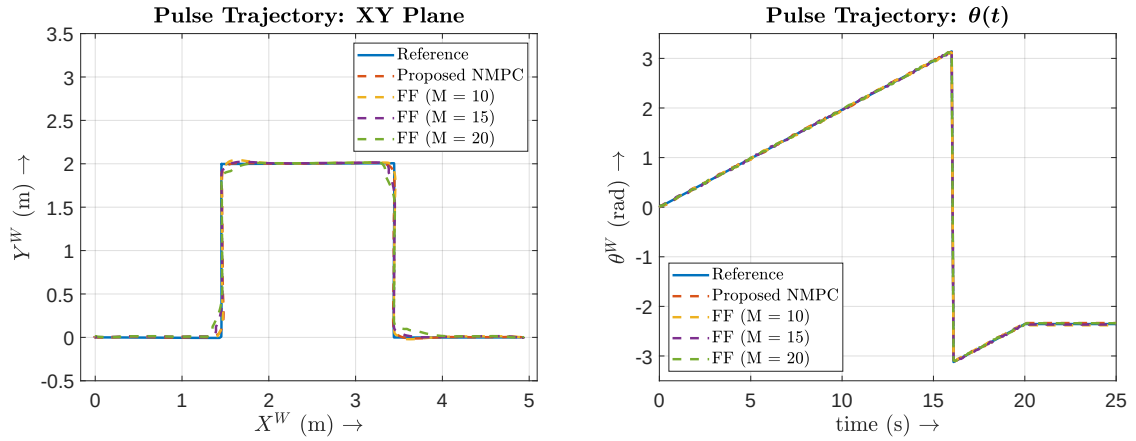


**Figure 3.18:** Comparison of the proposed controller with a feedforward controller for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

### 3.6.3 MPC with optimization constraints

Finally, an NMPC with optimization constraints was implemented using the open-source libraries CasADi [57] and Ipopt [58] to evaluate the impact of the proposed methods for handling velocity and acceleration inequality constraints outside the optimization problem. Considering the constraints on the velocities and acceleration of the robot wheels, the model used in the new NMPC

was slightly modified by incorporating equation 3.19. Consequently, the input of the new system model is  $\mathbf{u} = [v_{1,ref}, v_{2,ref}, v_{3,ref}, v_{4,ref}]^\top$ , while the system state remains  $\mathbf{x} = [x, y, \theta, v, v_n, w]^\top$ . Furthermore, the new system state equation  $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$  is represented in equation 3.20.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{v}_n \\ \dot{w} \end{bmatrix} = \begin{bmatrix} v \cdot \cos \theta - v_n \cdot \sin \theta \\ v \cdot \sin \theta + v_n \cdot \cos \theta \\ w \\ -\frac{1}{\tau_v} + \frac{1}{\tau_v} \cdot \frac{v_{1,ref} - v_{2,ref} + v_{3,ref} - v_{4,ref}}{4} \\ -\frac{1}{\tau_{v_n}} + \frac{1}{\tau_{v_n}} \cdot \frac{-v_{1,ref} - v_{2,ref} + v_{3,ref} + v_{4,ref}}{4} \\ -\frac{1}{\tau_w} + \frac{1}{\tau_w} \cdot \frac{-v_{1,ref} - v_{2,ref} - v_{3,ref} - v_{4,ref}}{2 \cdot (l_1 + l_2)} \end{bmatrix} \quad (3.20)$$

Furthermore, the following parameters were used for the new NMPC controller with optimization constraints:

- Cost function parameters:  $dt = 0.025 \text{ s}$ ,  $N_p = 20$ ,  $N_c = 4$ ,  
 $Q = \text{diag}\{1, 1, 1, 0, 0, 0\}$ ,  $R = \text{diag}\{1, 1, 1, 1\}$
- Optimization solver: Ipopt
- Optimization parameters:  $\varepsilon = 0.001$ ,  $itr_{max} = 20$
- Dynamic model parameters:  $\tau_v = 0.1 \text{ s}$ ,  $\tau_{v_n} = 0.1 \text{ s}$ ,  $\tau_w = 0.1 \text{ s}$
- Velocity inequality constraints:  $u_{max} = -u_{min} = [0.6, 0.6, 0.6, 0.6] \text{ m/s}$
- Acceleration inequality constraints:  $\Delta u_{max} = -\Delta u_{min} = [0.05, 0.05, 0.05, 0.05]$  (equivalent to  $2 \text{ m/s}^2$ )

Moreover, two additional versions of the new NMPC controller were also tested: one without any inequality constraints and another with a maximum acceleration of only  $1 \text{ m/s}^2$ . The first version aimed to compare the different optimization solvers used in the proposed controller (Fletcher-Reeves) and the new NMPC controller (Ipopt). The second version is intended to demonstrate the capability of the new NMPC formulation to restrict the maximum acceleration along the entire path. Consequently, since the maximum acceleration is now lower in this new version, the prediction horizon had to be extended from 20 to 40 to allow a complete variation from the minimum to the maximum velocity for each robot wheel.

The results from the experiments are presented in figure 3.19 and table 3.8. As shown in table 3.8, the processing time of the new NMPC is substantially higher than that of the proposed NMPC controller for all versions tested. Even the version without any optimization constraints had a processing time three times higher than the proposed controller. The Ipopt solver's design for handling constrained optimization problems contributes to its complex structure with numerous algorithms and verifications that are unnecessary for unconstrained optimization problems, resulting in slower convergence. Despite this, the NMPC using the Ipopt solver produced equivalent errors in both position and orientation compared to the proposed controller when considering

the same velocity and acceleration constraints, indicating that the proposed methods for handling velocity and acceleration inequality constraints outside the optimization problem are appropriate. Furthermore, with a maximum acceleration of  $1 \text{ m/s}^2$ , the controller produced a cut inside each corner (see figure 3.19), indicating that the wheel acceleration was saturated by the controller constraints. This is an advantage over the proposed controller, where the acceleration along the middle of the trajectory cannot be precisely defined and depends on the tuning of the cost function weights ( $R$ ). However, the significant increase in processing times implies that the new NMPC structure is not suitable for robots with limited processing power, as the maximum processing time required almost reaches the control cycle time step.

**Table 3.8:** Comparison of the proposed controller with an NMPC controller using the open-source libraries CasADi and Ipopt for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

| Metric                                | NMPC Dynamic Model                    |                                       | NMPC Ipopt                                         |                                                    |
|---------------------------------------|---------------------------------------|---------------------------------------|----------------------------------------------------|----------------------------------------------------|
|                                       | $N_p = 20, N_c = 4$<br>No constraints | $N_p = 20, N_c = 4$<br>No constraints | $N_p = 40, N_c = 4$<br>$a_{max} = 1 \text{ m/s}^2$ | $N_p = 20, N_c = 4$<br>$a_{max} = 2 \text{ m/s}^2$ |
| $\epsilon_{avg,d} \text{ (m)}$        | <u>0.007</u>                          | 0.009                                 | 0.020                                              | 0.009                                              |
| $\epsilon_{max,d} \text{ (m)}$        | 0.040                                 | 0.035                                 | 0.107                                              | <u>0.034</u>                                       |
| $\epsilon_{avg,\theta} \text{ (rad)}$ | 0.025                                 | 0.021                                 | 0.027                                              | <u>0.020</u>                                       |
| $\epsilon_{max,\theta} \text{ (rad)}$ | <u>0.069</u>                          | 0.070                                 | 0.106                                              | <u>0.069</u>                                       |
| $t_{avg,proc} \text{ (ms)}$           | <u>2.33</u>                           | 6.21                                  | 11.36                                              | 8.61                                               |
| $t_{max,proc} \text{ (ms)}$           | <u>6.71</u>                           | 17.58                                 | 25.61                                              | 19.08                                              |

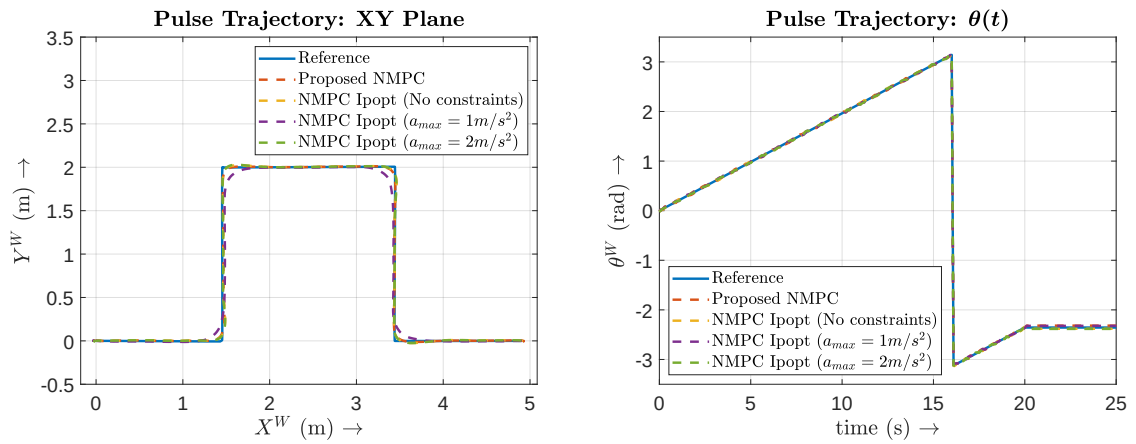


**Figure 3.19:** Comparison of the proposed controller with an NMPC controller using the open-source libraries CasADi and Ipopt for a pulse trajectory: Robot “Adam”; Steady-state linear velocity of  $0.5 \text{ m/s}$ ; Time constants of  $0.1 \text{ s}$

### 3.7 Discussion

This chapter presented an NMPC control architecture for trajectory tracking for omnidirectional mobile robots. Moreover, the proposed controller was compared with other controllers from the literature, namely a PD-only controller, a feedforward controller [16], and an NMPC controller with optimization constraints using the open-source libraries CasADi [57] and Ipopt [58].

First, simulations performed in Gazebo validated the proposed controller architecture by direct comparison to a PD controller. The results indicated that a first-order approximation of the robot dynamics is sufficient for accurately controlling a mobile robot. For robots with fast dynamics (time constants close to 0.1 s), a simple kinematic model was adequate, as incorporating a dynamic model did not affect position and orientation errors. Additionally, variations in steady-state velocity highlighted the differences in errors between the PD controller and the proposed NMPC controller. However, the differences between the dynamic and kinematic models became noticeable only when the robot's time constants changed, suggesting that a dynamic model is justifiable for higher time constants ( $\geq 0.3$  s) and has a minor relation to the robot's velocity. Moreover, a control horizon of 4 resulted in smaller errors across all results, reducing maximum and average position errors by about 10%, at the cost of 2 ms in processing time, which is twice the processing time required for a control horizon of 1. Finally, by handling velocity and acceleration inequality constraints outside the optimization process, the processing time was significantly reduced to less than 5 ms, being very close to the PD controller's processing time of around 1 ms.

The experiments conducted in real environments with an omnidirectional mobile robot confirmed the simulation results. Specifically, when compared to the PD controller, the proposed NMPC controller produced smaller errors but required slightly higher processing times. Moreover, the feedforward controller outperformed the proposed NMPC controller in terms of processing time and average orientation error due to its simple structure and independent control loops. However, the proposed NMPC controller achieved smaller maximum and average position errors, as well as smaller maximum orientation error. Additionally, the feedforward controller proved to be highly sensitive to the number of trajectory points ( $M$ ) used, with significant degradation in performance as the number of points increased. For instance, errors doubled when  $M$  increased from 10 to 20. Finally, comparisons with an NMPC that includes optimization constraints showed that incorporating velocity and acceleration inequality constraints within the optimization process significantly increases the required processing time. Specifically, the processing time was more than three times higher than the proposed controller's, making this NMPC structure inappropriate for robots with limited computational power. However, this structure has an advantage over the proposed NMPC, as it allows for the direct definition of maximum acceleration for the entire trajectory.

## Chapter 4

# Collision Avoidance

An analysis of the methods developed for mobile robots to avoid collisions is presented in this chapter. Section 4.1 provides an overview of the current methods employed in industrial settings for collision avoidance. Section 4.2 presents a collision avoidance method based on security zones designed to halt the robot and prevent collisions. Section 4.3 formulates an obstacle avoidance method developed to contour stationary obstacles. Section 4.4 presents the simulation results obtained using the proposed methods. Finally, section 4.5 is dedicated to an analysis of the results and the formulation of conclusions.

### 4.1 Collision Avoidance Overview

In the current literature, a considerable number of collision avoidance algorithms have been proposed, as presented in section 2.3. One prominent example is the A\* algorithm, widely used for pathfinding and collision avoidance in robotics. However, the majority of these methods, including A\*, require a complex structure and implementation. On the other hand, in certain applications, the complexity of these algorithms can be reduced by assuming that the obstacles are stationary and that they are less likely to exist. In industrial applications, the paths that the robots must follow are predefined. This facilitates the coordination of multiple robots in the same space, reducing the occurrence of deadlocks and enhancing overall productivity. To illustrate, the navigation stack utilized by the mobile robots of the CRIIS in INESC TEC employs a graph to map all the predefined paths (see figure 4.1). In this structure, each graph node can be connected to another one, either by a line segment or a parametric curve such as a spline. Therefore, since the robot's path is already predetermined, a collision avoidance algorithm based on a global planner cannot be used because it would require the robot to move freely within the space. Conversely, using a collision avoidance algorithm based on a local planner is often also not feasible, as robots typically cannot deviate significantly from their main path. Doing so could lead them onto another path, potentially blocking the way for other robots and disrupting the overall logistics. Additionally, even if the robot does not obstruct another path, a large deviation can still be unsafe, as industrial robots are often large and heavy.

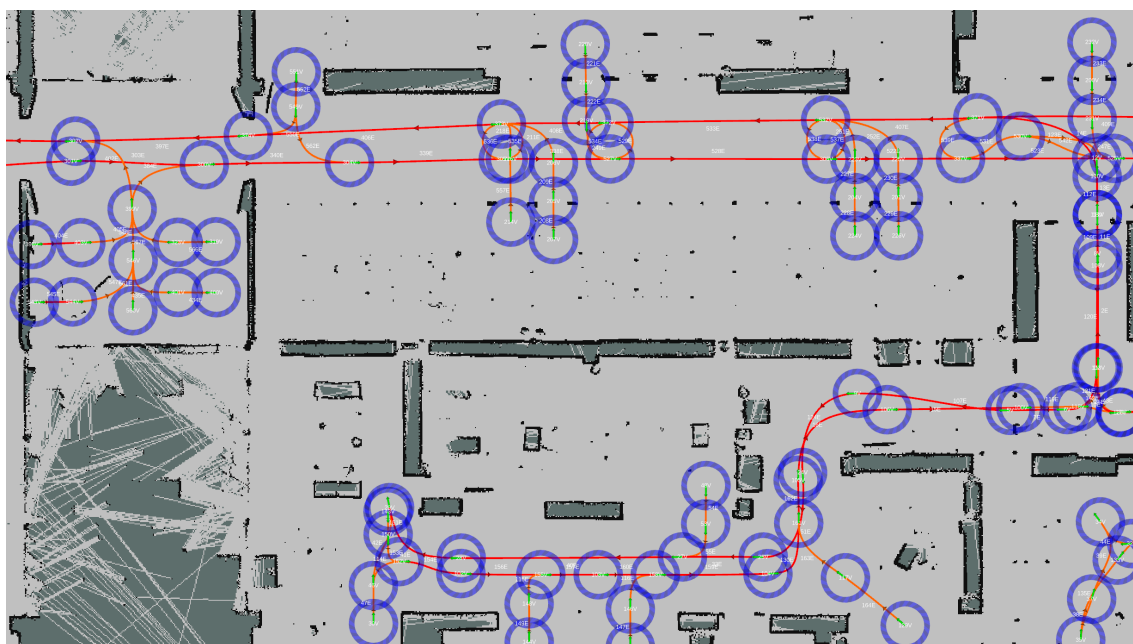


**Figure 4.1:** Graph representation of the navigation paths used by CRIIS robots at INESC TEC

The objective is to develop two collision avoidance methods tailored for industrial applications using this information. The first method aims to create a virtual security zone around the robot, stopping it when an obstacle cannot be contoured. The second method focuses on generating a smooth path to navigate around stationary obstacles and return to the main path.

## 4.2 Virtual Security Zone Method

The implementation of virtual security zones to prevent collisions with obstacles follows a straightforward procedure. By knowing the robot's velocity at a given time and the dimensions of its structure, it is possible to predict the robot's future trajectory for a short time interval. This allows the determination of whether or not a collision with an obstacle will occur. Therefore, a virtual security zone can be defined as a prediction of the space that the robot will occupy in the near future. For mobile robots, the algorithm can be simplified to two dimensions, as they do not move along the  $z$ -axis. Consequently, the robot's structure can be projected onto the  $xy$  plane, and the security zones can be represented as areas, as illustrated in figure 4.2.

Moreover, the footprint of a mobile robot can be quite complex, given the possibility of equipping it with various additional devices, such as manipulators and other tools. Nevertheless, the footprint can be simplified to a rectangle bounding box without significantly deviating from the original shape. The robot bounding box can thus be parametrized using four distances: robot length, robot width, distance from center to front, and distance from center to left. The first two parameters are relatively straightforward, whereas the last two may be omitted in certain instances, as they are intended to specify the center of the robot, particularly for asymmetric mobile robots. The information regarding the robot's center is of significant importance, as it indicates the geometric center of the robot wheels and, consequently, the center of rotation of the robot. Moreover,

the approximation of a rectangle and the consequent expansion result in a relatively conservative estimation of the percentage of the virtual security bounding box in relation to the footprint bounding box. However, for reasons of security, it is preferable to adopt a conservative approach rather than an optimistic one.

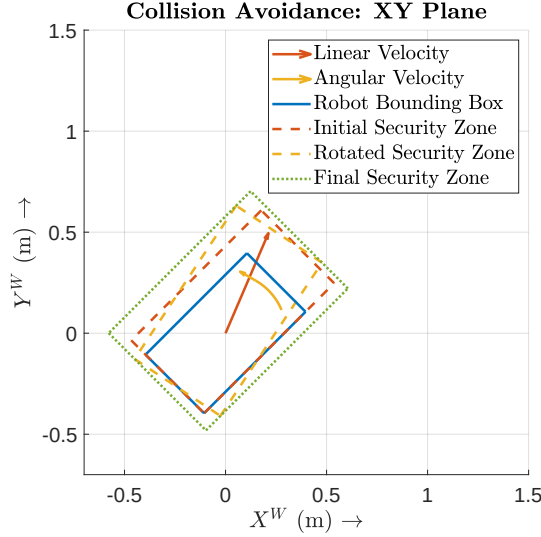


**Figure 4.2:** Collision Avoidance: Virtual security zone method

In the proposed methodology, the security zone is determined by expanding the robot's bounding box in the direction of motion, incorporating both linear and angular components. For linear motion, the initial security zone is created by extending the robot's bounding box toward its linear velocity. The linear velocity,  $v$ , is aligned with the robot's x-axis for mobile robots with differential drive or tricycle steering geometries. In contrast, omnidirectional robots, which can move in any direction regardless of orientation, have velocity components along both the x-axis,  $v$ , and y-axis,  $v_n$ . Thus, each edge of the robot's bounding box is expanded along the x-axis or y-axis by a certain distance, based on the sign and magnitude of the robot's velocities.

The distance  $D(x)$  formulated in equation 4.1, represents the safety distance, where  $x$  can be either  $v$  or  $v_n$ . In this equation, the first component ( $k_1 \cdot \Delta t \cdot |x|$ ), determines the distance that the robot will travel in that direction over the next  $k_1$  control cycles, where  $\Delta t$  represents the control cycle time interval. If the control cycle operates at a lower frequency, the time interval between two algorithm verifications is longer, resulting in a larger security zone as the robot travels a longer distance. On the other hand, if the control cycle operates at a higher frequency, the time interval is shorter, leading to a smaller security zone since the robot travels a shorter distance. Furthermore, due to physical constraints, the robot cannot stop instantaneously. Therefore, a second component is considered ( $k_2 \cdot |x|$ ), which predicts the distance the robot travels along a given direction until it stops completely. Here, the parameter  $k_2$  serves as a tuning factor, directly proportional to the magnitude of the velocity,  $|x|$ , along that axis.

$$D(x) = k_1 \cdot \Delta t \cdot |x| + k_2 \cdot |x| \quad (4.1)$$

Furthermore, the robot's behavior during a break depends on different conditions, such as friction forces and the robot's weight. In particular, a linear relation between distance and velocity cannot be driven, especially for small velocities. Consequently, a minimum safety distance,  $D_{min}$ , is defined for each edge if the robot velocity along that axis is above a specified tolerance level,  $x_{tol}$ , as presented in equation 4.2. Utilizing a tolerance value facilitates the mitigation of noise in the magnitude of the velocity vectors.

$$D(x) = \begin{cases} \max(k_1 \cdot \Delta t \cdot |x| + k_2 \cdot |x|, D_{min}), & \text{if } |x| > x_{tol} \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

On the other hand, with regard to angular motion, a comparable methodology may be employed to determine an angle of rotation along the z-axis (equation 4.3), utilizing the robot angular velocity  $w$  and two tuning parameters,  $k_3$  and  $k_4$ . Furthermore, it is also possible to consider a minimum angle,  $\phi_{min}$ , when the magnitude of the angular velocity exceeds a certain tolerance value. Using the calculated angle, the corners of the initial security zone ( $P_{ori,i}$ ) can be rotated around the robot center using the rotation matrix  $R_z(\phi)$ , as illustrated in equation 4.4, obtaining the rotated corners ( $P_{rot,i}$ ) that form the rotated security zone.

Using the calculated angle, the corners of the initial security zone ( $P_{ori,i}$ ) can be rotated around the robot center. This rotation is achieved using the rotation matrix  $R_z(\phi)$ , as presented in equation 4.4. Subsequently, a rotated security zone is formed with these rotated corners ( $P_{rot,i}$ ).

$$\phi = \begin{cases} \max(k_3 \cdot \Delta t \cdot w + k_4 \cdot w, \phi_{min}), & \text{if } w > w_{tol} \\ 0, & \text{otherwise} \end{cases} \quad (4.3)$$

$$P_{rot,i} = R_z(\phi) \cdot P_{ori,i} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \cdot P_{ori,i} \quad (4.4)$$

Integrating the linear and angular motion components, a final security zone can be established by defining a rectangular security zone aligned with the robot's x-axis. This zone contains both the previously formulated security zones, as illustrated in figure 4.2. This involves determining the maximum and minimum values of  $x$  and  $y$  within the set comprising all corners of the initial and rotated security zones relative to the robot frame.

Finally, to prevent collisions, the algorithm executes at each control cycle, creating a safety zone that considers the current speed of the robot. The current speed can be based on wheeled odometry or on other fused estimates of velocity. Additionally, a 2D point cloud can be acquired using a LiDAR sensor, which may involve processing raw point cloud data or a deskewed point cloud to compensate for motion distortion caused by the robot's movement. This processed point cloud is then used to determine if a maximum number of points fall within the safety zone, triggering the robot to stop if necessary.

## 4.3 Obstacle Avoidance Method

The proposed obstacle avoidance algorithm is designed to navigate around small stationary obstacles that obstruct a portion of the main path. This algorithm relies on two key components: obstacle detection and path generation. Additionally, the algorithm requires a sensor to identify obstacles. Since LiDAR sensors are widely used in most mobile robots for localization purposes, the same sensor can be utilized for the collision avoidance algorithm, eliminating the need for an additional sensor.

Furthermore, at a higher level of abstraction, the algorithm is designed to create an expanded convex hull from the point cloud to represent the estimated shape of the obstacle. By using the vertices of this convex hull and their positions relative to the main path, a smooth path is generated to navigate around the obstacle using splines. This structure allows the algorithm to operate online, continuously adjusting the path as new information about the obstacle shape is received from the LiDAR.

### 4.3.1 Obstacle detection

The proposed algorithm relies on a 2D point cloud that exclusively comprises points associated with a single obstacle. Consequently, it is imperative to filter and segment all the points retrieved by the LiDAR. Clustering the points in the point cloud ensures that obstacles distant from each other are divided into separate point clouds. An algorithm capable of executing this task is detailed by Pfister et al. in [60]. Additionally, for simplicity, the algorithm assumes that this segmentation occurs beforehand, ensuring that the resulting point cloud includes only points from the nearest obstacle.

First, since the point cloud is acquired by a sensor mounted on the robot, as depicted in figure 4.3a, the exact size of the obstacle is not initially known. As such, a projection can be performed with a prediction of its length. This involves projecting the boundary points of the point cloud a certain distance backward, as shown in figure 4.3b. Identifying these boundary points involves finding the two points that are furthest apart. However, due to the disorganized nature and potential size of point clouds, this task can be computationally expensive. Nevertheless, optimized algorithms in libraries like Open3D <sup>1</sup> facilitate tasks such as organizing point clouds and generating convex hulls.

---

<sup>1</sup><https://www.open3d.org/>

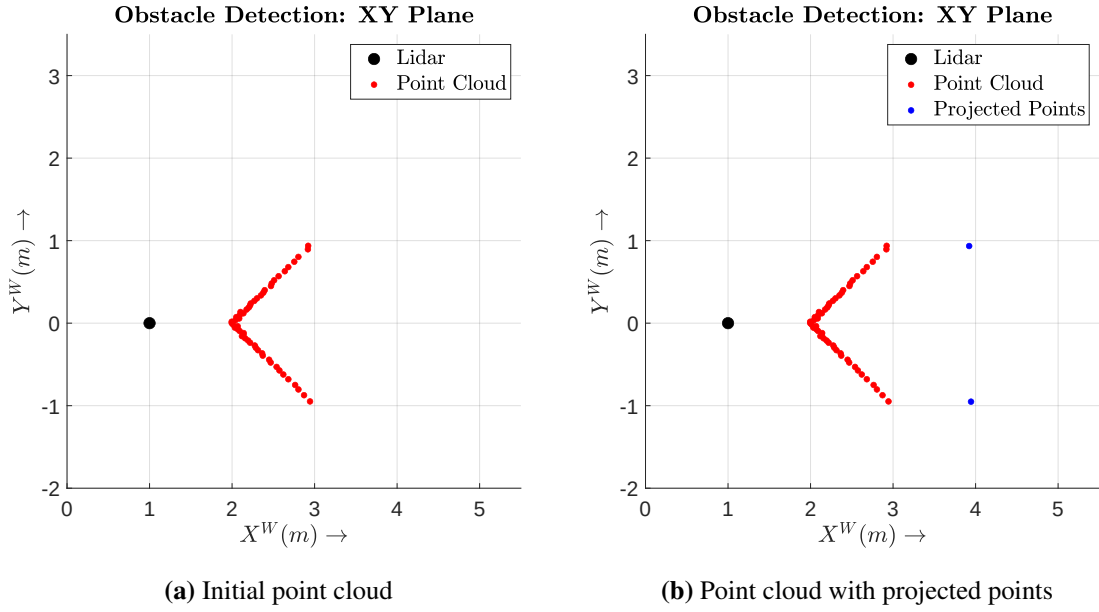


**Figure 4.3:** Obstacle detection: point clouds

Furthermore, each point's projection follows the equation 4.5, where  $P_1$ ,  $P_2$ ,  $P_{lidar}$ , and  $P_{mid}$  denote the positions of the boundary points, the LiDAR, and the midpoint of the boundary points, respectively. In addition,  $V_{proj}$  represents the projection vector, and  $d_{back}$  indicates the projection distance.

$$\begin{aligned}
 P_{med} &= \frac{P_1 + P_2}{2} \\
 V_{proj} &= \frac{P_{mid} - P_{lidar}}{\|P_{mid} - P_{lidar}\|} \\
 P_{proj,1} &= P_1 + d_{back} \cdot V_{proj} \\
 P_{proj,2} &= P_2 + d_{back} \cdot V_{proj}
 \end{aligned} \tag{4.5}$$

Subsequently, the point cloud and projected points can be used to create a convex hull encompassing all the points, as illustrated in figure 4.4a. Additionally, to facilitate the representation of the robot as a single point in space and, consequently, the detection of collisions, each convex hull vertex is expanded along the bisectors of the two edges that form that vertex. The distance of the expansion is proportional to the robot's dimensions, ensuring that a collision with the robot can only occur when the robot center is inside the expanded convex hull. The expanded convex hull is represented in figure 4.4b.

Given the inherent noise present in the point cloud data, the resulting convex hulls may contain unnecessary vertices, as depicted in figure 4.4. Therefore, additional procedures are considered to mitigate the impact of noise and streamline the contours of objects with minor irregularities. Subsequently, following the initial determination of the convex hull, the Ramer-Douglas-Peucker (RDP) algorithm [61] is employed. This algorithm, historically utilized for cartographic simplification, reduces a curve composed of line segments to a similar curve with fewer points. Moreover,

during the expansion phase, a tolerance angle between two consecutive edges determines whether a vertex should be expanded. This tolerance angle helps eliminate vertices that connect two nearly collinear edges. The resulting convex hulls generated by this revised algorithm are presented in figure 4.5. Finally, since the algorithm is designed to run in a cyclic manner, a buffer is maintained to store the previous  $n$  point clouds. This buffer is employed to mitigate the impact of noise and ensure that the obstacle convex hulls of consecutive cycles exhibit diminished variations.

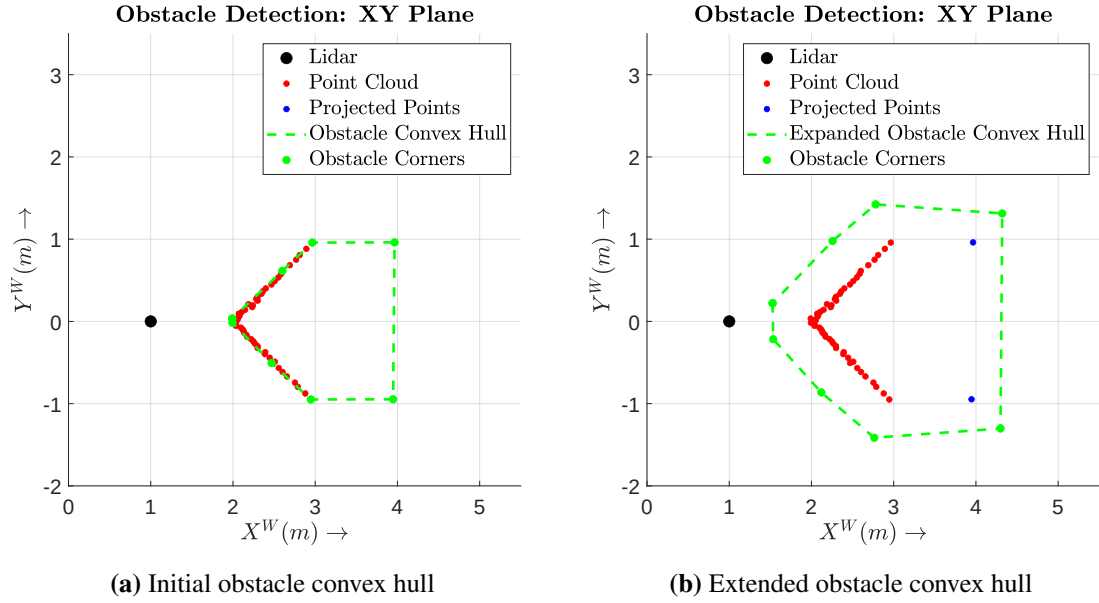


**Figure 4.4:** Obstacle detection: convex hulls

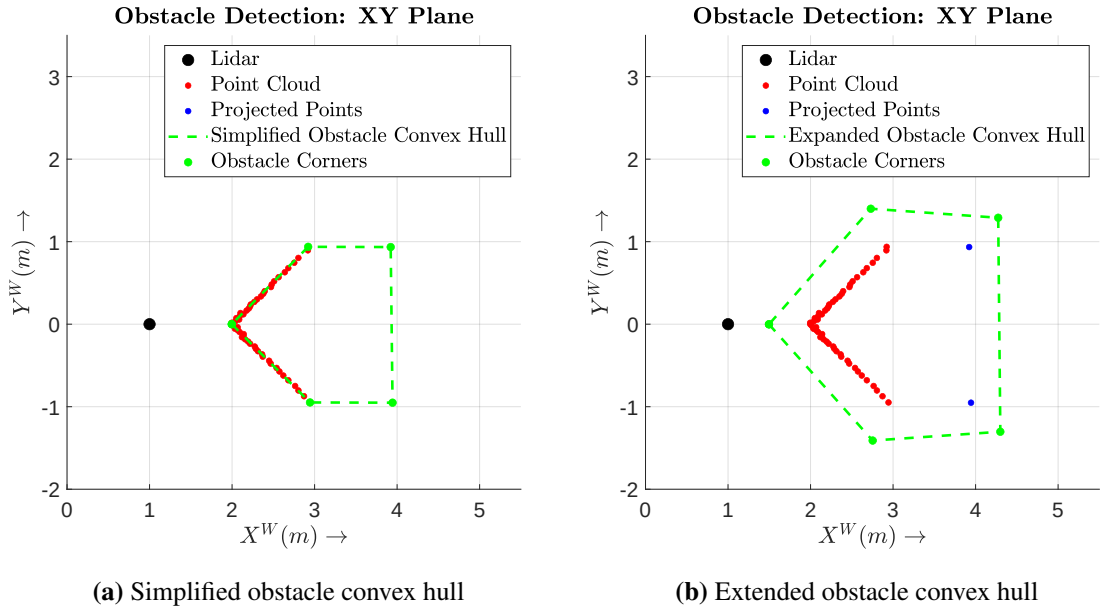


**Figure 4.5:** Obstacle detection: filtered convex hulls

### 4.3.2 Path generation

Once the expanded obstacle convex hull has been defined, a smooth path is created using parametric curves. As previously stated in section 2.2, parametric curves allow for the definition of smooth paths with a reduced number of parameters. Moreover, cubic splines are well suited for mobile robot paths as they can maintain continuity in both position and curvature without losing localism. Consequently, the path generation process begins with the definition of a set of poses that consider the vertices of the obstacle convex hull. These poses are then used to create a spline path, formulated as a cubic Bézier spline. Appendix C elaborates on spline formulation, offering deeper insights into the notations introduced in the current section. Additionally, it encompasses relevant numerical algorithms essential for trajectory tracking tasks, including determining the closest point on a spline and computing subsequent positions. Moreover, it also includes a possible implementation of a numeric algorithm for determining the spline length, which is necessary for the obstacle avoidance method proposed in the current section.

First, to define the set of poses, each vertex of the convex hull is projected onto the main path to determine its relative position (above or below) and its order. The method for computing these projections varies depending on the main path's formulation, whether it is a line, an arc, or a parametric curve. However, without loss of generality, the following equations will assume that the main path is a line segment. Therefore, given a line defined by an initial point  $(x_i, y_i)$  and a final point  $(x_f, y_f)$ , the projection of an arbitrary point  $(x, y)$  can be calculated using equation 4.6. After projecting all vertices, they can be divided into two ordered sets: one containing all the vertices above the main path and another containing all the vertices below the main path. The position of each vertex relative to the main path is determined by the sign of the variable  $k$ , while the order of the vertices within each set is based on their distance to the beginning of the path.

$$\begin{aligned}
 u_x &= \frac{x_f - x_i}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}} \\
 u_y &= \frac{y_f - y_i}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}} \\
 k &= \frac{x \cdot u_y - y \cdot u_x - x_i \cdot u_y + y_i \cdot u_x}{u_x^2 + u_y^2} \\
 x_n &= x - k \cdot u_y \\
 y_n &= y + k \cdot u_x
 \end{aligned} \tag{4.6}$$

Next, each vertex is moved a certain distance  $d_1$  along the bisectors. Additionally, two points are added to each set to represent the entry and exit points of the main path. These points are calculated using equation 4.7, where  $(x_{new,1}, y_{new,1})$  and  $(x_{new,2}, y_{new,2})$  represent the exit and entry points,  $d_2$  is a tuning parameter, and  $(x_{int,1}, y_{int,1})$  and  $(x_{int,2}, y_{int,2})$  represent the first and last intersection points of the convex hull with the main path. These intersection points are found by calculating all the intersections between each edge of the convex hull and the main path and then

identifying the closest and farthest intersection points.

$$\begin{aligned} \begin{bmatrix} x_{new,1} \\ y_{new,1} \end{bmatrix} &= \begin{bmatrix} x_{int,1} \\ y_{int,1} \end{bmatrix} - d_2 \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} \\ \begin{bmatrix} x_{new,2} \\ y_{new,2} \end{bmatrix} &= \begin{bmatrix} x_{int,2} \\ y_{int,2} \end{bmatrix} + d_2 \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} \end{aligned} \quad (4.7)$$

Furthermore, since a constant distance is not suitable for all obstacles due to their varying shapes and sizes, the calculation of the entry and exit points can be enhanced. This enhancement involves computing two additional points,  $(x_{new,3}, y_{new,3})$  and  $(x_{new,4}, y_{new,4})$ , using the first and last projected vertices of each set,  $(x_{proj,1}, y_{proj,1})$  and  $(x_{proj,2}, y_{proj,2})$ , and their distances of projection,  $|k_{proj,1}|$  and  $|k_{proj,2}|$ , as shown in equation 4.8. Moreover, a combination of both methods can be employed. This involves selecting the closest point to the beginning of the main path from  $(x_{new,1}, y_{new,1})$  and  $(x_{new,3}, y_{new,3})$ , and the closest point to the end of the main path from  $(x_{new,2}, y_{new,2})$  and  $(x_{new,4}, y_{new,4})$ , for each set.

$$\begin{aligned} \begin{bmatrix} x_{new,3} \\ y_{new,3} \end{bmatrix} &= \begin{bmatrix} x_{proj,1} \\ y_{proj,1} \end{bmatrix} - (d_2 + 0.5 \cdot |k|) \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} \\ \begin{bmatrix} x_{new,4} \\ y_{new,4} \end{bmatrix} &= \begin{bmatrix} x_{proj,2} \\ y_{proj,2} \end{bmatrix} + (d_2 + 0.5 \cdot |k|) \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} \end{aligned} \quad (4.8)$$

Finally, the orientation of each pose is determined by the direction vector from that pose to the next one in the set, given by  $\theta_i = \text{atan2}(y_{i+1} - y_i, x_{i+1} - x_i)$ . The exceptions are the first and last poses, which are aligned with the orientation of the main path,  $\theta_i = \text{atan2}(u_y, u_x)$ . Figure 4.6a shows the resulting poses generated by the algorithm for an arbitrary obstacle.

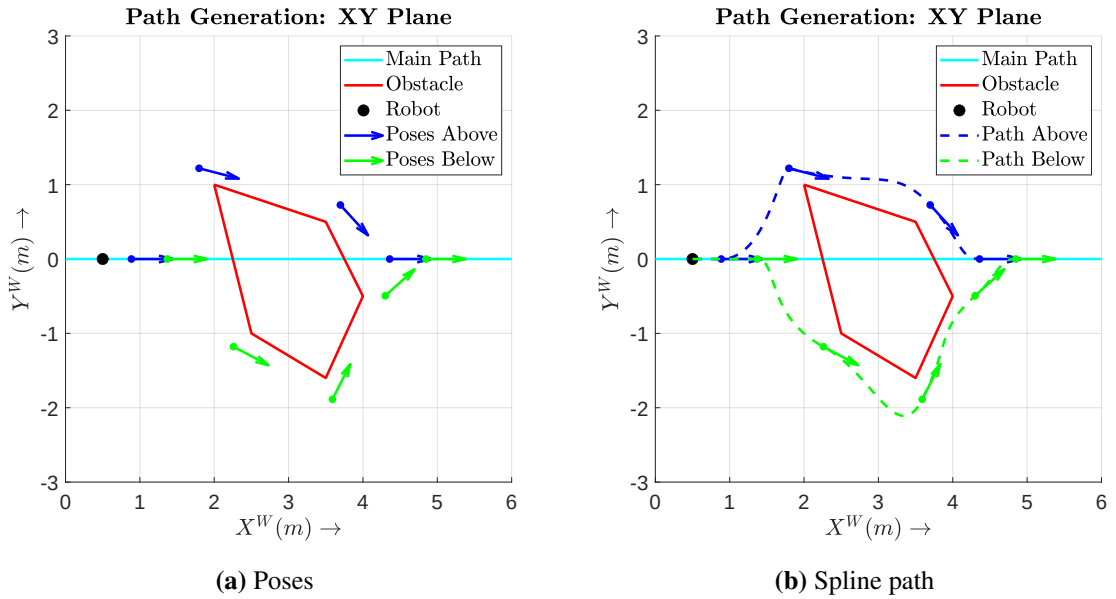


**Figure 4.6:** Path generation

After computing two sets of poses, a cubic Bézier spline is generated for each set by minimizing the strain energy of each spline segment, similar to the approach outlined by Yong and Cheng [62]. However, to ensure the smoothest possible path, each segment ideally minimizes the maximum curvature of the parametric curve. The curvature of a two-dimensional parametric curve relies on equation 4.9.

$$\kappa(t) = \frac{|x(t)' \cdot y''(t) - y'(t) \cdot x''(t)|}{\left(x'(t)^2 + y'(t)^2\right)^{3/2}} \quad (4.9)$$

As a result, the maximum curvature of a quadratic parametric curve can be determined analytically because it is a convex curve. However, this analytical solution does not apply to a cubic parametric curve, which requires a numerical algorithm. Alternatively, the minimization of the maximum curvature can be replaced by minimizing the strain energy,  $\int_0^1 S''(t)^2 dt$ , of the spline segment  $S(t)$ , as indicated by Yong and Cheng [62]. As such, considering two consecutive poses,  $(x_i, y_i, \theta_i)$  and  $(x_{i+1}, y_{i+1}, \theta_{i+1})$ , the four control points of the spline segment can be determined by the equation 4.10, where the parameters  $a_0$  and  $a_1$  are obtained by minimizing the strain energy, resulting in the equation 4.11 presented in the paper of Yong and Cheng [62]. In this equation,  $V_0 = [\cos \theta_i, \sin \theta_i]^T$  and  $V_1 = [\cos \theta_{i+1}, \sin \theta_{i+1}]^T$ .

$$\begin{aligned} P_0 &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} & P_1 &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \frac{a_0}{3} \begin{bmatrix} \cos \theta_i \\ \sin \theta_i \end{bmatrix} \\ P_3 &= \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} & P_2 &= \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \frac{a_1}{3} \begin{bmatrix} \cos \theta_{i+1} \\ \sin \theta_{i+1} \end{bmatrix} \end{aligned} \quad (4.10)$$

$$\begin{aligned} a_0 &= \frac{6 \cdot (P_3 - P_0) \cdot V_0^T \cdot V_1 \cdot V_1^T - 3 \cdot (P_3 - P_0) \cdot V_1^T \cdot V_0 \cdot V_1}{4 \cdot V_0 \cdot V_0^T \cdot V_1 \cdot V_1^T - (V_0 \cdot V_1^T)^2} \\ a_1 &= \frac{3 \cdot (P_3 - P_0) \cdot V_0^T \cdot V_0 \cdot V_1^T - 6 \cdot (P_3 - P_0) \cdot V_1^T \cdot V_0 \cdot V_0^T}{(V_0 \cdot V_1^T)^2 - 4 \cdot V_0 \cdot V_0^T \cdot V_1 \cdot V_1^T} \end{aligned} \quad (4.11)$$

However, as explained by Yong and Cheng [62], the analytic expressions of equation 4.11 can result in negative values for  $a_0$  and  $a_1$ . In that cases, the parametric curve defined by the control points will not respect the orientation of the initial and/or final poses of that segment, which is not desired. Moreover, they demonstrate that the equation 4.11 results in positive values for  $a_0$  and  $a_1$  when the angle between  $V_0$  and  $\overrightarrow{P_0 P_3}$ , and angle between  $V_1$  and  $\overrightarrow{P_0 P_3}$  are both smaller than  $\arccos \frac{1}{3}$ . In their paper, Yong and Cheng present several methods for defining middle poses using trigonometry to validate their algorithm for the entire angle space of  $[-\pi, \pi] \times [-\pi, \pi]$ . Nevertheless, considering that in the current application, almost all the poses are oriented towards the subsequent pose, the values of  $a_0$  and  $a_1$  determined by equation 4.11 are valid in most of the cases. Consequently, for the small number of cases where the method is not valid, a simple approach can be considered. This involves defining  $a_0$  and  $a_1$  as the distance between the initial and final poses,  $\|P_3 - P_0\|$ .

Finally, since the algorithm must be executed at each iteration, additional checks are necessary. First, an obstacle is considered valid if its expanded convex hull intersects with the main path. Upon encountering a valid obstacle, two sets of pose sequences are determined, and respective paths are generated. Each path begins with the robot's current pose and includes only the poses ahead. In other words, only the poses projected further along the main path than the robot's current position are considered.

Moreover, to prevent noise from obstacle detection affecting path generation, the new path is only considered if the previous one collides with the current expanded convex hull, or if the new path is significantly shorter. The first condition can be checked by verifying the intersection between each edge of the convex hull and the line segments connecting each pose in the path. The second condition is parameterized by either a constant minimum difference in length between the previous and new paths or a fraction of the previous path's length.

The decision between the two paths, above or below the main path, is determined by the robot's position relative to the main path. In the event that the robot is situated on the main path, the shorter path is selected. In the remaining two cases, only the path on the corresponding side is considered. The algorithm terminates when the robot either reaches the end of the path and returns to the main path or deviates from it a certain distance. A possible implementation of a numerical algorithm to determine the length of a spline is presented in appendix C, in the algorithm 10.

## 4.4 Simulation Results

Only simulations were performed regarding the collision avoidance algorithms. The simulations were performed in MATLAB, both for the virtual security zone and obstacle avoidance algorithms. Moreover, as quantitative results of the algorithms are not trivial to retrieve, only qualitative results are presented. Consequently, this video<sup>2</sup> demonstrates the variation of the virtual security zone for different linear and angular velocities of the robot. The parameterization of the algorithm in the simulation presented in the video is the following:

- **Robot:**

- Length:  $0.6m$
- Width:  $0.4m$
- Distance from center to front:  $0.3m$
- Distance from center to left:  $0.2m$

- **Algorithm:**

- Delta time ( $\Delta t$ ):  $0.1s$
- Number of control cycles to consider for security distance ( $k_1$ ): 2
- Linear constant for security distance ( $k_2$ ): 0.4
- Minimum security distance ( $D_{min}$ ):  $0.1m$

---

<sup>2</sup><https://youtu.be/kgks3F1DkAo>

- Number of control cycles to consider for security angle ( $k_3$ ): 2
- Linear constant for security angle ( $k_4$ ): 0.4
- Minimum security angle ( $\phi_{min}$ ):  $0^\circ$

In the video, the algorithm is tested by first varying the linear velocity from  $0 \text{ m/s}$  to a maximum value of  $1.0 \text{ m/s}$  with a constant acceleration of  $0.5 \text{ m/s}^2$ , and then decreasing it from  $1.0 \text{ m/s}$  back to  $0 \text{ m/s}$  with a constant deceleration of  $-0.5 \text{ m/s}^2$ . Following this, the algorithm is tested for variations in angular velocity: first by increasing the angular velocity from  $0 \text{ rad/s}$  to  $1.0 \text{ rad/s}$  with a constant angular acceleration of  $1.0 \text{ rad/s}^2$ , then by decreasing it from  $v \text{ rad/s}$  to  $-1.0 \text{ rad/s}$  with a constant angular deceleration of  $-1.0 \text{ rad/s}^2$ , and finally by increasing it from  $-1.0 \text{ rad/s}$  back to  $0 \text{ rad/s}$  with a constant angular acceleration of  $1.0 \text{ rad/s}^2$ . The video also demonstrates the algorithm being tested on a circular trajectory with a radius of  $1.0 \text{ m}$ , maintaining a constant linear velocity of  $0.5 \text{ m/s}$  and an angular velocity of  $0 \text{ rad/s}$ . Additionally, the algorithm is tested under the same conditions but with an angular velocity of  $0.5 \text{ rad/s}$  to illustrate its behavior in more complex motion scenarios.

Regarding the obstacle avoidance algorithm, this was tested for several obstacle configurations, as presented in this video<sup>3</sup>. Moreover, the following parametrization was used for the simulations performed:

- **LiDAR:**

- Distance range:  $0.1\text{--}6\text{m}$
- Angular field of view:  $360^\circ$
- Angular resolution:  $1^\circ$
- Noise standard deviation: 0.02

- **Obstacle detection:**

- Projection distance for the boundary points ( $d_{back}$ ):  $1\text{m}$
- Distance of expansion of the convex hull:  $0.5\text{m}$
- Tolerance of the RDP algorithm: 0.2
- Tolerance for nearly collinear edges:  $5^\circ$
- Point cloud buffer size: 5

- **Path generation:**

- Distance of expansion of the vertices ( $d_1$ ):  $0.3\text{m}$
- Distance for entry and exit points in the main path ( $d_2$ ):  $1.0\text{m}$
- Condition to use the new path due to length:  $l_{new} < \frac{3}{4} \cdot l_{old}$  and  $l_{old} - l_{new} > 0.5\text{m}$

In the simulations conducted, a line segment was utilized as the main path. Furthermore, each point cloud retrieved from the LiDAR sensor exclusively contains points associated with a single obstacle. Additionally, three of the setups presented in the video are illustrated in figures 4.7, 4.8,

---

<sup>3</sup><https://youtu.be/yeX7hJUrwPU>

and 4.9. Each figure contains four frames from their respective simulations, carefully selected to best illustrate the algorithm's performance. The setup in figure 4.7 includes two obstacles, one square and one rectangular, connected to form a more complex and irregular shape. While the robot follows the main path, only part of the obstacle is detected by the LiDAR, so the expanded convex hull does not encompass the entire obstacle. However, as the robot begins to navigate around the obstacle along the initially generated path, new points are detected by the LiDAR. Consequently, the path is updated to avoid collisions using the new obstacle convex hull.

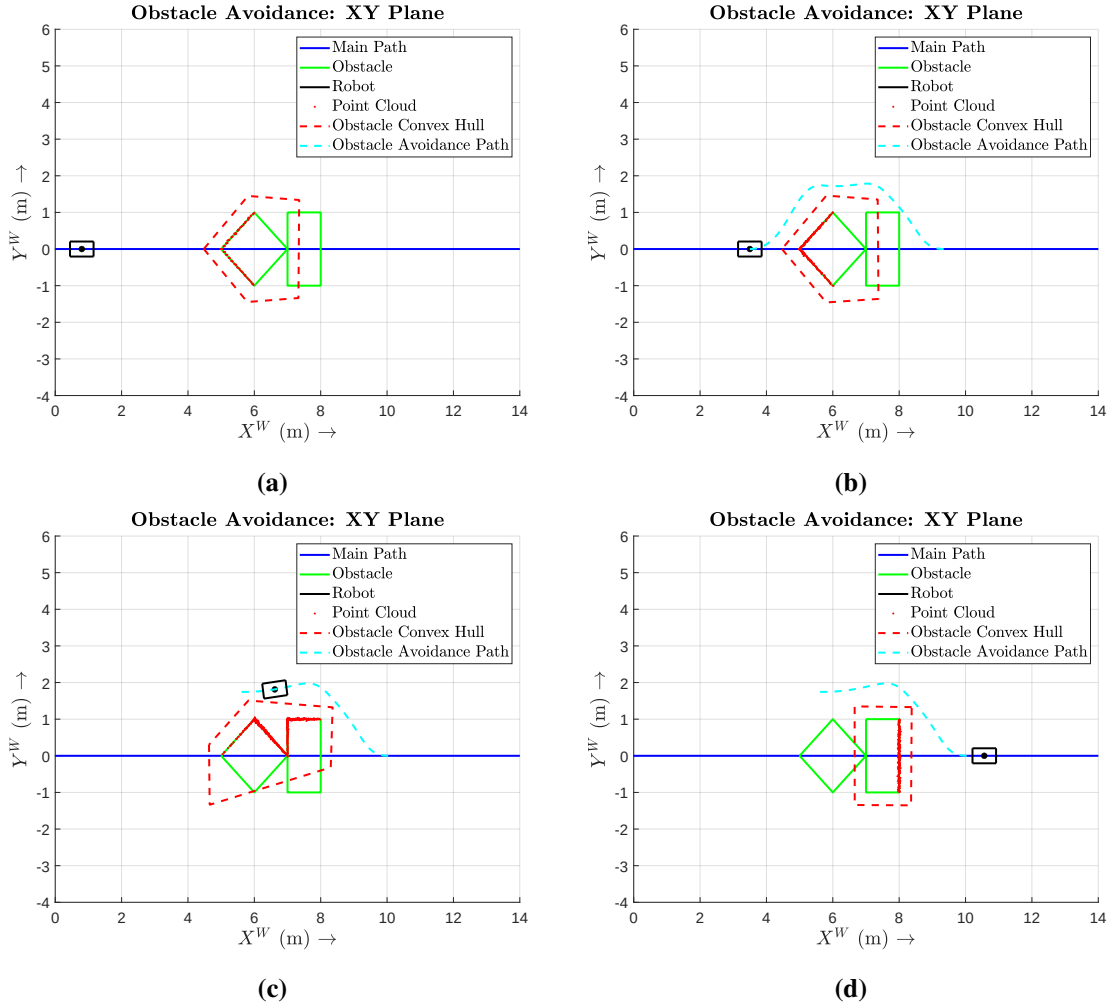


**Figure 4.7:** Obstacle Avoidance: Setup 1

Figure 4.8 illustrates a different algorithm component. In this setup, the height of the rectangle was increased so that the path above and below the main trajectory had a different length. As a result, while in the main path, the LiDAR was able to detect part of the rectangular obstacle. Furthermore, the generated path to contour the obstacle below resulted in a relatively short length. Consequently, the algorithm successfully selected the shortest path.

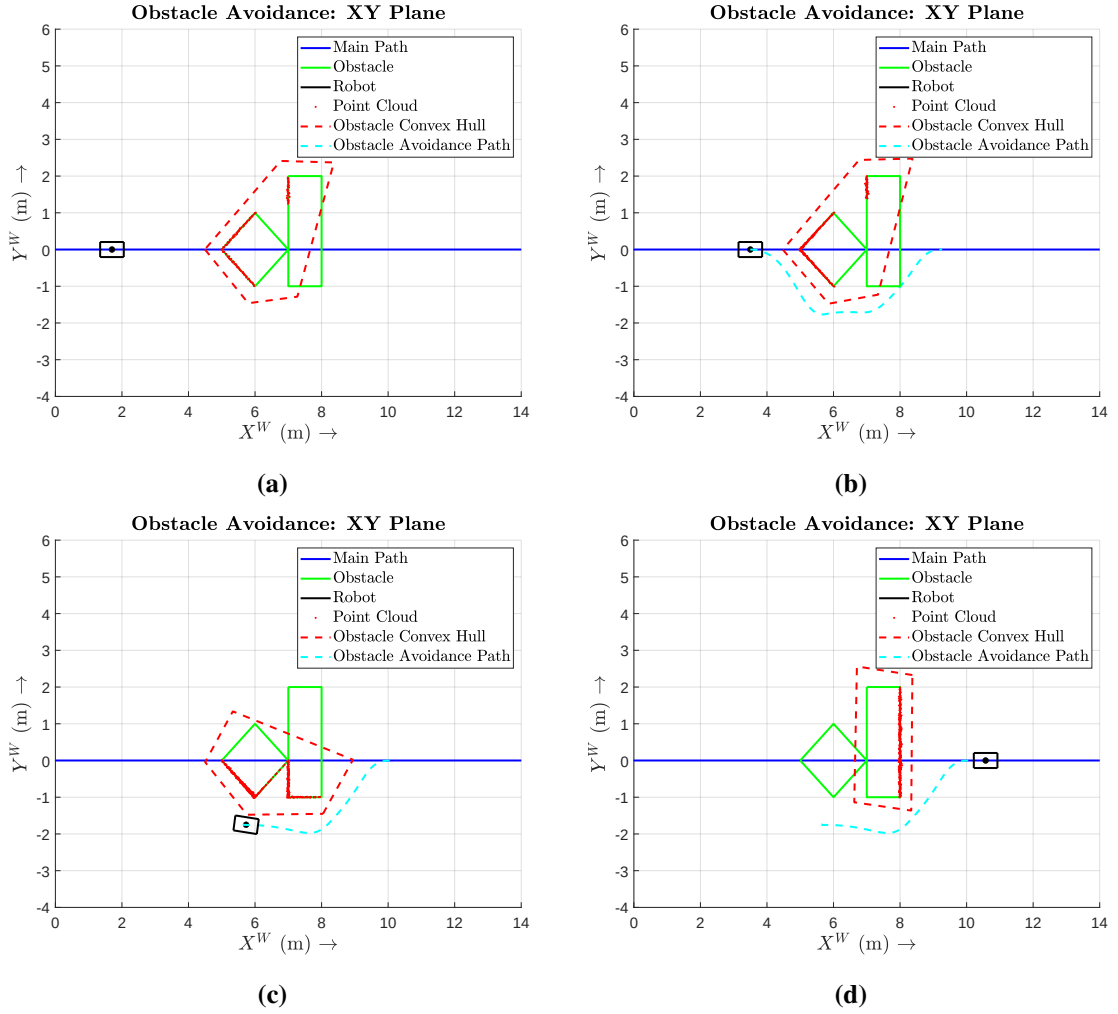


**Figure 4.8:** Obstacle Avoidance: Setup 2

Figure 4.9 illustrates the algorithm's behavior regarding the determination of the entry and leaving points of the main path. In this setup, a trapezoidal obstacle was used to demonstrate how the algorithm efficiently selects the exit point of the main path, taking into account the convex hull dimensions and shape. In particular, as the obstacle is larger below the main path, the generated path below will be longer than the path above. Furthermore, the closest projected points of the vertices above and below the main path are significantly different. Consequently, the first pose of each set, above and below, will differ. Therefore, the verification of a close obstacle is performed before reaching any of those poses, so that both the path above and below can be performed if necessary. Moreover, by using the respective exit pose of each set, each path will deviate from the main path only when necessary, and not earlier. On the other hand, the entry point in the main path was also updated at the end of the simulation, as the LiDAR provided additional information regarding the obstacle's posterior region, resulting in the generation of a shorter path.

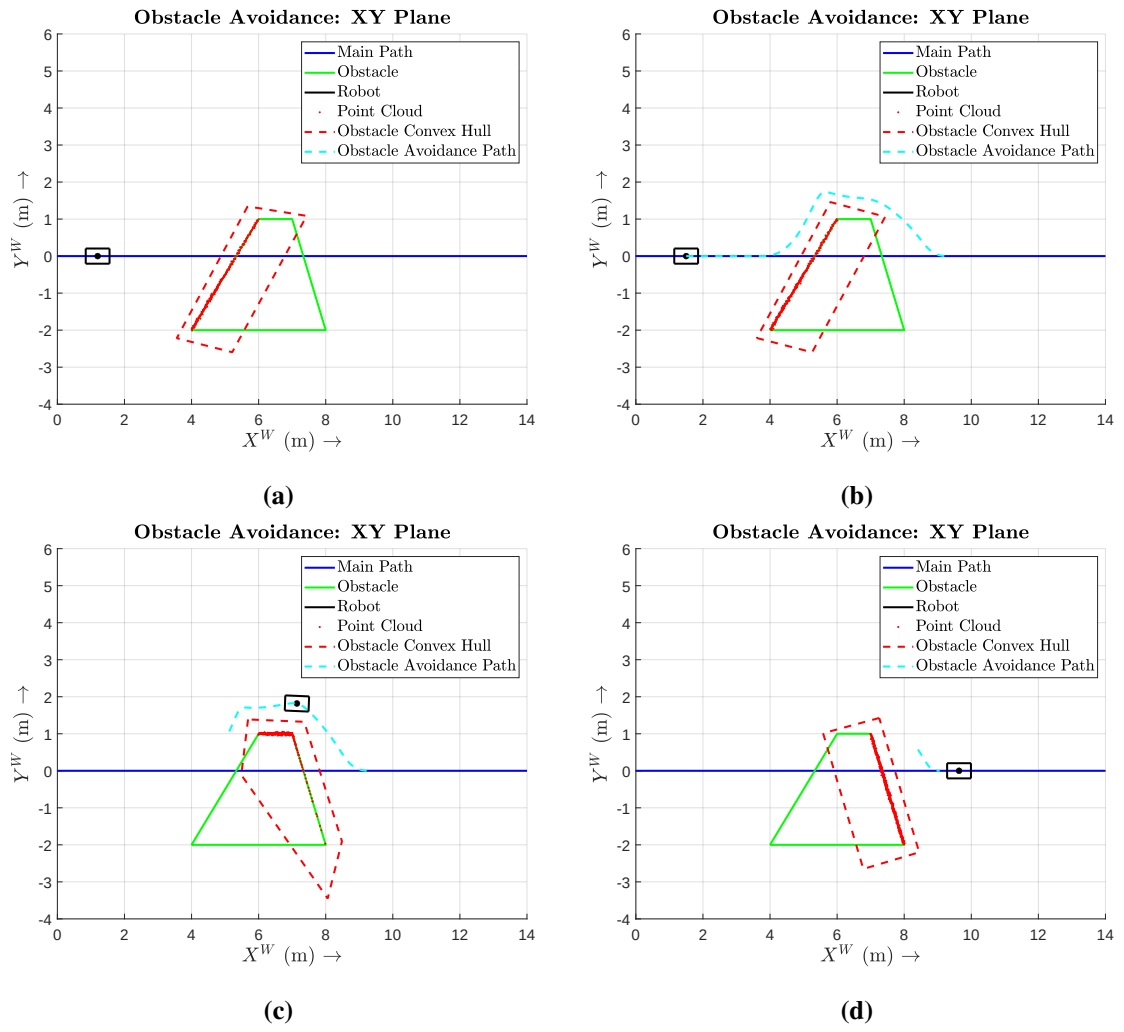


Figure 4.9: Obstacle Avoidance: Setup 3

## 4.5 Discussion

This chapter outlines the proposed techniques for collision avoidance and obstacle avoidance. Initially, it presents the formulation of the collision avoidance method, which relies on establishing virtual security zones. Subsequently, it introduces the obstacle avoidance algorithm, comprising both obstacle detection and path generation components. These methods are designed to be mutually compatible, allowing simultaneous use.

In terms of results, only simulations were conducted. Firstly, the virtual security zone method was tested across various robot velocity configurations. It was concluded that the algorithm's parameterization is suitable for generating security zones for diverse motions, including complex movements such as rotation combined with linear motion in omnidirectional robots.

Moreover, the obstacle avoidance method was tested across multiple obstacle configurations, successfully generating smooth paths without intersecting the obstacle. Incorporating techniques like buffering for the point cloud and employing the RDP algorithm for convex hull simplification proved effective in generating paths while mitigating noise from LiDAR measurements. However, an extra algorithm is still required to filter and cluster LiDAR measurements into distinct point clouds, ensuring that each one exclusively comprises points pertaining to a single obstacle. Furthermore, the proposed algorithms benefit from their systematic approach, which is particularly advantageous in industrial environments where robust and predictable performance is crucial.

## Chapter 5

# Conclusion

The initial objective of this MSc thesis was to study and develop a trajectory tracking controller with predictive capabilities and a collision avoidance module in order to enable a mobile robot to contour stationary obstacles. Consequently, three literature reviews were conducted in the fields of trajectory tracking, trajectory definition, and collision avoidance with particular emphasis on their application to mobile robotics. Considering the literature review, a controller architecture based on a Nonlinear Model Predictive Control (NMPC) was developed for the trajectory tracking problem. With regard to the issue of collision avoidance, an algorithm was proposed that utilizes an LiDAR to detect obstacles and generates smooth trajectories using parametric curves, specifically splines.

For the trajectory tracking problem, the proposed controller was compared to other controllers from the literature, both in simulations and real-world environments. Specifically, comparisons were made with a generic PD-only controller, a Feedforward controller [16], and a NMPC controller incorporating optimization constraints, developed using the open-source libraries CasADi [57] and Ipopt [58]. The proposed controller demonstrated superior accuracy compared to the PD-only and Feedforward controllers, resulting in smaller position and orientation errors. Additionally, it was found that the PI controller for the robot wheels can compensate for some of the system's nonlinearities. This allows for the use of a simple kinematic model when the robot has fast dynamics (time constants around 0.1 s) and a first-order approximation dynamic model for slow robot dynamics (time constants  $\geq 0.3$  s). Furthermore, comparisons with an NMPC with optimization constraints revealed that handling velocity and acceleration inequality constraints outside the optimization process significantly reduces processing time. This makes it feasible to use NMPC controllers in robots with limited computational power.

With regard to the collision avoidance algorithm, only simulations were conducted. The efficacy of the virtual security zone method was evaluated through a series of experiments, wherein it was demonstrated to be an effective approach for the generation of security zones for a range of diverse and complex motions. The obstacle avoidance method successfully generate smooth paths in multiple obstacle configurations, with the use of techniques such as point cloud buffering and the RDP algorithm for noise reduction. Moreover, the systematic approach of these algorithms is advantageous for industrial environments, as it ensures robust and predictable performance.

## 5.1 Future Work

Concerning future work, improvements and developments could still be made over this MSc thesis. These improvements are relative to both trajectory tracking and collision avoidance.

With regard to trajectory tracking, the proposed controller was subjected to testing solely with a single omnidirectional mobile robot, thereby limiting the scope of the tests to the robot's inherent characteristics. In particular, the maximum velocity of the robot was observed to be  $0.6 \text{ m/s}$ , which is relatively low. Thus, further testing with additional robots would permit more precise validation, encompassing higher velocities and diverse robot dynamics. Moreover, although the developed controller was designed with omnidirectional mobile robots in mind, the implemented code is compatible with other robot models. Consequently, models for other robot steering geometries can be introduced and tested using the developed framework. Furthermore, future work could be extended in other directions by exploring the incorporation of adaptive and robust control methodologies into the proposed controller. Such capabilities could enhance the controller's performance by addressing issues related to uncertainties.

Regarding collision avoidance, it should be noted that only simulations were conducted. Consequently, the proposed algorithms still require validation in real-world environments. To ensure the robustness and reliability of the algorithms, future work should include extensive testing in various real-world scenarios to assess their performance and adaptability under different conditions. Moreover, enhancements can also be made to the obstacle avoidance algorithm, particularly in the area of obstacle detection. The algorithm presented in the thesis assumes a preliminary filtering of the point cloud, with the objective of only receiving points associated with the same obstacle. Consequently, the implementation of such an algorithm can be achieved through the utilization of point cloud segmentation techniques, such as clustering. Furthermore, the velocity of the robot can be considered in order to reduce the distortion of the point cloud associated with the robot's motion.

# References

- [1] Global autonomous mobile robots market report. <https://www.researchandmarkets.com/>, 2024. Grand View Research Report.
- [2] Using amrs in warehouses: Productivity benefits. <https://www.mckinsey.com/mgi/our-research/all-research>, 2024. McKinsey Global Institute Report.
- [3] M. Wu, C. F. Yeong, E. L. Su, W. Holderbaum, and C. Yang. A review on energy efficiency in autonomous mobile robots. *Robotic Intelligence and Automation*, 43(6):648–668, 2023. DOI: 10.1108/RIA-05-2023-0060.
- [4] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*, volume 23. MIT press, 2011.
- [5] S. G. Tzafestas. Mobile robot control and navigation: A global overview. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 91:35–58, 2018. DOI: 10.1007/s10846-018-0805-9.
- [6] B. K. Patle, G. L. Babu, A. Pandey, D. R. Parhi, and A. Jagadeesh. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15:582–606, 2019. DOI: 10.1016/j.dt.2019.04.011.
- [7] J. D. Ribeiro, H. M. Sobreira, and A. P. Moreira. A nonlinear model predictive control strategy for trajectory tracking of omnidirectional robots. 2024.
- [8] A. P. Aguiar and J. P. Hespanha. Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. *IEEE Transactions on Automatic Control*, 52:1362–1379, 2007. DOI: 10.1109/TAC.2007.902731.
- [9] B. Kouvaritakis and M. Cannon. *Non-linear Predictive Control: theory and practice*. Institute of Engineering and Technology, London, United Kingdom, 2001. DOI: 10.1049/pbce061e.
- [10] K. Watanabe, Y. Shiraishi, S. G. Tzafestas, J. Tang, and T. Fukuda. Feedback control of an omnidirectional autonomous platform for mobile service robots. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 22:315–330, 1998. DOI: 10.1023/a:1008048307352.
- [11] T. Li, S. Jian, and M. Tsai. Design and implementation of fuzzy trajectory following and planning control for mobile robots. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology. TENCON 2001 (Cat. No. 01CH37239)*, volume 1, pages 453–458. IEEE, 2001. DOI: 10.1109/tencon.2001.949634.

- [12] Y. Liang, L. Xu, R. Wei, and H. Hu. Adaptive fuzzy control for trajectory tracking of mobile robot. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4755–4760. IEEE, 2010. DOI: [10.1109/IROS.2010.5651060](https://doi.org/10.1109/IROS.2010.5651060).
- [13] I. Benaoumeur, B. Laredj, H. E. Reda, and A. F. Zoubir. Backstepping approach for autonomous mobile robot trajectory tracking. *Indonesian Journal of Electrical Engineering and Computer Science*, 2, 2016. DOI: [10.11591/ijeecs.v2.i3.pp478-485](https://doi.org/10.11591/ijeecs.v2.i3.pp478-485).
- [14] J. Mu, X. Yan, B. Jiang, S. K. Spurgeon, and Z. Mao. Sliding mode control for a class of nonlinear systems with application to a wheeled mobile robot. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 4746–4751. IEEE, 2015.
- [15] W. Jianbin and C. Jianping. An adaptive sliding mode controller for four-wheeled omnidirectional mobile robot with input constraints. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 5591–5596. IEEE, 2019. DOI: [10.1109/CCDC.2019.8832894](https://doi.org/10.1109/CCDC.2019.8832894).
- [16] R. B. Sousa, P. G. Costa, and A. P. Moreira. A pose control algorithm for omnidirectional robots. In *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 91–96. IEEE, 2021. DOI: [10.1109/ICARSC52212.2021.9429803](https://doi.org/10.1109/ICARSC52212.2021.9429803).
- [17] B. Käpernick. *Gradient-based nonlinear model predictive control with constraint transformation for fast dynamical systems*. PhD thesis, Universität Ulm, 2016.
- [18] M. Diehl, H. J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. *Nonlinear model predictive control: towards new challenging applications*, 384:391–417, 2009. DOI: [10.1007/978-3-642-01094-1\\_32](https://doi.org/10.1007/978-3-642-01094-1_32).
- [19] A. S. Conceição, A. P. Moreira, and P. J. Costa. A nonlinear model predictive control strategy for trajectory tracking of a four-wheeled omnidirectional mobile robot. *Optimal Control Applications and Methods*, 29:335–352, 2008. DOI: [10.1002/oca.827](https://doi.org/10.1002/oca.827).
- [20] J. R. Ferreira and A. P. Moreira. Non-linear model predictive controller for trajectory tracking of an omni-directional robot using a simplified model. In *9th Portuguese conference on automatic control*, pages 57–62, 2010.
- [21] Y. Pei, K. Zhang, J. Pan, and Y. Shi. Nonlinear model predictive tracking control of non-holonomic wheeled mobile robot using modified c/gmres algorithm. In *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 6298–6303. IEEE, 2017. DOI: [10.1109/IECON.2017.8217095](https://doi.org/10.1109/IECON.2017.8217095).
- [22] T. Ohtsuka. A continuation/gmres method for fast computation of nonlinear receding horizon control. *Automatica*, 40:563–574, 2004. DOI: [10.1016/j.automatica.2003.11.005](https://doi.org/10.1016/j.automatica.2003.11.005).
- [23] T. P. Nascimento, C. E. Dórea, and L. M. Gonçalves. Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach. *International Journal of Advanced Robotic Systems*, 15, 2018. DOI: [10.1177/1729881418760461](https://doi.org/10.1177/1729881418760461).
- [24] C. Igel and M. Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003. DOI: [10.1016/S0925-2312\(01\)00700-7](https://doi.org/10.1016/S0925-2312(01)00700-7).
- [25] M. K. Ouach, T. Eren, and E. Özcan. Prm path smoothening by circular arc fillet method for mobile robot navigation. *Uluslararası Muhendislik Arastirma ve Gelistirme Dergisi*, 16:1–19, 2023. DOI: [10.29137/umagd.1278980](https://doi.org/10.29137/umagd.1278980).

- [26] C. De Boor. A practical guide to splines. *Mathematics of Computation*, 27, 1978. DOI: [10.2307/2006241](https://doi.org/10.2307/2006241).
- [27] P. M. Prenter. *Splines and Variational Methods*, volume 56. Dover Publications, 2008.
- [28] C. Sprunk. Planning motion trajectories for mobile robots using splines. *University of Freiburg*, 2008.
- [29] V. Munoz, A. Ollero, M. Prado, and A. Simon. Mobile robot trajectory planning with dynamic and kinematic constraints. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, volume 4, pages 2802–2807. IEEE, 1994. DOI: [10.1109/robot.1994.350914](https://doi.org/10.1109/robot.1994.350914).
- [30] T. N. Goodman. Properties of  $\beta$ -splines. *Journal of Approximation Theory*, 44:132–153, 1985. DOI: [10.1016/0021-9045\(85\)90076-0](https://doi.org/10.1016/0021-9045(85)90076-0).
- [31] H. Wang, J. Kearney, and K. Atkinson. Robust and efficient computation of the closest point on a spline curve. In *Proceedings of the 5th International Conference on Curves and Surfaces*, pages 397–406, 2002.
- [32] X. Chen, Y. Zho, Z. Shu, and H. Su. Improved algebraic algorithm on point projection for bézier curves. In *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, pages 158–163. IEEE, 2008. DOI: [10.1109/imscs.2007.17](https://doi.org/10.1109/imscs.2007.17).
- [33] J. Wang and D. Herath. *What Makes Robots? Sensors, Actuators, and Algorithms*, chapter 4, pages 177–203. Springer, 2022. DOI: [10.1007/978-981-19-1983-1\\_7](https://doi.org/10.1007/978-981-19-1983-1_7).
- [34] A. Discant, A. Rogozan, C. Rusu, and A. Bensrhair. Sensors for obstacle detection - a survey. In *2007 30th International Spring Seminar on Electronics Technology (ISSE)*, pages 100–105. IEEE, 2007. DOI: [10.1109/ISSE.2007.4432828](https://doi.org/10.1109/ISSE.2007.4432828).
- [35] N. Wessendorp, R. Dinaux, J. Dupeyroux, and G. C. De Croon. Obstacle avoidance onboard mavs using a fmcw radar. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 117–122. IEEE, 2021. DOI: [10.1109/IROS51168.2021.9635901](https://doi.org/10.1109/IROS51168.2021.9635901).
- [36] S. Thongchai and K. Kawamura. Application of fuzzy control to a sonar-based obstacle avoidance mobile robot. In *Proceedings of the 2000. IEEE International Conference on Control Applications. Conference Proceedings (Cat. No. 00CH37162)*, volume 1, pages 425–430. IEEE, 2000. DOI: [10.1109/cca.2000.897461](https://doi.org/10.1109/cca.2000.897461).
- [37] A. N. Rafai, N. Adzhar, and N. I. Jaini. A review on path planning and obstacle avoidance algorithms for autonomous mobile robots. *Journal of Robotics*, 2022:1–14, 2022. DOI: [10.1155/2022/2538220](https://doi.org/10.1155/2022/2538220).
- [38] C. Cheng, Q. Sha, B. He, and G. Li. Path planning and obstacle avoidance for auv: A review. *Ocean Engineering*, 235:109355, 2021. DOI: [10.1016/j.oceaneng.2021.109355](https://doi.org/10.1016/j.oceaneng.2021.109355).
- [39] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4:100–107, 1968. DOI: [10.1109/TSSC.1968.300136](https://doi.org/10.1109/TSSC.1968.300136).
- [40] C. W. Warren. Global path planning using artificial potential fields. In *1989 IEEE International Conference on Robotics and Automation*, pages 316–317. IEEE Computer Society, 1989. DOI: [10.1109/ROBOT.1989.100007](https://doi.org/10.1109/ROBOT.1989.100007).

- [41] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. DOI: [10.1007/BF01386390](https://doi.org/10.1007/BF01386390).
- [42] L. G. Veras, F. L. Medeiros, and L. N. Guimaraes. Systematic literature review of sampling process in rapidly-exploring random trees. *IEEE Access*, 7:50933–50953, 2019. DOI: [10.1109/ACCESS.2019.2908100](https://doi.org/10.1109/ACCESS.2019.2908100).
- [43] P. Sudhakara and V. Ganapathy. Trajectory planning of a mobile robot using enhanced a-star algorithm. *Indian Journal of Science and Technology*, 9, 2016. DOI: [10.17485/ijst/2016/v9i41/93816](https://doi.org/10.17485/ijst/2016/v9i41/93816).
- [44] J. Santos, P. Costa, L. F. Rocha, A. P. Moreira, and G. Veiga. Time enhanced a\*: Towards the development of a new approach for multi-robot coordination. In *2015 IEEE international conference on industrial technology (ICIT)*, pages 3314–3319. IEEE, 2015. DOI: [10.1109/ICIT.2015.7125589](https://doi.org/10.1109/ICIT.2015.7125589).
- [45] S. M. Rostami, A. K. Sangaiah, J. Wang, and X. Liu. Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *Eurasip Journal on Wireless Communications and Networking*, 2019, 2019. DOI: [10.1186/s13638-019-1396-2](https://doi.org/10.1186/s13638-019-1396-2).
- [46] M. R. Bachute and J. M. Subhedar. Autonomous driving architectures: Insights of machine learning and deep learning algorithms. *Machine Learning with Applications*, 6:100164, 2021. DOI: [10.1016/j.mlwa.2021.100164](https://doi.org/10.1016/j.mlwa.2021.100164).
- [47] M. Duguleana and G. Mogan. Neural networks based reinforcement learning for mobile robots obstacle avoidance. *Expert Systems with Applications*, 62, 2016. DOI: [10.1016/j.eswa.2016.06.021](https://doi.org/10.1016/j.eswa.2016.06.021).
- [48] K. Maček, I. Petrović, and N. Perić. A reinforcement learning approach to obstacle avoidance of mobile robots. In *7th International Workshop on Advanced Motion Control. Proceedings (Cat. No. 02TH8623)*, pages 462–466. IEEE, 2002. DOI: [10.1109/amc.2002.1026964](https://doi.org/10.1109/amc.2002.1026964).
- [49] Y. Li and X. Chen. Mobile robot navigation using particle swarm optimization and adaptive nn. In *Advances in Natural Computation*, volume 3612, pages 628–631. Springer, 2005. DOI: [10.1007/11539902\\_76](https://doi.org/10.1007/11539902_76).
- [50] J. K. Pothal and D. R. Parhi. Navigation of multiple mobile robots in a highly clutter terrains using adaptive neuro-fuzzy inference system. *Robotics and Autonomous Systems*, 72, 2015. DOI: [10.1016/j.robot.2015.04.007](https://doi.org/10.1016/j.robot.2015.04.007).
- [51] R. B. Sousa. Odometry and extrinsic sensor calibration on mobile robots. Master’s thesis, Faculty of Engineering of the University of Porto (FEUP), INESC TEC – Institute for Systems and Computer Engineering, Technology and Science, Porto, Portugal, 2020. DOI: [10.13140/RG.2.2.27052.28802](https://doi.org/10.13140/RG.2.2.27052.28802).
- [52] R. B. Sousa, M. R. Petry, P. G. Costa, and A. P. Moreira. Optiodom: a generic approach for odometry calibration of wheeled mobile robots. *Journal of Intelligent & Robotic Systems*, 105:39, 2022. DOI: [10.1007/s10846-022-01630-3](https://doi.org/10.1007/s10846-022-01630-3).
- [53] H. F. Oliveira, A. J. Sousa, A. P. Moreira, and P. J. Costa. Precise modeling of a four wheeled omni-directional robot. In *Proceedings of the 8th conference on autonomous robot systems and competitions*. Citeseer, 2008.

- [54] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591. IEEE, 1993. DOI: [10.1109/ICNN.1993.298623](https://doi.org/10.1109/ICNN.1993.298623).
- [55] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The computer journal*, 7:149–154, 1964. DOI: [10.1093/comjnl/7.2.149](https://doi.org/10.1093/comjnl/7.2.149).
- [56] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue française d’informatique et de recherche opérationnelle. Série rouge*, 3:35–43, 1969.
- [57] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019. DOI: [10.1007/s12532-018-0139-4](https://doi.org/10.1007/s12532-018-0139-4).
- [58] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006. DOI: [10.1007/s10107-004-0559-y](https://doi.org/10.1007/s10107-004-0559-y).
- [59] H. Sobreira, C. M. Costa, I. Sousa, L. Rocha, J. Lima, P. C. Farias, P. Costa, and A. P. Moreira. Map-matching algorithms for robot self-localization: a comparison between perfect match, iterative closest point and normal distributions transform. *Journal of Intelligent & Robotic Systems*, 93:533–546, 2019. DOI: [10.1007/s10846-017-0765-5](https://doi.org/10.1007/s10846-017-0765-5).
- [60] S. T. Pfister, S. I. Roumeliotis, and J. W. Burdick. Weighted line fitting algorithms for mobile robot map building and efficient data representation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 1304–1311. IEEE, 2003. DOI: [10.1109/ROBOT.2003.1241772](https://doi.org/10.1109/ROBOT.2003.1241772).
- [61] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1:244–256, 1972. DOI: [10.1016/S0146-664X\(72\)80017-0](https://doi.org/10.1016/S0146-664X(72)80017-0).
- [62] J. Yong and F. F. Cheng. Geometric hermite curves with minimum strain energy. *Computer Aided Geometric Design*, 21:281–301, 2004. DOI: [10.1016/j.cagd.2003.08.003](https://doi.org/10.1016/j.cagd.2003.08.003).
- [63] C. Igel and M. Hüsken. Improving the rprop learning algorithm. In *Proceedings of the Second International Symposium on Neural Computation, ICSC Academic Press, US*, pages 115–121, 2000.
- [64] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons — from backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16:265–278, 1994. DOI: [10.1016/0920-5489\(94\)90017-5](https://doi.org/10.1016/0920-5489(94)90017-5).
- [65] R. Battiti. First- and second-order methods for learning: Between steepest descent and newton’s method. *Neural Computation*, 4:141–166, 1992. DOI: [10.1162/neco.1992.4.2.141](https://doi.org/10.1162/neco.1992.4.2.141).
- [66] M. Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2018.

# Appendix A

## Gradient Descent Methods

Gradient descent methods are a class of optimization algorithms used to minimize a function by iteratively moving in the direction of its steepest descent. These methods are widely applied in various optimization problems, including those encountered in nonlinear model predictive control. This chapter explores several commonly used gradient descent methods and evaluates their performance within the context of nonlinear model predictive control.

Sections A.1, A.2, and A.3 detail the algorithms for each gradient descent method. Finally, Section A.4 presents a comparative study of these algorithms, focusing on convergence speed and accuracy, to determine the most effective method for solving an NMPC optimization problem.

### A.1 Resilient Propagation (Rprop)

Resilient Propagation (Rprop) is an iterative optimization algorithm introduced by Riedmiller and Braun [54] for training artificial neural networks. Renowned for its simplicity and efficiency in high-dimensional parameter spaces, Rprop updates neural network weights based solely on the signs of the partial derivatives of the error function, disregarding their magnitude.

As a first-order optimization algorithm, Rprop requires only the first derivative of the cost function and can be applied to neural networks of any topology as well as other optimization problems. Its step size is independent of the gradient's magnitude, enhancing its robustness.

Rprop's main advantages include its simplicity, accuracy, robustness, and fast convergence speed [63]. Over time, three new approaches have been developed based on the original Rprop, further improving its performance and applicability.

#### A.1.1 Rprop with Weight-Backtracking

As previously mentioned, Riedmiller and Braun created the Rprop algorithm in 1993 [54]. This algorithm introduced the parameter  $\Delta_i$ , replacing the traditional gradient descent magnitude. The core principle is straightforward: if the partial derivative of the cost function,  $\partial f / \partial x_i$ , retains the same sign over successive iterations, the step size increases; if the signs differ, the step size

decreases. Moreover, these step sizes are bounded by  $\Delta_{min}$  and  $\Delta_{max}$ , and the update factors  $\eta^-$  and  $\eta^+$  must satisfy  $0 < \eta^- < 1 < \eta^+$ . This logic is illustrated in equation A.1.

$$\Delta_i^{(k)} = \begin{cases} \eta^+ \cdot \Delta_i^{(k-1)} & , \text{ if } \frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} > 0 \\ \eta^- \cdot \Delta_i^{(k-1)} & , \text{ if } \frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} < 0 \\ \Delta_i^{(k-1)} & , \text{ otherwise} \end{cases} \quad (\text{A.1})$$

Subsequently,  $\Delta_i$  is utilized to adjust the step of the variable being estimated,  $x_i$ . This adjustment relies on the first derivative of the cost function, similar to the gradient descent approach. However, it solely considers the sign of the partial derivative, as shown in equation A.2:

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_i^{(k)}, \text{ where } \Delta x_i^{(k)} = -\text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)} \quad (\text{A.2})$$

In the variant known as Rprop with weight-backtracking (referred to as Rprop+ in [63]), the step size,  $\Delta x_i$ , for the variable  $x_i$  reverses its direction from the previous iteration if the partial derivative of the cost function changes sign between steps. The implementation of the Rprop+ algorithm is outlined in algorithm 2.

---

**Algorithm 2:** Rprop with Weight-Backtracking (Rprop+)

---

**Input:**  $\Delta^{(k-1)}, x^{(k)}, \nabla f(x)^{(k)}, \nabla f(x)^{(k-1)}$

**Data:**  $\Delta_{min}, \Delta_{max}, \eta^-, \eta^+$

**Output:**  $\Delta^{(k)}, x^{(k+1)}$

```

1 foreach  $x_i$  do
2   if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} > 0$  then
3      $\Delta_i^{(k)} := \min(\Delta_i^{(k-1)} \cdot \eta^+, \Delta_{max})$ 
4      $\Delta x_i^{(k)} := -\text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)}$ 
5   else if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} < 0$  then
6      $\Delta_i^{(k)} := \max(\Delta_i^{(k-1)} \cdot \eta^-, \Delta_{min})$ 
7      $\Delta x_i^{(k)} := -\Delta x_i^{(k-1)}$ 
8      $\frac{\partial f}{\partial x_i}^{(k)} := 0$ 
9   else
10     $\Delta_i^{(k)} := \Delta_i^{(k-1)}$ 
11     $\Delta x_i^{(k)} := -\text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)}$ 
12  end
13   $x_i^{(k+1)} := x_i^{(k)} + \Delta x_i^{(k)}$ 
14 end

```

---

### A.1.2 Rprop without Weight-Backtracking

In Riedmiller's subsequent work [64] in 1994, a modified version of the original Rprop algorithm is proposed, where weight-backtracking is not considered. Therefore, in this new algorithm, the update is always defined by the equation A.2, in contrast to the first version. The algorithm proposed in Riedmiller [64] (named Rprop- in [63]) can be implemented as described in algorithm 3.

---

**Algorithm 3:** Rprop without Weight-Backtracking (Rprop-)
 

---

**Input:**  $\Delta^{(k-1)}, x^{(k)}, \nabla f(x)^{(k)}, \nabla f(x)^{(k-1)}$

**Data:**  $\Delta_{min}, \Delta_{max}, \eta^-, \eta^+$

**Output:**  $\Delta^{(k)}, x^{(k+1)}$

```

1 foreach  $x_i$  do
2   if  $\frac{\partial f^{(k-1)}}{\partial x_i} \cdot \frac{\partial f^{(k)}}{\partial x_i} > 0$  then
3      $\Delta_i^{(k)} := \min(\Delta_i^{(k-1)} \cdot \eta^+, \Delta_{max})$ 
4   else if  $\frac{\partial f^{(k-1)}}{\partial x_i} \cdot \frac{\partial f^{(k)}}{\partial x_i} < 0$  then
5      $\Delta_i^{(k)} := \max(\Delta_i^{(k-1)} \cdot \eta^-, \Delta_{min})$ 
6   else
7      $\Delta_i^{(k)} := \Delta_i^{(k-1)}$ 
8   end
9    $x_i^{(k+1)} := x_i^{(k)} - \text{sign}\left(\frac{\partial f^{(k)}}{\partial x_i}\right) \cdot \Delta_i^{(k)}$ 
10 end

```

---

### A.1.3 Improved Rprop with and without Weight-Backtracking

In the Rprop+ algorithm, weight-backtracking is always performed, regardless of whether the variable's update contributes to an increase or decrease in the cost function. To address this, Igel and Hüsken [63] suggested a modification where the step reversal occurs only if the cost function increased with the last update ( $f(x)^{(k)} > f(x)^{(k-1)}$ ). This enhancement resulted in the improved Rprop with weight-backtracking (iRprop+), detailed in algorithm 4.

Moreover, Igel and Hüsken [63] also refined the Rprop- variant. Their adaptation, named the improved Rprop without weight-backtracking (iRprop-), mirrors Rprop- with one key distinction: iRprop- sets the cost function's derivative to 0. While iRprop- shares similarities with iRprop+, it does not employ weight-backtracking. The algorithm can be implemented as described in algorithm 5.

**Algorithm 4:** Improved Rprop with Weight-Backtracking (iRprop+)**Input:**  $\Delta^{(k-1)}, x^{(k)}, \nabla f(x)^{(k)}, \nabla f(x)^{(k-1)}, f(x)^{(k-1)}, f(x)^{(k)}$ **Data:**  $\Delta_{min}, \Delta_{max}, \eta^-, \eta^+$ **Output:**  $\Delta^{(k)}, x^{(k+1)}$ 

```

1 foreach  $x_i$  do
2   if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} > 0$  then
3      $\Delta_i^{(k)} := \min(\Delta_i^{(k-1)} \cdot \eta^+, \Delta_{max})$ 
4      $\Delta x_i^{(k)} := -\text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)}$ 
5   else if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} < 0$  then
6      $\Delta_i^{(k)} := \max(\Delta_i^{(k-1)} \cdot \eta^-, \Delta_{min})$ 
7     if  $f(x)^{(k)} > f(x)^{(k-1)}$  then  $\Delta x_i^{(k)} := -\Delta x_i^{(k-1)}$ 
8     else  $\Delta x_i^{(k)} := 0$ 
9      $\frac{\partial f}{\partial x_i}^{(k)} := 0$ 
10  else
11     $\Delta_i^{(k)} := \Delta_i^{(k-1)}$ 
12     $\Delta x_i^{(k)} := -\text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)}$ 
13  end
14   $x_i^{(k+1)} := x_i^{(k)} + \Delta x_i^{(k)}$ 
15 end

```

**Algorithm 5:** Improved Rprop without Weight-Backtracking (iRprop-)**Input:**  $\Delta^{(k-1)}, x^{(k)}, \nabla f(x)^{(k)}, \nabla f(x)^{(k-1)}$ **Data:**  $\Delta_{min}, \Delta_{max}, \eta^-, \eta^+$ **Output:**  $\Delta^{(k)}, x^{(k+1)}$ 

```

1 foreach  $x_i$  do
2   if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} > 0$  then
3      $\Delta_i^{(k)} := \min(\Delta_i^{(k-1)} \cdot \eta^+, \Delta_{max})$ 
4   else if  $\frac{\partial f}{\partial x_i}^{(k-1)} \cdot \frac{\partial f}{\partial x_i}^{(k)} < 0$  then
5      $\Delta_i^{(k)} := \max(\Delta_i^{(k-1)} \cdot \eta^-, \Delta_{min})$ 
6      $\frac{\partial f}{\partial x_i}^{(k)} := 0$ 
7   else
8      $\Delta_i^{(k)} := \Delta_i^{(k-1)}$ 
9   end
10   $x_i^{(k+1)} := x_i^{(k)} - \text{sign}\left(\frac{\partial f}{\partial x_i}^{(k)}\right) \cdot \Delta_i^{(k)}$ 
11 end

```

## A.2 Steepest Descent

The Steepest Descent method is a straightforward gradient descent approach that moves in an iterative manner in the opposite direction of the gradient of the objective function at the current point. Unlike more complex methods such as the Newton method, it does not require the calculation of the Hessian of the objective function, which results in a reduction in computational expense. However, this approach exhibits a slower convergence rate [65].

Algorithm 6 outlines the steps for a single optimization cycle using the steepest descent method, where new optimization variables ( $x_{k+1}$ ) are determined by moving the current variables ( $x_k$ ) a distance  $\alpha_k$  along the direction  $d_k$ .

In the steepest descent, the choice of step size ( $\alpha_k$ ) significantly affects convergence speed and effectiveness. Line search algorithms are crucial in determining this step size, aiming for an optimal  $\alpha_k$  that minimizes the objective function along the descent direction ( $d_k$ ). One such line search algorithm is the backtracking line search, which starts with a large estimate of  $\alpha_k$  and progressively reduces it until a sufficiently small value is reached. This reduction ensures a decrease in the objective function, guided by the expected decrease based on the local function gradient  $\nabla f(x)$ .

The backtracking line search, operating under the Armijo–Goldstein condition, assesses whether moving step-wise from the current position  $x$  to a modified position  $x + \alpha_k \cdot d_k$  results in a sufficient reduction in the objective function. This condition ensures that  $f(x + \alpha_k \cdot d_k) \leq f(x) + \alpha_k \cdot c \cdot m$ , where  $c \in (0, 1)$  is a control parameter, and  $m = \nabla f(x)^T \cdot d_k$  represents the local slope in the direction  $d_k$ . While the Armijo–Goldstein condition prevents excessively large step sizes, it doesn't guarantee near-optimal ones. Hence, the backtracking line search reduces  $\alpha_k$  iteratively by a factor  $\tau \in (0, 1)$  until the condition is met.

---

### Algorithm 6: Steepest Descent

---

**Input:**  $x_k, \nabla f(x_k)$

**Output:**  $x_{k+1}$

```

1 begin
2    $d_k := -\nabla f(x_k)$ 
3    $\alpha_k \leftarrow$  determined by a line search algorithm
4    $x_{k+1} := x_k + \alpha_k \cdot d_k$ 
5 end

```

---

### A.3 Conjugate Gradient Methods

Conjugate gradient methods are a class of iterative optimization algorithms known for their efficiency in minimizing functions by searching along conjugate directions. Compared to traditional gradient descent methods, they typically demonstrate faster convergence, particularly advantageous for large-scale optimization problems [55].

The conjugate gradient methods update the search direction  $d_k$  in each iteration by combining the negative gradient direction with the previous search direction. This update ensures conjugacy between successive search directions, preventing redundant traversal of previously optimized paths and facilitating efficient convergence.

In algorithm 7, the steps for a single optimization cycle using the conjugate gradient method are presented. Here, the direction  $d_k$  is updated based on a chosen conjugacy condition,  $\beta_k$ , and the previous direction  $d_{k-1}$ . Similar to the Steepest Descent method, the step size ( $\alpha_k$ ) is determined using a line search algorithm. However, because this method requires the gradient and direction from the previous iteration, it cannot be applied in the first iteration. Therefore, the Steepest Descent algorithm is used for the initial step.

---

**Algorithm 7:** Conjugate gradient methods

---

**Input:**  $x_k, \nabla f(x_k), \nabla f(x_{k-1}), d_{k-1}$

**Output:**  $x_{k+1}$

```

1 begin
2    $\beta_k \leftarrow \beta_k^{FR} \text{ or } \beta_k^{PR}$ 
3    $d_k := -\nabla f(x_k) + \beta_k \cdot d_{k-1}$ 
4    $\alpha_k \leftarrow$  determined by a line search algorithm
5    $x_{k+1} := x_k + \alpha_k \cdot d_k$ 
6 end

```

---

Two commonly used formulas for updating  $\beta_k$  are the Fletcher-Reeves ( $\beta_k^{FR}$ ) and Polak-Ribière ( $\beta_k^{PR}$ ) formulas. Named after their respective authors, Fletcher and Reeves [55] and Polak and Ribière [56], these formulas calculate  $\beta_k$  based on the gradients at the current and previous iterations.

$$\beta_k^{FR} = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})} \quad (\text{A.3})$$

$$\beta_k^{PR} = \max \left( \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}, 0 \right) \quad (\text{A.4})$$

## A.4 Performance

All algorithms were compared to determine the most suitable one for optimizing the nonlinear programming problem of the NMPC. To achieve this, various datasets from simulation experiments were used to assess each algorithm's performance in terms of convergence speed and precision. These analyses aimed to select the best algorithm and establish terminal conditions for optimization, including the maximum number of iterations ( $itr_{max}$ ) and the minimum gradient value ( $\epsilon$ ).

Furthermore, consistency was maintained by keeping all the algorithm parameters constant throughout the tests. These parameters were defined through a preliminary tuning process. For the Rprop algorithms, specific parameters were set:  $\Delta_{min} = 1e-6$ ,  $\Delta_{max} = 1$ ,  $\eta^+ = 1.2$ , and  $\eta^- = 0.5$ . Regarding the backtracking line search,  $\tau = 0.25$  and  $c = 1e-5$ .

Figure A.1 shows the decrease of the cost function at each iteration for each gradient descent algorithm. Since the NLP is solved at each control cycle, the complexity of the optimization problem varies across cycles. For instance, at the start of the trajectory, the robot is stationary, meaning the local minimum of the cost function is far from the initial estimate. In contrast, when the robot reaches a steady-state velocity, the optimization problem's solution is close to the initial estimate (the previous cycle's solution). Additionally, the number of optimization variables changes depending on the model used, requiring more computations and time for problems with more variables. Therefore, the algorithms' precision was analyzed at the beginning, middle, and end of the trajectory for both the kinematic and dynamic models.

The results showed that all Rprop algorithms had slower convergence speeds compared to the other algorithms. This is mainly because Rprop algorithms require very precise tuning of their parameters to perform well. Additionally, the optimal parameters vary depending on the trajectory and cost function, making it challenging to maintain consistent performance without dynamically calibrating the parameters. Another drawback of Rprop algorithms, evident in datasets 3 and 4 (middle of the trajectory), is the fact that they update each optimization variable independently without ensuring a decrease in the cost function. This can lead to significant deviations from the solution, requiring more iterations to recover. In contrast, the Steepest Descent and conjugate gradient methods showed very similar results, both converging to the solution in fewer than 10 iterations in all cases. However, the conjugate gradient method of Fletcher-Reeves slightly outperformed the others. Regarding the termination criteria related to the minimum gradient value ( $\epsilon$ ), it was found that for gradients smaller than  $1e-3$ , the cost function remains practically unchanged, making this value a good termination criterion. Figure A.2 shows the number of iterations required for each algorithm to find the solution to the optimization problem for the entire trajectory, considering the previously defined termination criterion. It also presents the processing time in milliseconds required to find the solution in each of those cycles. Once again, the Rprop algorithm converged slower than the other algorithms, resulting in a higher processing time. On the other hand, the Fletcher-Reeves algorithm outperformed the other methods, particularly at the beginning and end of the trajectory. Considering the maximum processing time required and assuming a control cycle of  $40Hz$ , the termination criterion  $itr_{max} = 20$  was defined.

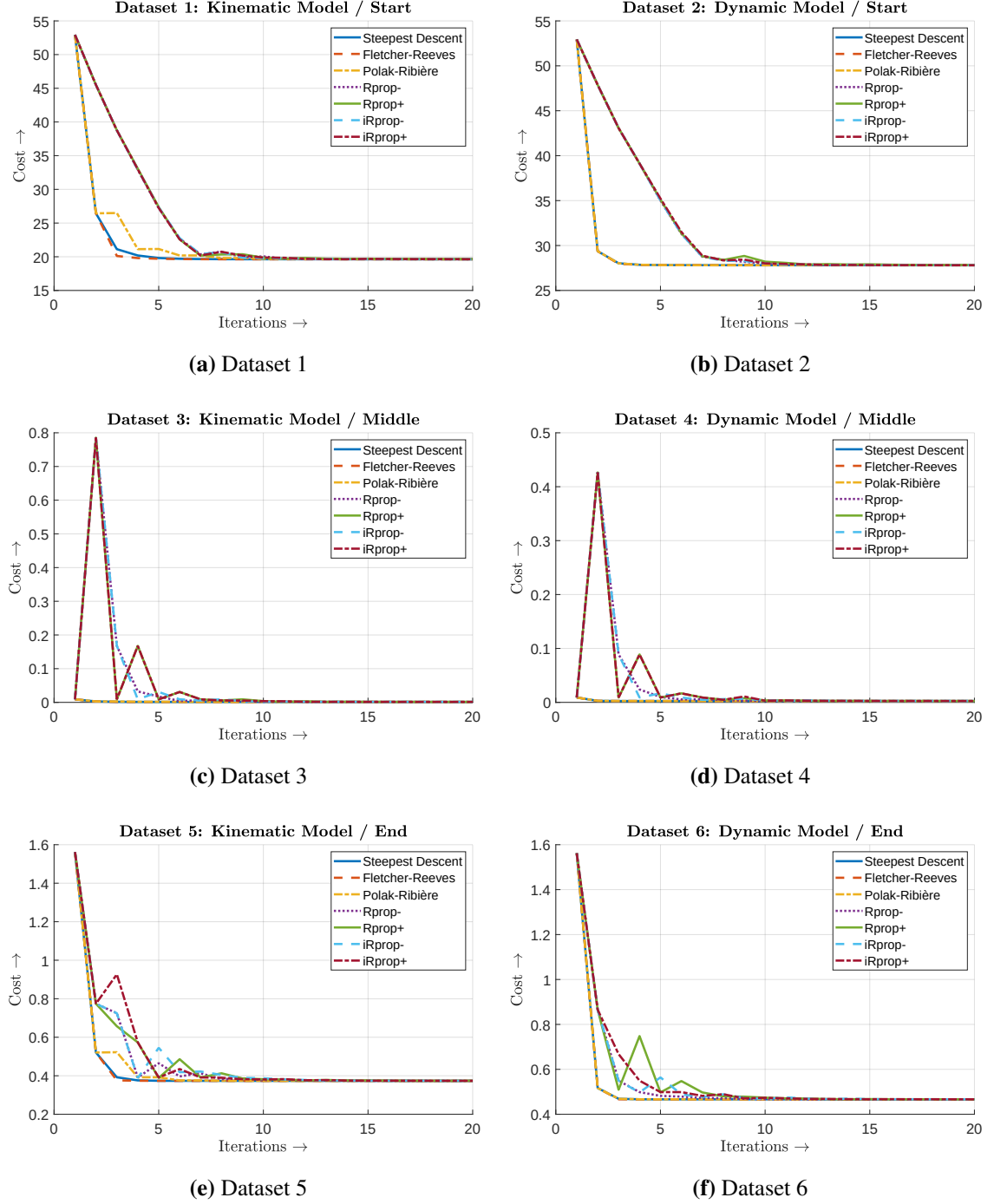
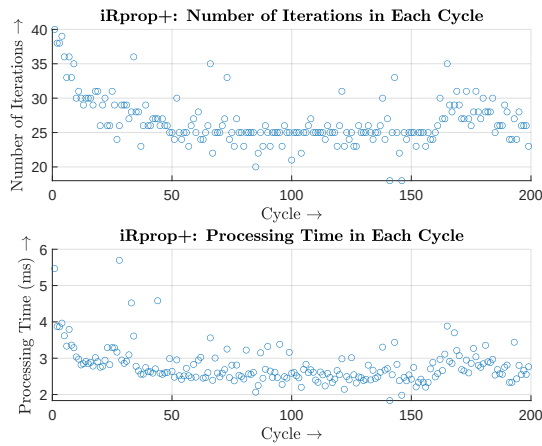
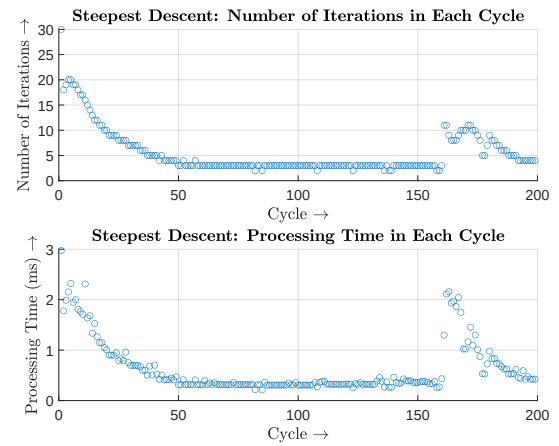


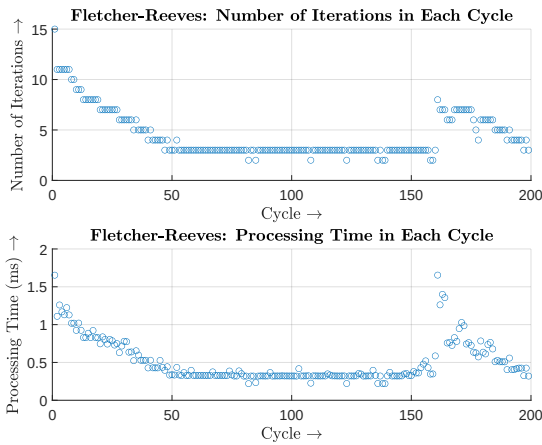
**Figure A.1:** Comparison of gradient descent methods: Datasets 1, 2, 3, 4, 5, and 6



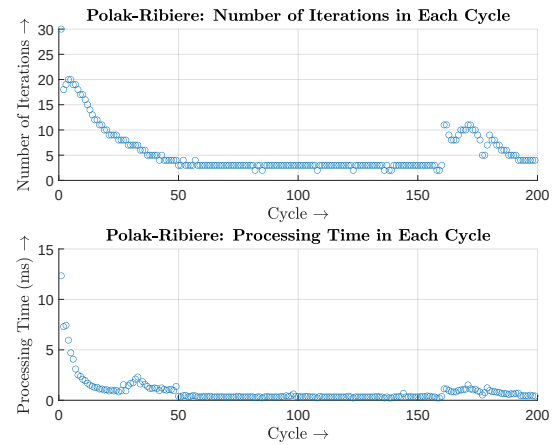
(a) iRprop+



(b) Steepest Descent



(c) Fletcher-Reeves



(d) Polak-Ribière

**Figure A.2:** Comparison of gradient descent methods in several cycles

## Appendix B

# Motion Velocity Profiles

This chapter introduces a possible implementation of the motion velocity profiles using finite state machines. As such, a Unified Modeling Language (UML) state machine diagram [66] will be presented in each of the following sections, along with the required explanation and respective equations. Figure B.1 presents a comparison between each motion profile for both the velocity and the acceleration. The primary distinction lies in the presence of instantaneous changes in acceleration in the trapezoidal profile versus a linear variation in the s-curve profile, defined by a maximum jerk. Moreover, the motion velocity profiles are derived from the one-dimensional equations of motion, as shown in equation B.1. In these equations,  $x_0$ ,  $v_0$ , and  $a_0$  denote the initial position, velocity, and acceleration, respectively, while  $j$  represents the jerk.

$$\begin{aligned} x(t) &= x_0 + v_0 \cdot t + \frac{1}{2} \cdot a_0 \cdot t^2 + \frac{1}{6} \cdot j \cdot t^3 \\ v(t) &= v_0 + a_0 \cdot t + \frac{1}{2} \cdot j \cdot t^2 \\ a(t) &= a_0 + j \cdot t \end{aligned} \tag{B.1}$$

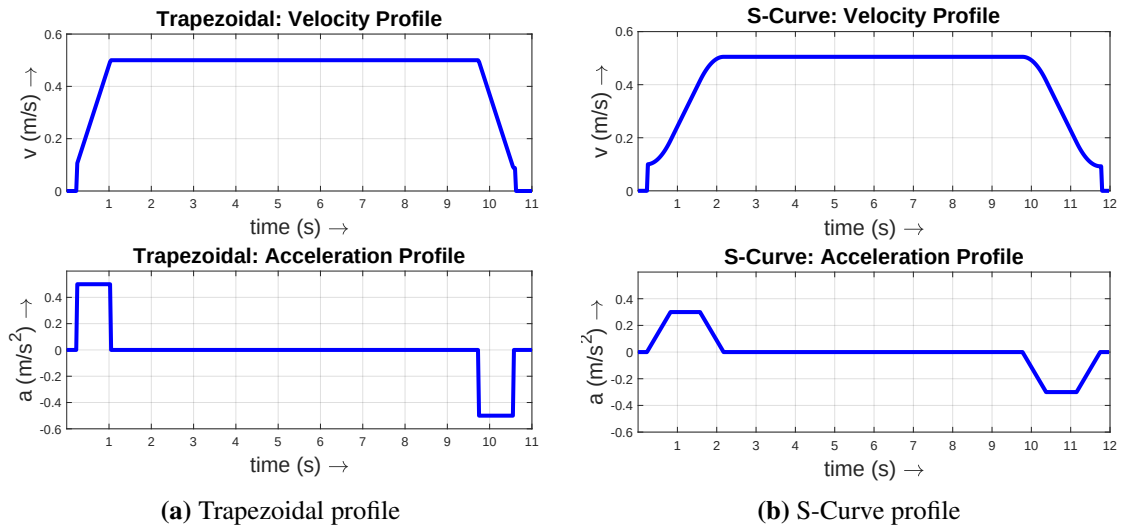


**Figure B.1:** Comparison of the motion profiles

## B.1 Trapezoidal

The UML state machine diagram for the trapezoidal motion profile is presented in figure B.2, containing six states: “Stop”, “Acceleration”, “Constant Velocity”, “Deceleration”, “Final Deceleration”, and “Final Constant Velocity”. Moreover, two compound states are used for simplification purposes, namely to facilitate the transitions between the states related to “Motion” and “Final Motion”.

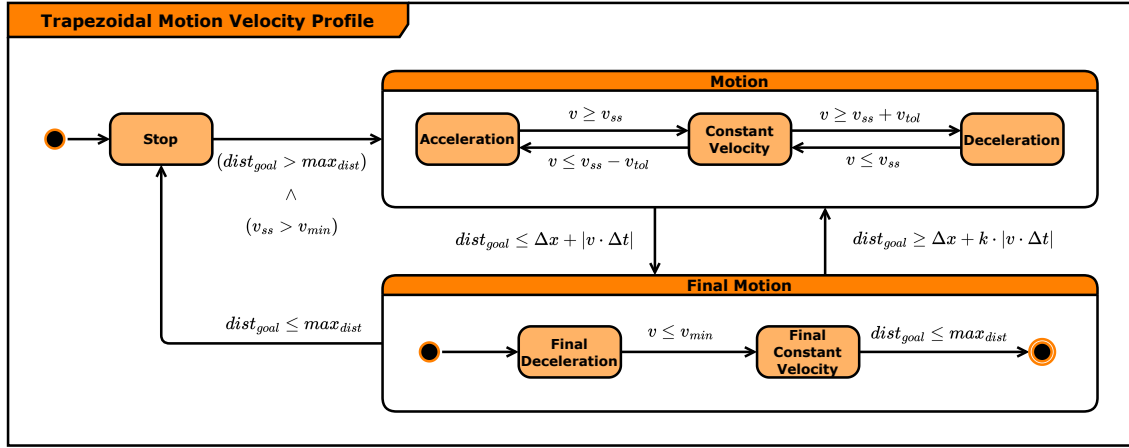


**Figure B.2:** Trapezoidal motion velocity profile UML state machine diagram

Each state follows the same motion equations (see equation B.2), where  $x_0$  and  $v_0$  denote the position and velocity when the state becomes active, and  $a$  is the acceleration specific to each state. The acceleration values are 0 for the “Stop”, “Constant Velocity”, and “Final Constant Velocity” states,  $a_{max}$  for the “Acceleration” state, and  $-a_{max}$  for the “Deceleration” and “Final Deceleration” states.

$$\begin{aligned} x(t) &= x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \\ v(t) &= v_0 + a \cdot t \end{aligned} \quad (\text{B.2})$$

Furthermore, transitions between states use as input variables the desired steady-state linear velocity ( $v_{ss} \leq v_{max}$ ) and the distance to the target position ( $dist_{goal}$ ). As such, a parameter defining the maximum distance to the target position ( $max_{dist}$ ) governs the main transitions between the compound states and the “Stop” state. Additionally, within the “Motion” compound state, transitions among “Acceleration”, “Constant Velocity”, and “Deceleration” states are managed using a tolerance value  $v_{tol}$  to allow for changes in the steady-state linear velocity ( $v_{ss}$ ).

Moreover, it is important to note that since the controller operates at a specific frequency ( $f = 1/\Delta t$ ) and is subject to disturbances, identifying the exact cycle that the final transition must start to reach the final position at 0 m/s is challenging. Furthermore, due to the robot motors’ dead zone, the robot cannot move at very low speeds. Consequently, the motion velocity profiles account for a minimum velocity outside the dead zone ( $v_{min}$ ) to facilitate transition identification and improve the accuracy at the final point. As a result, transitions between the “Motion” and

“Final Motion” compound states are determined by estimating the time ( $t_1$ ) and distance ( $\Delta x$ ) required for the final transition, considering the current velocity  $v$ , as shown in equation B.3. Additionally, the tuning parameter  $k$  must always exceed one to prevent oscillations between states.

$$\begin{aligned} v(t_1) = 0 &\Leftrightarrow t_1 = \frac{v}{a_{max}} \\ \Delta x &= v \cdot t_1 - \frac{1}{2} \cdot a_{max} \cdot t_1^2 \end{aligned} \quad (\text{B.3})$$

## B.2 S-Curve

The UML state machine diagram for the s-curve motion profile is presented in figure B.2, containing twelve states: “Stop”, “Acceleration Concave”, “Acceleration Constant”, “Acceleration Convex”, “Constant Velocity”, “Deceleration Convex”, “Deceleration Constant”, “Deceleration Concave”, “Final Deceleration Convex”, “Final Deceleration Constant”, “Final Deceleration Concave”, and “Final Constant Velocity”. To simplify the transitions, four compound states are used, similar to the trapezoidal profile: “Motion”, “Final Motion”, “Acceleration”, and “Deceleration”.

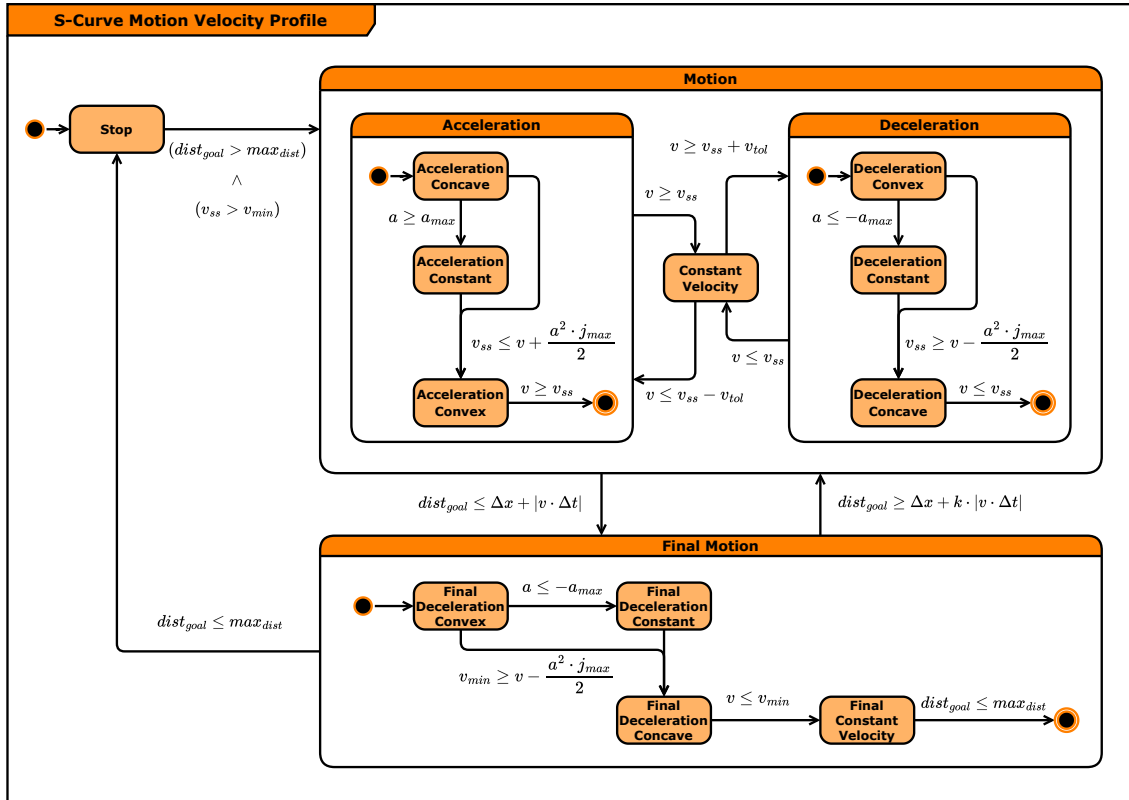


Figure B.3: S-Curve motion velocity profile UML state machine diagram

Each state follows the same motion equations (see equation B.4), where  $x_0$ ,  $v_0$ , and  $a_0$  denote the position, velocity, and acceleration when the state becomes active, and  $j$  is the jerk associated to each state. Therefore, the jerk values are 0 for the “Stop”, “Constant Velocity”, “Acceleration

Constant”, “Deceleration Constant”, “Final Deceleration Constant”, and “Final Constant Velocity” states,  $j_{max}$  for the “Acceleration Concave”, “Deceleration Convex”, and “Final Deceleration Convex” states, and  $-j_{max}$  for the “Acceleration Convex”, “Deceleration Concave” and “Final Deceleration Concave” states.

$$\begin{aligned}x(t) &= x_0 + v_0 \cdot t + \frac{1}{2} \cdot a_0 \cdot t^2 + \frac{1}{6} \cdot j \cdot t^3 \\v(t) &= v_0 + a_0 \cdot t + \frac{1}{2} \cdot j \cdot t^2 \\a(t) &= a_0 + j \cdot t\end{aligned}\tag{B.4}$$

Furthermore, the structure of the S-curve motion profile finite state machine is similar to that of the trapezoidal motion profile. However, since the acceleration cannot change instantaneously, a more complex approach is required during the acceleration and deceleration phases. During the acceleration phase, the acceleration increases linearly at a rate proportional to the maximum jerk ( $j_{max}$ ) in the “Acceleration Concave” state, until the estimated velocity variation ( $0.5 \cdot a^2 \cdot j_{max}$ ) in the “Acceleration Convex” state results in the desired final velocity. Additionally, if during the “Acceleration Concave” state, the acceleration reaches a specified maximum value ( $a_{max}$ ), it remains constant in the “Acceleration Constant” state. A similar logic applies to the deceleration phase, considering a transition from a higher to a lower velocity.

On the other hand, estimating the distance ( $\Delta x$ ) traveled during the final transition (“Final Motion” compound state) is more complex than in the trapezoidal motion profile, primarily because acceleration saturation may not necessarily occur. As a result, initially, a transition without considering saturation is performed to determine the minimum acceleration reached,  $a(t_2)$  (see equation B.5). If the minimum acceleration reached is greater than the specified minimum acceleration ( $-a_{max}$ ), indicating that saturation did not occur, the estimated distance is calculated using equation B.6. Otherwise, it is calculated using equation B.7.

$$\begin{aligned}a(t_1) &= 0 \quad \Leftrightarrow \quad t_1 = \frac{a}{j_{max}} \\v(t_1) &= v + a \cdot t_1 - \frac{1}{2} \cdot j_{max} \cdot t_1^2 = v + \frac{1}{2} \cdot \frac{a^2}{j_{max}} \\v(t_2) &= \frac{v(t_1)}{2} \quad \Leftrightarrow \quad t_2 = \sqrt{\frac{v}{j_{max}} + \frac{1}{2} \cdot \frac{a^2}{j_{max}^2}} \\a(t_2) &= -j_{max} \cdot t_2\end{aligned}\tag{B.5}$$

if  $a(t_2) \geq -a_{max}$  :

$$\left\{ \begin{array}{lcl} t_1 & = & \frac{a}{j_{max}} + \sqrt{\frac{v}{j_{max}} + \frac{1}{2} \cdot \frac{a^2}{j_{max}^2}} \\ x(t_1) & = & x + v \cdot t_1 + \frac{1}{2} \cdot a \cdot t_1^2 - \frac{1}{6} \cdot j_{max} \cdot t_1^3 \\ v(t_1) & = & v + a \cdot t_1 - \frac{1}{2} \cdot j_{max} \cdot t_1^2 \\ a(t_1) & = & a - j_{max} \cdot t_1 \\ t_2 & = & \sqrt{\frac{v}{j_{max}} + \frac{1}{2} \cdot \frac{a^2}{j_{max}^2}} \\ x(t_2) & = & x(t_1) + v(t_1) \cdot t_2 + \frac{1}{2} \cdot a(t_1) \cdot t_2^2 + \frac{1}{6} \cdot j_{max} \cdot t_2^3 \\ \Delta x & = & x(t_2) - x \end{array} \right. \quad (B.6)$$

if  $a(t_2) < -a_{max}$  :

$$\left\{ \begin{array}{lcl} t_1 & = & \frac{a+a_{max}}{j_{max}} \\ x(t_1) & = & x + v \cdot t_1 + \frac{1}{2} \cdot a \cdot t_1^2 - \frac{1}{6} \cdot j_{max} \cdot t_1^3 \\ v(t_1) & = & v + a \cdot t_1 - \frac{1}{2} \cdot j_{max} \cdot t_1^2 \\ t_3 & = & \frac{a_{max}}{j_{max}} \\ v(t_2) & = & a_{max} \cdot t_3 - \frac{1}{2} \cdot j_{max} \cdot t_3^2 \\ t_2 & = & \frac{v(t_1) - v(t_2)}{a_{max}} \\ x(t_2) & = & x(t_1) + v(t_1) \cdot t_2 - \frac{1}{2} \cdot a_{max} \cdot t_2^2 \\ x(t_3) & = & x(t_2) + v(t_2) \cdot t_3 - \frac{1}{2} \cdot a_{max} \cdot t_3^2 + \frac{1}{6} \cdot j_{max} \cdot t_3^3 \\ \Delta x & = & x(t_3) - x \end{array} \right. \quad (B.7)$$

## Appendix C

# Splines

This chapter introduces splines as used throughout this work and presents several numeric algorithms required to determine the closest point on a spline, the next positions on the spline, and the length of a spline.

### C.1 Formulation

In this thesis, splines are defined as piecewise polynomial parametric curves denoted by  $Q(t)$ . While splines can exist in various spaces, they're typically confined to a two-dimensional plane in the field of ground mobile robotics:  $Q(t) \in \mathbb{R}^2$ . However, most principles discussed here apply universally. Traditionally, splines consist of segments, each being a polynomial of the same degree  $n$ . However, this thesis will focus on cubic splines, which offer a good balance between simplicity and localism.

Furthermore, the existing literature presents three common methods for defining the polynomial segments of a spline [28]. The following section will examine each of the three approaches, with a focus on the cubic segment within the  $\mathbb{R}^2$  domain.

**Polynomial Parameterization:** This method employs standard polynomial notation, expressing the segment  $S(t)$  as individual cubic polynomials for each dimension.

$$S(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} a_x t^3 + b_x t^2 + c_x t + d_x \\ a_y t^3 + b_y t^2 + c_y t + d_y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \mathbf{T} \cdot \mathbf{C} \quad (\text{C.1})$$

Equation C.1 introduces the coefficient matrix  $\mathbf{C}$ , characterizing a specific cubic curve, and introduces  $\mathbf{T}$  as the vector of parameter powers.

**Weighted Sum of Control Vertices:** Alternatively,  $S(t)$  can be represented as a weighted sum of control vertices  $p_i \in \mathbb{R}^2$ ; for cubic polynomials, there are four of them.

$$S(t) = \sum_{i=0}^3 p_i B_i(t) \quad (\text{C.2})$$

This notation interprets the cubic polynomial as a linear combination of control vertices  $p_i \in \mathbb{R}^2$ , weighted by basis splines  $B_i(t)$ , which are cubic polynomials. A particular segment is here characterized by its control vertices, whereas the basis splines can be interpreted as defining the family of segments to which it belongs.

**Product of Matrices:** Combining the previous methods, a segment can also be expressed through the product of matrices and vectors.

$$S(t) = \mathbf{T} \cdot \mathbf{M} \cdot \mathbf{G} \quad (\text{C.3})$$

Here,  $T$  is the vector containing the parameter's potencies as introduced in equation C.1. Additionally, the coefficient matrix  $\mathbf{C}$  has been split into a basis matrix  $\mathbf{M}$  and a geometry matrix  $\mathbf{G}$  by  $\mathbf{C} = \mathbf{M} \cdot \mathbf{G}$ .

$$\mathbf{M} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix} \quad (\text{C.4})$$

The geometry matrix in equation C.4 characterizes the segment by its four control vertices  $G_i \mid i \in \{0 \dots 3\}$ , while the basis matrix defines the curve's family. Furthermore, comparing with equation C.2, the product of the parameter vector  $\mathbf{T} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$  with the basis matrix  $\mathbf{M}$  is equivalent to a vector containing the basis splines from equation C.2.

$$\mathbf{T} \cdot \mathbf{M} = \begin{bmatrix} m_{00} t^3 & m_{01} t^2 & m_{02} t & m_{03} \\ m_{10} t^3 & m_{11} t^2 & m_{12} t & m_{13} \\ m_{20} t^3 & m_{21} t^2 & m_{22} t & m_{23} \\ m_{30} t^3 & m_{31} t^2 & m_{32} t & m_{33} \end{bmatrix}^T \quad (\text{C.5})$$

Furthermore, it's worth noting that within each spline segment  $S(t)$ , the parameter  $t$  ranges from 0 to 1. This thesis focuses exclusively on two spline families: Bézier and Hermite splines. Consequently, equation C.6 presents the  $\mathbf{M}$  matrix for a cubic Bézier spline, while equation C.7 presents it for a cubic Hermite spline.

$$\mathbf{M} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (\text{C.6})$$

$$\mathbf{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (\text{C.7})$$

These two spline families share significant similarities, differing primarily in their control point structure. The cubic Bézier spline employs four control points defined by pairs of  $(x, y)$  coordinates. Among these, the initial and final control points establish the start and end positions of the spline, while the intermediate control points define its shape. In contrast, the cubic Hermite spline utilizes two control points and two control vectors. Like the Bézier spline, these two points determine the spline's start and end positions. However, the additional control vectors define the velocity at these points, directly influencing the spline's orientation.

Moreover, a transformation between these two representations can be achieved using equation C.8, where  $\mathbf{B}$  denotes the geometric matrix of a cubic Bézier spline, encompassing all four control points. On the other hand,  $\mathbf{H}$  represents the geometric matrix of a cubic Hermite spline, containing its two control points,  $(h_{0,x}, h_{0,y})$  and  $(h_{3,x}, h_{3,y})$ , along with the two control vectors,  $(h_{1,x}, h_{1,y})$  and  $(h_{2,x}, h_{2,y})$ .

$$\mathbf{B} = \frac{1}{3} \begin{bmatrix} 3 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 0 & 3 & 0 & -1 \\ 0 & 3 & 0 & 0 \end{bmatrix} \cdot \mathbf{H}, \quad \mathbf{B} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad (\text{C.8})$$

Finally, both the Bézier and Hermite spline families fall under the category of manual stitching splines, meaning they are constructed from independent spline segments. Therefore, equation C.9 presents how the mapping between the global spline  $Q(t)$  and each segment  $S_i(t)$  can be executed, taking into account the parameter  $t$ .

$$Q(t) = S_i(t^*), \quad \text{where } i = \lfloor t \rfloor \text{ and } t^* = t - \lfloor t \rfloor \quad (\text{C.9})$$

## C.2 Numeric Algorithms

While splines offer advantageous properties, such as their ability to represent diverse smooth paths using a minimal parameter set, they also have limitations that must be considered. One of the most significant limitations is the inability to analytically determine certain information, such as the closest point on a spline or its length. Consequently, this section introduces numerical algorithms designed to extract essential information, crucial for both trajectory tracking and collision avoidance applications in mobile robotics.

First, regarding trajectory tracking, two key algorithms are essential: one for determining the closest point on the spline and another for discretizing the spline to identify subsequent positions forming the trajectory. In the literature several numerical algorithms for finding the closest point on a spline were already proposed, often using the Newton method to minimize the distance between the point and the spline [31, 32]. Alternatively, algorithm 8 employs a semi-brute-force approach to effectively locate the closest point on the spline.

In each iteration, the algorithm divides the spline into  $m$  equally spaced points and identifies the nearest one. The search window is then narrowed, and a new search is conducted around that point in the next iteration. Since the closest point must be determined at each control cycle in trajectory tracking applications, and the new solution is near the previous one, the algorithm can be initialized with the previously calculated solution. This reduces the search space in the first iteration, thereby finding the closest point in fewer iterations.

The algorithm was tested on various paths and initialization conditions, using  $m = 10$  and  $\Delta t_{min} = 1e-3$ . It consistently found the solution with high precision in less than 1 ms.

On the other hand, algorithm 9 was proposed to discretize the spline and determine the next positions taking into account a motion velocity profile for the robot. This algorithm is required as normally the velocity of the spline  $Q'(t)$  is not directly used as the trajectory velocity. By decoupling the velocity from the spline formulation, the trajectory velocity can be specified independently, allowing the robot to travel spline segments with different lengths at a constant speed.

Therefore, the algorithm involves a direct transformation between the desired trajectory velocity  $\mathbf{v}[i]$  and the spline velocity  $Q'(t)$ . Using this transformation, a step size ( $\Delta t$ ) is defined to iteratively move along the spline until all the required points are found. These points are retrieved over a specified time interval  $\tau$ , which typically matches the controller's time step. If the end of the spline is reached before all points are determined, the final point is repeated to fill the required positions.

Additionally, for an Nonlinear Model Predictive Control (NMPC) system of a mobile robot, where the reference trajectory includes both position and orientation, the orientation at each point can be derived from the spline's direction  $\theta = \text{atan2}(v_y, v_x)$ . Alternatively, for omnidirectional robots, an arbitrary reference orientation can be used.

Finally, to determine the spline length, a similar algorithm is employed (algorithm 10), but with a fixed resolution  $\delta$  to move along the spline.

---

**Algorithm 8:** Closest Point on a Spline

---

**Input:**  $Q(t)$  (spline),  $\mathbf{q}$  (point),  $t_{ini}$  (initial guess),  $t_{max}$  (number of piecewise polynomials)**Data:**  $m$  (number of samples per iteration),  $\Delta t_{min}$  (minimum resolution)**Output:**  $\mathbf{q}_{closest}$  (closest point),  $t_{closest}$  (closest parameter),  $d_{closest}$  (distance)

```

1 begin
2    $t_{ini} \leftarrow \lfloor t_{ini} \rfloor$ 
3   if  $(t_{ini} < 0)$  or  $(t_{ini} > n)$  then
4      $\Delta t \leftarrow t_{max}/m$ 
5      $t_{closest} \leftarrow 0$ 
6   else
7      $t_1 \leftarrow \max(t_{ini} - 1, 0)$ ,  $t_2 \leftarrow \min(t_{ini} + 1, t_{max})$ 
8      $\Delta t \leftarrow (t_2 - t_1)/m$ 
9      $t_{closest} \leftarrow t_1$ 
10  end
11   $\mathbf{q}_{closest} \leftarrow Q(t_{closest})$ 
12   $d_{closest} \leftarrow \|\mathbf{q}_{closest} - \mathbf{q}\|$ 
13  repeat
14     $t_1 \leftarrow \max(t_{closest} - \Delta t, 0)$ ,  $t_2 \leftarrow \min(t_{closest} + \Delta t, t_{max})$ 
15     $\Delta t \leftarrow (t_2 - t_1)/m$ 
16     $t \leftarrow t_1$ 
17    for  $i \leftarrow 1$  to  $n$  do
18       $t \leftarrow t + \Delta t$ 
19       $\mathbf{q}_{new} \leftarrow Q(t)$ 
20       $d_{new} \leftarrow \|\mathbf{q}_{new} - \mathbf{q}\|$ 
21      if  $d_{new} < d_{closest}$  or  $i = 1$  then
22         $t_{closest} \leftarrow t$ ,  $\mathbf{q}_{closest} \leftarrow \mathbf{q}_{new}$ ,  $d_{closest} \leftarrow d_{new}$ 
23      end
24    end
25  until  $\Delta t < \Delta t_{min}$ 
26 end

```

---

---

**Algorithm 9:** Next Points on a Spline

---

**Input:**  $Q(t)$  (spline),  $t_{ini}$  (starting point parameter),  $\mathbf{v}$  (velocities),  $\tau$  (time interval),  
 $n$  (number of poses),  $t_{max}$  (number of piecewise polynomials)

**Data:**  $m$  (number of samples per iteration)

**Output:**  $\mathbf{p}$  (next points parameters)

```

1 begin
2    $t \leftarrow t_{ini}$ 
3    $\mathbf{q}_{old} \leftarrow Q(t), \mathbf{v}_{old} \leftarrow Q'(t)$ 
4    $\delta \leftarrow (\mathbf{v}[0] \cdot \tau) / m$ 
5    $\Delta t \leftarrow \delta / \|\mathbf{v}_{old}\|$ 
6    $d \leftarrow 0, i \leftarrow 1$ 
7   repeat
8      $t \leftarrow t + \Delta t$ 
9      $t \leftarrow \min(t, t_{max})$ 
10     $\mathbf{q}_{new} \leftarrow Q(t), \mathbf{v}_{new} \leftarrow Q'(t)$ 
11     $d \leftarrow d + \|\mathbf{q}_{new} - \mathbf{q}_{old}\|$ 
12    if  $d > \mathbf{v}[i] \cdot \tau$  then
13       $\mathbf{p}[i] \leftarrow t$ 
14       $d \leftarrow 0, i \leftarrow i + 1$ 
15    end
16     $\Delta t \leftarrow \delta / \|\mathbf{v}_{new}\|$ 
17     $\mathbf{q}_{old} \leftarrow \mathbf{q}_{new}$ 
18  until  $t > t_{max}$  or  $i > n$ 
19  if  $i < n$  then
20    for  $j \leftarrow i$  to  $n$  do
21       $\mathbf{p}[j] \leftarrow t_{max}$ 
22    end
23  end
24 end

```

---

---

**Algorithm 10:** Length of a Spline

---

**Input:**  $Q(t)$  (spline),  $t_{ini}$  (starting point parameter),  $t_{max}$  (number of piecewise polynomials)**Data:**  $m$  (number of samples per iteration)**Output:**  $l$  (length)

```

1 begin
2    $t \leftarrow t_{ini}$ 
3    $\mathbf{q}_{old} \leftarrow Q(t), \mathbf{v}_{old} \leftarrow Q'(t)$ 
4    $\delta \leftarrow 1/m$ 
5    $\Delta t \leftarrow \delta / \|\mathbf{v}_{old}\|$ 
6    $l \leftarrow 0$ 
7   repeat
8      $t \leftarrow t + \Delta t$ 
9      $t \leftarrow \min(t, t_{max})$ 
10     $\mathbf{q}_{new} \leftarrow Q(t)$ 
11     $l \leftarrow l + \|\mathbf{q}_{new} - \mathbf{q}_{old}\|$ 
12     $\mathbf{v}_{new} \leftarrow Q'(t)$ 
13     $\Delta t \leftarrow \delta / \|\mathbf{v}_{new}\|$ 
14     $\mathbf{q}_{old} \leftarrow \mathbf{q}_{new}$ 
15  until  $t > t_{max}$ 
16 end

```

---