Non-linear Model Predictive Control Based Motion Planning Among People

Master thesis | Control and Automation

Balázs Reiser, Martin Bieber Jensen and Simon Nanoq Callisen

June, 2020

AALBORG UNIVERSITY
DENMARK

Title:
Non-linear Model Predictive Control Based Motion Planning Among People

Project Type:
Master Thesis

Project Period:
Spring 2020
Date: 01/02-2020 - 16/06-2020

Group number:
CA10-1032

Author:
Balázs Reiser
Martin Bieber Jensen
Simon Nanoq Callisen

AAU Supervisor:
Karl Damkjær Hansen
kdh@es.aau.dk

Total Pages: 108

Control and Automation 4^{th} Semester
The Technical Scientific Faculty
School of Information and Communication Technologies
Fredrik Bajers Vej 7B
9220 Aalborg East
webinfo@es.aau.dk

Abstract:

The scope of this thesis is to develop a non-linear model predictive control (NMPC) based path planning for a non-holonomic mobile robot. The purpose of the planner is to enable the PAL moving base robotic platform to navigate in a dynamic environment considering static and moving obstacles. A non-linear kinematic model is designed for the robot as well as a linear state-space model to represent moving obstacles. These models are implemented in a NMPC to predict robot and obstacle movement. Driving the design of this NPMC is a parallel simulation development of different planning methods, Resulting in a set-point stabilisation NMPC local planner with multiple shooting discretization. In addition, two different polygon shaped obstacle representation method is developed and implemented as constraints to the NMPC design. Finally the planner is implemented on the PAL moving base and compared to a contemporary planner to evaluate performance, through these test results concluding comparable performance is achievable using the NMPC based planner with added moving obstacle functionality.

PREFACE

This thesis is made as a completion of the master education in Control and Automation at Aalborg University, ending spring 2020. The aim of this thesis is to develop a local path planning algorithm for motion among people in addition to navigating a static environment. The local planner has to follow a series of local goal given by the global planner until the robot reaches it's target destination or global goal. This local planner is developed alongside the working body of the Human Aware Mobile Robotics lab (HAMR-lab) at Aalborg University(AAU). The planner has been mainly developed on the PAL Robotics TIAGo robot. The TIAGo robot is intended to be used as a development platform for the KUGLE Project where the dynamics of a ball-balancing robot, such as the KUGLE robot, does not have to be taken into account. This is done such that the focus can be directed towards planning rather than modelling. The local planner is based upon the concepts of Non-linear Model Predictive Control, set-point stabilization and constraint identification. The thesis also relies on the concepts of control theory, path planning, kinematic modelling, linear algebra and optimization. To understand the Thesis knowledge within these concepts is required.

The report's structure is built up from seven chapters, each covering a significant part of the entire project.
Chapter 1 - Introduces the Human Aware Mobile Robotics lab, the TIAGo robot and what the scope of the project is. The chapter concludes with some core questions that will be further investigated.
Chapter 2 - Presents the findings of the questions asked and further narrows down the core questions that seeks to be answered in this thesis. This Chapter concludes with a problem formulation.
Chapter 3 - Presents the modelling of the TIAGo robot as well as the pedestrians/-moving obstacles. It also defines how the static and moving obstacles are represented as constraints.
Chapter 4 - Details the formulation of a Non-linear Model Predictive Control (NMPC) problem developed for local planning of the TIAGo robot. The general ideas of stabilization and tracking NMPC are detailed. The Chapter gives a brief description of the optimal control problem, NMPC discretization and the consideration of constraints for the NMPC local planner.
Chapter 5 - Presents simulations of the local planner in MATLAB and Gazebo. The simulations have been used to develop the NMPC.

Chapter 6 - Describes the process of implementing the local planner into the TIAGo platform and the formulation of the NMPC using the software tool CasADi for numerical optimization.
Chapter 7 - Compares the obstacle representations in different scenarios. The Chapter also compares the local planner developed in this thesis to the on-board PAL-local planner pre-installed on the robot.
Chapter 8 - Discusses the results of the tests, the limitations of the tests and what they mean for the local planner.
Chapter 9 - Summarizes and concludes on the overall results and findings of the Thesis related to the problem formulation as well as the future work that could improve the local planner.

Reading directions:

- The Nomenclature contains a list of abbreviations of terms and phrases used throughout this report.
- The Figures that are not made by the authors of this report are referenced below them.
- The Bibliography contains the sources and is written in order of appearance throughout the report. The bibliography follows the IEEE referencing scheme.
- The Appendix includes extra materials that are used in the report, such as algorithms, test results, simulation results and large figures.
- There is a list of figures and a list of tables that can be used to look up figures in the report.
- The learning objectives can also be found in the Appendix, which should be fulfilled through the formation and writing of this Thesis.

Special thanks to ***Karl Damkjær Hansen*** *for the continuous supervising and suggestions of possible solutions and methods throughout the project.*

NOMENCLATURE

Common Abbreviations

This section contains explanations and abbreviations of specific words which are used throughout the report.

- **AAU** - Aalborg Unviversity
- **AMCL** - Adaptive Monte Carlo Localization
- **CVM** - Curvature-Velocity Method
- **DBSCAN** - Density-Based Spatial Clustering of Applications with Noise
- **DoF** - Degrees of Freedom
- **DWA** - Dynamic Window Approach
- **HAMR** - Human Aware Mobile Robotics
- **HRI** - Human Robot Interaction
- **IMU** - Inertial Measurement Unit
- **LiDAR** - Light Detection And Ranging.
- **MPC** - Model Predictive Control
- **NLP** - Non-linear Programing
- **NMPC** - Non-linear Model Predictive Control
- **ODE** - Ordinary Differential Equation
- **OCP** - Optimal Control Problem
- **PFM** - Potential Field Method
- **PMB** - PAL Mobile Base
- **RDP** - Ramer–Douglas–Peucker algorithm
- **ROS** - Robot Operation System
- **RRT** - Rapidly-Exploring Random Tree
- **TEB** - Timed Elastic Band
- **VFH** - Virtual Field Histogram
- **VO** - Velocity Obstacle

CONTENTS

CHAPTER 1

INTRODUCTION

Mobile robotics is increasingly deployed in public areas, dense with humans and other moving obstacles. Navigating, through such an environment effectively without collision, is one of the factors driving the need for more advanced motion planning algorithms. Under the topic of manoeuvring in a space with moving obstacles, a vital part is taking into consideration the movements of the people. Planning motion of a robot while avoiding moving people is not a simple task, as the future position of a given person has to be taken into consideration, as people usually do not move in a strictly linear patterns. Therefore prediction of the people's movement is needed, based on a representative model of human movement, to figure out which path to take to not collide with people in the environment. To provide the necessary comfort for someone who is moving in a given environment, it is crucial to provide a sense of certainty about how one's environment is going to change. This is a feeling which is often hard to instil using robots in a Human-Robot Interaction(HRI) scenario. Because of these factors, another important topic in shared environments to consider is how to ensure comfortable HRIs in a dynamic environment.

This thesis is developed alongside the working body of the Human Aware Mobile Robotics lab (HAMR-lab) at Aalborg University(AAU). The focus of HAMR-lab lies in creating mobile robotics solutions, implementable in social environments around people. Social environments being areas in which people move around freely. An emerging focus in the field of mobile robotics is that of guiding robots for assistance in navigation. For this purpose, the KUGLE robot [1] project is developed specifically for the scenario of aiding in the navigation of a hospital environment. An imagined scenario could be the robot being available at the hospital entrance with a list of locations, to which it can guide a visitor around by leading them to their destination. The KUGLE robot is a ball balancing robot capable of navigating static environments. In this, the HAMR research group works on product development along with software development and modelling research pertaining to both KUGLE and human movement modelling.
The focus of this Thesis lies in creating a navigation package for the mobile robots used by the HAMR-lab group. Specifically, the robotic platform TIAGo from PAL Robotics is used to simulate the KUGLE robot. The TIAGo platform is an extensive robotics platform with many capabilities within the field of navigation.
The navigation package aims to increase the capabilities of the TIAGo platform. It is developed to be a local planner, which receives goal points from the already extensive global

planner. There are already some local planning methods implemented on the TIAGo platform, such as the Dynamic Window Approach(DWA) and Timed Elastic Bands(TEB). However, the envisioned environment includes moving obstacles (humans), which are not considered in these methods, requiring the expansion of local planning methods to include capabilities which consider moving obstacles. The algorithm used for planning the trajectory of the robot has to ensure that the robot can navigate to the goal and consider the motion of the robot throughout the trajectory, to make sure it moves naturally and smoothly.

The TIAGo robot is used to avoid the challenges that arise with using a ball-balancing robot, simplifying the model needed for control of the system. The TIAGo platform runs Ubuntu with Robot Operating System(ROS) for inter-system communication, the idea being for the navigation package to be a plug and play solution for navigation able to run on any platform with ROS, given a minor initial setup. For solving the issue of local planning in a dynamic environment, specific problems arise, some of the relevant problems have been formulated as follows:

- How to represent pedestrian motion, modelling humans as moving obstacles?
- How can social situations be recognized and represented to ensure comfortable human-robot interaction?
- How can local planning be implemented to ensure collision-less movement through an environment with moving obstacles?
- How to ensure smooth robot motion throughout operation given an environment, including moving obstacles?

A literature review is conducted in the following Chapter 2, specifically investigating methods that propose solutions to above questions. Based on this state of the art research, an approach is selected for further investigation throughout this Thesis.

CHAPTER 2

STATE OF THE ART

This chapter introduces the background of robot motion planning and investigates some state of the art procedures previously used to solve problems similar to the ones treated in this thesis. The problems, as stated in Section 1, can be split into two main issues. The first being that of ensuring proper HRI, ensuring socially aware and responsible navigation from the robot, described in Section 2.1. The second issue is that of facilitating robot navigation through a dynamic environment. Ensuring no collisions in an environment, including dynamic obstacles, described in Section 2.2. The chapter concludes with an evaluation of the currently employed methods that solve the problem of navigation in an environment with moving obstacles.

2.1 Human-Robot Interaction

An important part of most robot navigation scenarios is the ability to avoid obstacles. For this thesis specifically, both static and moving obstacles are considered. The robot should be able to navigate in an environment where humans should be able to interact with the robot and while also avoiding humans that come in the way of the robot. The root of this problem is ensuring natural Human-Robot Interaction (HRI). To interact appropriately with humans, Dautenhahn [2] proposes that robots need to be equipped with several non-trivial behaviours instilling a feeling of intelligence in the robot from the human perspective. Such as ensuring that the robot responds reliably in real-time to highly dynamic human behaviour.
To ensure natural HRI, criteria for defining natural interaction, need to be established. This is done to ensure both physically safe and socially comfortable robot operation [3]. Specifically useful for the scenario envisioned in this project is the measure of proxemic interpersonal distance as found by E.T.Hall [4], seen in Table 2.1. Describing different distance thresholds in which people generally are comfortable, given different types of interactions with actors in their environment. These distances are useful for guiding how the robot should circumnavigate people in the environment while considering personal space of the people.

Designation	Specification	Reserved for...
Intimate Distance	0-45cm	Embracing, touching, whispering
Personal Distance	45-120cm	Friends
Social Distance	1.2-4.6m	Acquaintances and strangers
Public Distance	>3.6m	Public speaking

Table 2.1: Proxemic interpersonal distances found by E.T.Hall[4]. The use of these distances regarding human-robot interaction is an open research question.

The proxemic interpersonal distances in Table 2.1 is an excellent guide to help figure out the distance a robot should try to keep to people, specifically given the nature of the interaction they might have. This way, basic distance measure would only pose as an obstacle enlargement parameter; it is also important to consider the direction the person is facing [5].
Kollmitz et al. propose a way to represent the people in the environment as dynamic obstacles[6]. This is done by representing individual human obstacles as a set of Gaussian distributions with varying standard deviation along the side and facing the direction of the person, as well by the velocity in that direction. The obstacle is then being represented by a field of probabilities, with a strictly non-traversable area in the centre representing the predicted position given the velocity of the person. How the field of probabilities evolve can be seen in Figure 2.1

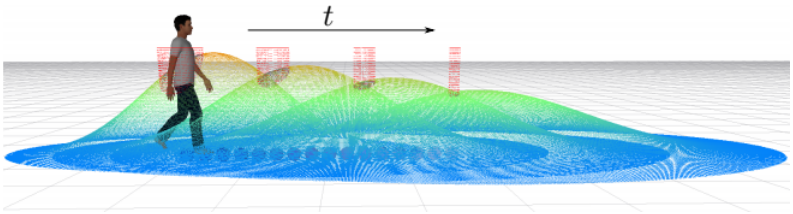


Figure 2.1: The dynamic cost mapping created to represent human motion by Kollmitz et al., the cost decreasing and spreading over time as confidence in path decreases. [6]

The field of probabilities is convertible to a potential field representation to enforce socially preferable trajectories during motion planning. To represent the prediction of the position of the obstacle a layered dynamic cost-map is proposed, with the social cost field being propagated in time, the standard deviations of the distribution increasing for further prediction steps.
Another thing to consider regarding these is the interpersonal relationships between actors in the environment. A way to include these relations and circumnavigate them as not to disturb social interactions is set forth by Truong & Ngo [5]. This method extends upon the findings of Kollmitz by introducing social interaction spaces. Social interaction spaces being a combination of dynamic obstacles given close enough proximity, if two people are

gathered close enough, then a new object is created to signify the social interaction as an obstacle. This can be seen illustrated in Figure 2.2.

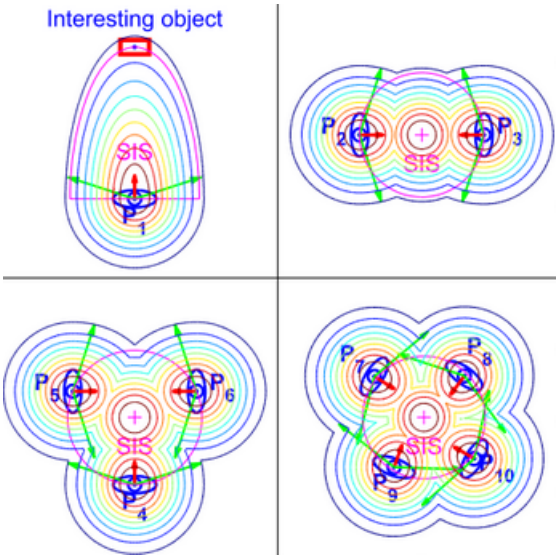


Figure 2.2: Representation of different social zone situations. A person interacting with an object (top left), a two person social zone (top right), 3 person social zone(bottom left) and a 4 person social zone [5]

The method by Truong and Ngo implements these social interaction spaces as their own obstacles with a potential field to deter close movement by the robot, based on people gathered closely together. Based on these differently sized social zone representations a preferred angle of interruption can also be investigated. For example the robot should approach the depicted 3 person social zone from the top, as all three actors would be able to see the robot approaching, providing a more comfortable interaction.

2.2 History of Motion Planning

Motion planning for mobile robots gained the attention of several researchers around the world since 1979 when Loyano-Pérez and Wesley presented the concept of the configuration space [7]. All service robots feature some kind of motion planning in a collision-free manner, ranging from simple algorithms that detect an obstacle and terminate the robot's movement in order to avoid collision, through advanced algorithms, which enable the robot to circumnavigate obstacles in an optimal way.
Early collision avoidance strategies were based on the use of artificial potential field methods (PFM). O. Khatib in [8] has suggested the idea of imaginary forces acting on a robot. In this global planning strategy, obstacles have repulsive forces, while the target applies an attractive force to the robot. By harnessing these repulsive and attractive forces, the

mobile robot can avoid colliding with nearby obstacles and reach its target safely. The way that these forces are implemented is based on the definition of their respective potential fields. The attractive potential field being defined by:

$$U_{att}(q) = \frac{1}{2}\xi\rho^m(\mathbf{q}, \mathbf{q}_{goal}) \tag{2.1}$$

Where ξ is a positive scaling factor, $\rho(q, q_{goal})$ is the distance between the robot and goal and m is a shaping factor for the field, if 1 it is conic and if 2 it has a paraboloid shape. The corresponding attractive force acting upon the robot is then the negative gradient of the attractive potential field U_{att}:

$$F_{att}(q) = -\nabla U_{att}(q) = \xi(\mathbf{q}_{goal} - \mathbf{q} \tag{2.2}$$

A commonly used simple definition of repulsive potential function being defined as following:

$$U_{rep}(q) = \begin{cases} \frac{1}{2} * \eta \left(\frac{1}{\rho(\mathbf{q}, \mathbf{q}_{obs}}) - \frac{1}{\rho_0} \right), & \text{if } \rho(\mathbf{q}, \mathbf{q}_{obs}) \leq \rho_0 \\ 0, & \text{if } \rho(\mathbf{q}, \mathbf{q}_{obs}) > \rho_0 \end{cases} \tag{2.3}$$

Where η is a positive scaling factor, $\rho(q, q_{obs})$ describes the distance between the robot and a given obstacle and ρ_0 a constant describing the influence distance of the obstacle, beyond which the repulsive potential is zero. The corresponding repulsive force is given by:

$$F_{rep}(q) = \begin{cases} \eta \left(\frac{1}{\rho(\mathbf{q}, \mathbf{q}_{obs}} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(\mathbf{q}, \mathbf{q}_{obs})} \nabla\rho(\mathbf{q}, \mathbf{q}_{obs}), & \text{if } \rho(\mathbf{q}, \mathbf{q}_{obs}) \leq \rho_0 \\ 0, & \text{if } \rho(\mathbf{q}, \mathbf{q}_{obs}) > \rho_0 \end{cases} \tag{2.4}$$

Adding the gradient of the distance to the obstacle $\rho(q, q_{obs})$ and a scaling factor for proximity to the obstacle $\frac{1}{\rho^2(q,q_{obs})}$.

B. H. Krogh [9] has improved this concept by taking into consideration the robot's velocity in the vicinity of obstacles. C. F. Thorpe [10] has applied the potential field method to off-line path planning. Later Krogh and Thorpe [11] suggested a combined method for global and local path planning, which uses a "generalized potential field" approach. J. Borenstein in [12] improved the artificial potential field method with certainty grids for obstacle representation. He also considered the local minima problem as "trap states" and implemented recovery routines such as wall following in order to overcome them. Despite all the advantages over conventional PFM, it was abandoned due to its limitations, such as the lack of passage between closely placed obstacles due to the local minima problem and oscillating behaviour in narrow passages [13].

To overcome these limitations, J. Borenstein and Y. Koren et al. [14] introduced the

Virtual Field Histogram (VFH) which was later improved and extended it to VFH+ [15] and VFH* [16] by I. Ulrich and J. Borenstein. These are alternative local planning approaches that avoid collisions and allows the detection of unknown obstacles while steering the robot toward the target. In addition, it handles local minima problems caused by narrow passage greater then PMF methods. The VFH can select a central path through a passage determined by the two-dimensional histogram and ignore the oscillation in steering control with the averaging effect of the polar histogram. All different VFH method based on polar histogram generation where all the area big enough to allow the robot to pass through is identified separatly and included in the cost function. This technique also comes with some drawback. The world model has to be represented as a two-dimensional histogram grid where all the sensor measurements are represented. The processing of this continuously updating data requires more memory and computation. Besides, the inaccurate sensor data can effect the stability of the robot motion. Thus unpredicted disturbances cause the robot to misidentify the environment openings and consequently may not be able to maneuver through passages.

In 2000 Ge and Gui [17] highlighted the problem of goals being unreachable with obstacles nearby when using potential field methods and presented a new repulsive function to overcome it. They also proposed modification upon conventional potential fields methods taking into consideration the relative distance between a robot and its goal to solve the problem of non-reachable goals due to nearby obstacles[17] [18].On the other hand, this method increased complexity and computation.

Other local planning approaches based on velocity space calculations are the curvature-velocity method (CVM) by R. Simmons [19] and the Dynamic Window Approach (DWA) by D. Fox et al. [20]. These approaches achieved high-speed navigation with decent collision avoidance performance. The CVM local obstacle avoidance treats the problem as one of constrained optimization in the velocity space of the robot and approximating how far the robot could travel along a given curvature before the collision. Similarly, in the case of DWA, the control commands are selected by maximizing an objective function that considers speed, goal and safety while incorporating constraints in velocity space arising from obstacles.

Concerning the physical aspect of safety, conventional or reactive robot navigation systems usually consider humans as regular static obstacles. The ability of safe navigation in an unknown environment becomes a crucial requirement for modern service robots, especially when they are employed in a social environment where the people expect the robot to behave consistently and predictably. Various research use potential fields and social cost functions in global planning level ([21],[3], [6]).

R. Kirby et al. [22] presents human social conventions, such as personal space and tending to one side of hallways, represented as constraints on the robot's navigation. E. A. Sisbot et al. [23] adjust the cost model with various aspect of safety, visibility, comfort along

with hidden zones. One of the main drawbacks of the human-aware navigation approaches mentioned above is that they do not consider the motion of humans over time. This static planning treats humans as static obstacles and requires high-frequency robot behavioural changes in order to react to uncertain human movements. This results in inconsistent robot behaviour that is not consistent with socially comfortable movement.
Several research papers have been proposed to formulate motion planning in the presence of moving obstacles. A method that works in velocity space has been suggested by Fiorini and Shiller [24] that deals with collision avoidance in the presence of moving obstacles. They discussed the velocity obstacle (VO) concept. This method lacks the simplicity of potential fields, and it considers that the obstacles are moving with a constant speed. Also, this collision avoidance strategy requires the environment to be pre-known. Similarly, Fujimura and Samet [25] presented research in motion planning in the presents of moving obstacles where the dynamic environment has to be completely pre-known. Tsoularis and Kambhampati [26] and Chakravarthy and Ghose [27] proposed another method which uses relative speed between the robot and the obstacle in order to detect collisions.

2.3 Set Point Stabilization, Trajectory Tracking and Model Predictive Control

A set point stabilization problem is a fundamental control problem which can be explained as the following: Consider the dynamical system as

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.5}$$

where $\boldsymbol{x} \in \mathbb{R}^{n_x}$ is the state, $\boldsymbol{u} \in \mathbb{R}^{n_u}$ is the input, $t \in \mathbb{R}$ is the time and the feedback is $\boldsymbol{u} = \mu(\boldsymbol{x})$ starting at $\boldsymbol{x}_0$. Design a feedback control signal $\mu : \boldsymbol{x} \mapsto \boldsymbol{u}$ where the solution $\boldsymbol{x}(t, \boldsymbol{x}_0|\mu(\boldsymbol{x}))$ converges the states to the set point $\boldsymbol{x}^{ref}$:

$$\lim_{t \to 0} \left\| \boldsymbol{x}(t) - \boldsymbol{x}^{ref} \right\| = 0 \tag{2.6}$$

This means that if the current state x is far away from the setpoint, then the controller should control the system towards the reference point, while if the current state is already near the reference, then the controller keeps it there. The set point stabilization problem is formulated in a variety of systems and has many examples in automation tasks. Well-known examples are keeping balancing vehicles like segways or ballbots upright or temperature control in buildings. Path planning for mobile robots can be formulated as a set point stabilization problem where the setpoint is a reference state that the robot should approach. This problem is usually called Set Point Tracking [28].
The Tracking problem is a similar control problem where instead of a set point the refer-

ence; $\boldsymbol{x}^{ref} : [t_0, \inf) \rightarrow \mathbb{R}^{n_x}$ is a time-varying reference state which is pre-computed offline. The task is to determine the control inputs $\boldsymbol{u}$ such that states $\boldsymbol{x}$ follows a given reference $\boldsymbol{x}^{ref}$ as well as possible, such that:

$$\lim_{t \to 0} \left\| \boldsymbol{x}(t) - \boldsymbol{x}^{ref}(t) \right\| = 0 \tag{2.7}$$

Tracking problems can also be formulated in the case of robot navigation with a reference trajectory $\boldsymbol{x}^{ref}$ to be followed, which is calculated offline and determines where and when the robot should be on this trajectory. This problem is known as Trajectory Tracking. Set Point Tracking is an intermediate problem between set point stabilization and Trajectory Tracking when the robot follows a set point which changes in the next step to another one without specifying the exact continuous transition on the trajectory [28].
For set point stabilization and Trajectory Tracking problems, a wide range of applicable control methods exist from classic linear control to advanced optimization-based control. Assuming that motion planning requires the design of a controller with the consideration of constraints on inputs and states or desires to predict the future behaviour of a system, only a few controller design methods can be considered.
Model Predictive Control (MPC) is a very powerful optimal control design approach, where the applied control actions are chosen based on repeated predictions of the future system behaviour. Generally, MPC built upon an optimal control problem (OCP) that calculates the best control actions by minimization an objective function and predicting the system future behaviour. Furthermore, MPC has many advantages from a system and control point of view with the consideration of [29]. One of the main advantage of an MPC that the constraints can be defined explicitly in the algorithm. The predicted states and control inputs can be bounded by constraints separately, to make it possible to neglect unwanted results. MPC is suitable for multiple inputs and outputs in non-linear systems. The online optimization respect to the cost criteria in order to find the most optimal solutions. The general idea of any MPC formulation is built upon an iterative control scheme that is executed by the following:

1. Obtaining the state measurement $\boldsymbol{x}(n)$ at every sampling instant n
2. At each sampling instant n, the controller optimizes the predicted future behaviour of the dynamical system over a finite time horizon $k = 0, .., N-1$ of length $N \geq 2$ and find a set of admissible control inputs considering constraints on states and inputs based the minimization on the objective function.
3. Applying the first element of the resulting optimal control sequence $\boldsymbol{u}^{\star}$.

This scheme will be further detailed in the Chapter 4. The above scheme shows that the central concept of MPC is based on a repetitive solution of an optimal control problem. Note, that its different from an optimal feedback controller which often turns to be

infeasible for non-linear systems, since solving the partial differential equations can be difficult [30]. Instead, the goal of an MPC is a simpler computation of an open-loop input trajectory at each step for the given state.
Nowadays, MPC problem formulation is a widespread discussion in scientific researches, but most MPC research is considered as set point stabilization ([27], [31]). For example, this frequently appears in problems such as flight stabilization [32] or HVAC large scale heating and cooling system control [33]. Other researches consider the change of the set point as a disturbance on the reference states and handle this stabilization problem with MPC formulations ([34] ,[35], [36]). Specifically regarding setpoint stabilization, it has been used for localization purposes by Nishio et al. in an MPC implementation taking into consideration moving obstacles, represented by fuzzy potential fields [37], given obstacles with constant velocity.
While trajectory tracking is a well-known problem, there are only a few related works that use trajectory tracking MPC design ([38], [39], [40]). The approaches in these researches show several shortcomings and challenges. The constraints in simultaneous path planning and tracking MPC algorithms are only sparsely considered and require long prediction horizons. Also, the trajectory tracking MPC is highly dependent on the time-varying reference signal, which might result in significant errors and less robust performance. As a middle ground between set point stabilization and trajectory-following or -tracking is the problem of path-following. Path-following allows for more freedom in the MPC formulation. Where in trajectory-following the MPC has a path with set times for each position, in path-following the controller is fed a curve which the robot should try to follow, but freedom in how much it accelerates and when. An example of such path-following is described by Kølbæk, in which a ball-balancing robot is made to follow a given path specified by a polynomial, coefficients of which are fed to the MPC [1]. As well Kanjanawanishkul et al. proposes an MPC for path-following scenario for an omnidirectional robot, similar to the work of Kølbæk, though with omnidirectional wheels [41]. This work by Kanjanawanishkul shows advantageous performance grounded in the ability for the MPC to handle system and input constraints while generating an optimal control sequence, though no obstacles are yet considered. In the following section the hardware available for this project is investigated.

2.4 Project Hardware

To solve the problems posed in circumnavigating an environment with dynamic obstacles, the TIAGo robot is used. Designed to work in indoor environments, combining mobility, perception, manipulation and HRI in one platform, TIAGo is a service robot. The TIAGo robot has different configurations based on what is needed for implementation, Base, Iron, Steel and Titanium. For this project, the Base and Steel versions are available. Of which

the Base is considered mainly, as it has all the sensor capabilities needed for the navigation package. The robot has a LiDAR capable of sensing its surroundings along with actuators to enable mobility and odometry sensors to provide movement feedback. The robots used in this project can be seen in Figure 2.3.

Figure 2.3: The robots provided for this project by AAU.

The hardware which provides the capabilities needed for robot navigation is further detailed in Table 2.2.

	TIAGo Base	TIAGo Steel
Sensors	Front LiDAR (5.6m range) 6 DoF IMU 3x Rear stereo sensors (1m range) Actuator current feedback (wheels)	Front LiDAR (5.6m range) 6 DoF IMU 3x Rear stereo sensors (1m range) Actuator current feedback (wheels and arm) RGB-D camera (head)
Computer Hardware	Intel i5 CPU 250GB SSD	Intel i5 CPU 250GB SSD
Software	Ubuntu 64bit ROS LTS	Ubuntu 64bit ROS LTS Arm controller (position/velocity/effort)
Max Speed	1.5 m/s	1 m/s

Table 2.2: Hardware specifications for TIAGo robots.

The TIAGo runs using an internal computer running ROS, providing a framework for communication between the processing unit, sensors and actuators. Through ROS, a user can access data gathered by the various sensors on the robot as well as sending commands to the actuators to perform some desired motion.

In order to enhance upon the envisioned package, the added capabilities of environment manipulation or further HRI provided by the Steel robot configuration could be implemented. The hardware present on the robot can be seen in Figure 2.4.

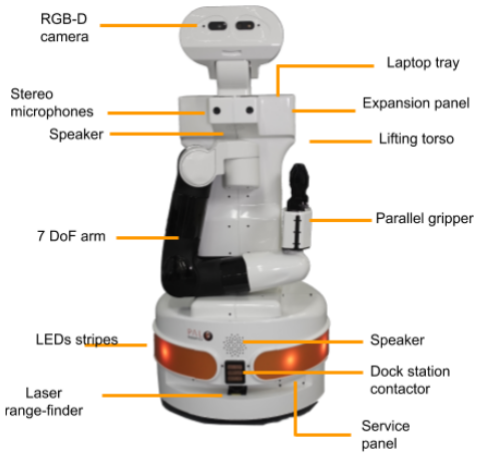


Figure 2.4: Hardware and sensor arrangement on the TIAGo Steel.

2.5 Problem Statement

The problem at hand of enabling a robot to navigate a dynamic environment sets up an array of problems which has to be solved. Firstly internal communication on the platform has to be established, using the sensors and actuators to get feedback on the state of the robot for this purpose ROS is used as a framework for system communication. Per the investigated methods for motion planning a decision is made to work with model predictive control, given the robot available for the project being a non-holonomic robot a non-linear model will be needed. From this arises the need of non-linear model predictive control(NMPC), so *how can NMPC be implemented in a robotic system as a local planner?* A representation of the environment has to be established, representing the various obstacles surrounding the robot, be they static or dynamic. This being necessary to establish a controller capable of circumnavigating obstacles in its environment. *How can an NMPC local planner be made to consider static and moving obstacles?*
Given the project being developed as parallel work as part of the work done by the AAU-HAMR-lab an important aspect is introduction of the developed product as a part of the framework used by the HAMR-lab. Per implementation in the ROS framework the question is posed of *How can an NMPC local planner be implemented on the robot as a part of a navigation package?*
Given the nature of the dynamic obstacles being human actors in the robot environment the robot has to be able to navigate the environment considering various sociodynamic factors, to ensure the comfort of people interacting with the robot in its environment.

CHAPTER 3

MODELLING

This Chapter will regard all the mathematical modelling done in this project. A non-linear kinematic model is designed for the robot and a linear state-space model for the pedestrians. These models will later be used in the local planner to predict the movement of the pedestrians and make the robot move according to these predictions. In section 3.2, the discretization method used in this project will also be described. After the models for the movement of the robot and obstacles have been formulated the representation of these obstacles is described. This representation is then used in the controller design for the local planner such that the local planner can account for and avoid both moving and static obstacles.

3.1 TIAGo Robot Model

The TIAGo robot is a differential drive robot, which means it has two wheels mounted on a common axis, and each wheel can move independently of each other. By varying the velocities of each independent wheel, different trajectories can be achieved. The simple kinematic model can be seen in Figure 3.1.

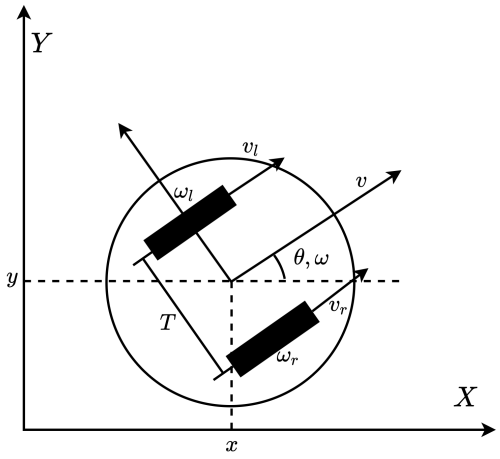


Figure 3.1: The differential drive scheme of the robot.

In Figure 3.1 (x, y) denotes the local position of the robot in (X, Y) global coordinate system. θ is the difference in orientation from the global coordinate frame. T is the track

length of the differential drive robot. The linear velocities v_l and v_r are the product of the angular velocity of the wheels and the radius of the wheels r_w.

$$v_l = \omega_l \, r_w \tag{3.1}$$

$$v_r = \omega_r \, r_w \tag{3.2}$$

Due to the wheels being on a common axis the movement of the robot can be purely rotational ($v_l = -v_r$) or translational ($v_l = v_r$). This can also be seen in Figure 3.2.

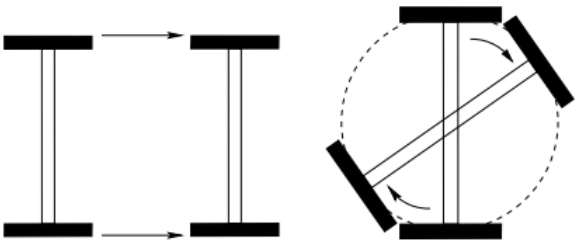


Figure 3.2: The pure translation or pure rotation of the robot given certain inputs [42].

Since the robot is a differential drive robot and assuming no slip, there will be no lateral velocity. The motion of the robot can then be described by longitudinal linear velocity from the centre of the robot and the angular velocity. This linear velocity is formulated as:

$$v = \frac{v_l + v_r}{2} \tag{3.3}$$

The angular velocity of the centre of the robot is defined as:

$$\omega = \frac{v_r - v_l}{T} \tag{3.4}$$

Respectively, the change of positions in the X and Y direction in the global frame is given as:

$$\dot{x} = v \, \cos(\theta) \tag{3.5}$$

$$\dot{y} = v \, \sin(\theta) \tag{3.6}$$

Moreover the change of orientation between the local frame and the global frame is defined by:

$$\dot{\theta} = \omega \tag{3.7}$$

The continuous-time model in state-space form can be seen in Equation 3.8.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\,\cos(\theta) \\ v\,\sin(\theta) \\ \omega \end{bmatrix} \tag{3.8}$$

The state-space model can then be reformulated with the input defined in a separate vector.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.9}$$

3.2 Discretization of the model

The Runge-Kutta method is used to discretize the model of the robot; more specifically, the fourth order Runge-Kutta method is used. The Runge-Kutta method consists of multiple iterative steps starting with the Euler method. The four steps can be seen in the following Equation 3.10.

$$\begin{aligned} \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{1}{6}\left(k_1 + 2\,k_2 + 2\,k_3 + k_4\right) \\ k_1 &= T_s\left(A\,\boldsymbol{x}_n + B\,\boldsymbol{u}_n\right) \\ k_2 &= T_s\left(A\left(\boldsymbol{x}_n + \frac{k_1}{2}\right) + B\,\boldsymbol{u}_n\right) \\ k_3 &= T_s\left(A\left(\boldsymbol{x}_n + \frac{k_2}{2}\right) + B\,\boldsymbol{u}_n\right) \\ k_4 &= T_s\left(A\left(\boldsymbol{x}_n + k_3\right) + B\,\boldsymbol{u}_n\right) \end{aligned} \tag{3.10}$$

Where k_1 to k_4 are the steps of the Runge-Kutta method, T_s is the step size, A is the system matrix, B is the input matrix, $\boldsymbol{x}$ is the state, and $\boldsymbol{u}$ is the input. Substituting these equations into each other gives the following expression for the numerical solution to the next time step as shown in Equation 3.11.

$$\begin{aligned} \boldsymbol{x}_{n+1} = &\left(I + T_s\,A + \frac{T_s^2}{2!}A^2 + \frac{T_s^3}{3!}A^3 + \frac{T_s^4}{4!}A^4\right)\boldsymbol{x}_n \\ &+ \left(T_s\,I + \frac{T_s^2}{2}A + \frac{T_s^3}{3!}A^2 + \frac{T_s^4}{4!}A^3\right)B\,\boldsymbol{u}_n \end{aligned} \tag{3.11}$$

This discretization method is simple to implement as well as fast and can be used for the

robot model. The discretized model derived in this chapter will then be used in the local planner designed in this project.

3.3 Pedestrian Model

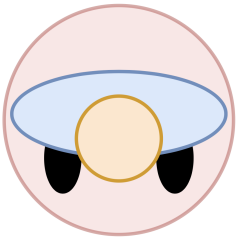
The people/pedestrians are modelled to have constant velocity and moving linearly in each time step. This moving obstacle will have a constant heading and variable velocity. Using these parameters, a linear state-space model can be derived for the motion of the obstacles.

$$\begin{bmatrix} x_{k+1}^{mo} \\ y_{k+1}^{mo} \end{bmatrix} = \begin{bmatrix} x_{k}^{mo} \\ y_{k}^{mo} \end{bmatrix} + \begin{bmatrix} \cos\theta^{mo} \\ \sin\theta^{mo} \end{bmatrix} v^{mo} \tag{3.12}$$

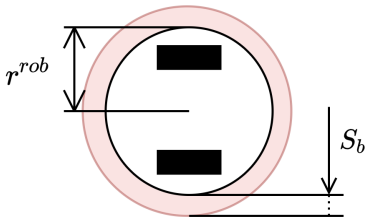
In Equation 3.12, θ^{mo} is the constant angle between the orientation of the obstacle and the global frame. v^{mo} denotes the velocity of the obstacles. This model can be expanded to a five-state model with varying velocity, radius and heading. If a perception package for detecting people and estimating their position, heading and direction is implemented. The varying radius of the obstacle could then represent the uncertainty in the estimation. However, due to the fact that a perception package fitting the needs of this project is not available, the two-state model in Equation 3.12 is implemented.

3.4 Moving Obstacle Representation

Mobile robots equipped with various sensors can detect their environment, and analyze the measured data in order make decisions. For service robots that are running in a dynamic environment populated with pedestrians it is essential to consider these obstacles to circumnavigate them. This Section revolves around how obstacles can be represented so they can be used in the path planning algorithms.



(a) An encirclement that is slightly larger than the shoulder width of a person.



(b) A variable safety boundary is employed around the robot footprint

Figure 3.3: Representation of the moving obstacles around pedestrians and the consideration of safety boundary S_b around the robot perimeter which acts as an extension of the robot radius r^{rob}.

One of the most trivial representation of an obstacle is the point-shape obstacles where the obstacle considered as a point with a radius that determines the size of it. To represent pedestrians in a two-dimensional map this is one of the primary methods. In Figure 3.3a, an encirclement about the shoulder width of a person is used to represent the pedestrian as a point-shape obstacle.
Using a convex structure for representing the moving obstacles is advantageous when defining the moving obstacle as a constraint. In Figure 3.3b, the robot footprint can be seen with a safety boundary denoted as S_b simply added onto the robot radius r^{rob}. The robot will have a safety distance implemented to make sure that it does not get close to any obstacle to avoid collision, and to prevent discomfort for the person moving in the same space as the robot. This also compensates for some of the modelling uncertainties such that the probability of collision is reduced. We are considering two main types of obstacle. Moving obstacles are obstacles which have a heading and velocity while static obstacles are fixed, defined as non-traversable positions in space.
The moving obstacles and the robot can be defined as circles. When both are defined as circles, they can be constrained, as two points that have to have a set minimum distance between them. The distance between the moving obstacle and the robot can be seen in Figure 3.4.

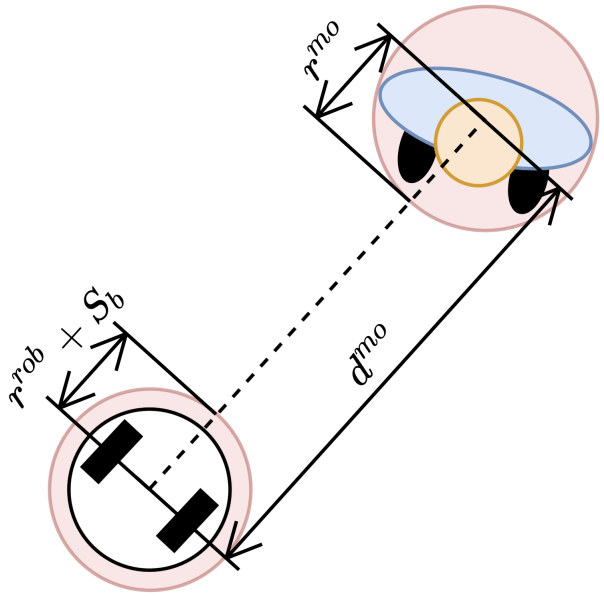


Figure 3.4: The distance d^{mo} between the robot and the moving obstacle

The distance in Figure 3.4 is denoted d^{mo}, the radius of the moving obstacle is denoted r^{mo}, and the robot and safety boundary is denoted $r^{rob} + S_b$. This distance can be set as

a constraint, as seen in Equation 3.13.

$$d^{mo} \geq r^{rob} + S_b + r^{mo} \tag{3.13}$$

The movement of these obstacles is described by the state space pedestrian model in Section 3.3. The next Section will describe how the static obstacles are found and represented in this thesis.

3.5 Static Obstacle Representaion

For identifying these obstacles, the robot has a costmap based on prior mapping of the room, and the static obstacles in the environment during runtime. This costmap is then converted to obstacles using the `costmap_converter` node. This node converts every obstacle recognized in the environment to polygons, and thereby, the environment can be perceived as a list of polygons to circumnavigate. There are several different plugins for the `costmap_converter` these are listed below:

- `CostmapToPolygonsDBSMCCH`
- `CostmapToPolygonsDBSConcaveHull`
- `CostmapToLinesDBSMCCH`
- `CostmapToLinesDBSRANSAC`

The plugin used in this thesis is the `CostmapToPolygonsDBSMCCH` because it creates simple polygons out of the costmap. Three algorithms are used to create static obstacles from the costmap. Firstly, a clustering algorithm is used to sort the costmap into clusters that can be made into obstacles. Secondly, an algorithm creating a convex hull around these clusters. Lastly, an algorithm which reduces the complexity of the polygons. The clustering algorithm used in this plugin is the Density-Based Spatial Clustering of Applications with Noise(DBSCAN) [43] this will be described briefly in the following. DBSCAN sorts points into three categories: *core points*, *reachable points* and *outliers*. A parameter for the neighbourhood of a point is denoted as ε. The classification is the following:

- A point p is a *core point* if at least `minPts` points are within the distance ε of the point p.
- A point q is directly reachable from p if point q is within distance ε from the *core point*. Points can only be said to be directly reachable from core points.
- A point q is *reachable* if there is a path $p_1, ..., p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i.
- All points not reachable from any other point are *outliers*.

The parameters which can be tuned in this clustering algorithm are the neighbourhood size, and the `minPts` required for a point q to be classified as a *core point*. An illustration of the DBSCAN algorithm with `minPts` $= 4$ can be seen in Figure 3.5.

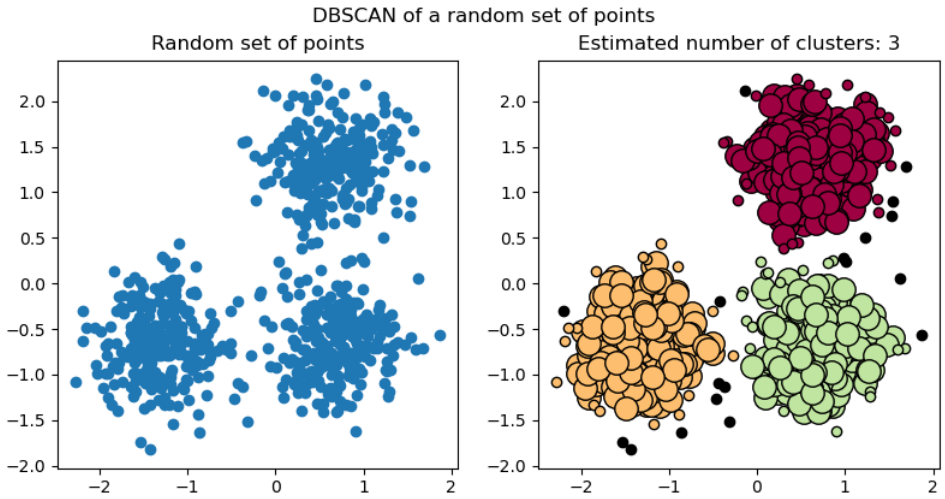


Figure 3.5: DBSCAN algorithm on a cluster of points with parameters $\varepsilon = 0.3$ and `minPts` $= 4$ made in python.

Once the points have been sorted into clusters, the convex hull is computed using Sklansky's algorithm(modified by S. Y. Shin and T. C. Woo) for finding a convex hull in linear time [44]. The input to the algorithm is a set of n points $P = V_0, V_1, ..., V_{n-1}$, and the output is a so-called *"zipper"*. A *zipper* is a non-self-intersecting, concave chain of line segments. A *zipper* for a set of n points/vertices is denoted as $ZPR(V_0, V_n)$. The algorithm starts by choosing an *extreme point*/vertex to be the first entry in the zipper. Then the algorithm traverses the set of points/polygon in the clockwise direction and classifies each point by one of the three cases:

- Case 1: vertex of the given set of points is added to the zipper
- Case 2: vertex of the set of points is not added to the zipper
- Case 3: zipper vertex is deleted

After complete traversal of the polygon/set of points, the constructed *zipper* constitutes the convex hull of the set of points. In order to classify each case, some definitions have to be made. A directed line segment belonging to the *zipper* is defined as $L(Z_{j-1}, Z_j)$ where Z_j is the *jth* entry in the *zipper*. Another definition is an *edge* of P and can be denoted as $E(V_i, V_{i+1})$, a directed line segment between two adjacent points. The algorithm looks for points that are to the right of the line segments to determine whether it is an *extreme point* of the set of points. In Algorithm 3.1, there are two checks to see if the point falls to the right side of the line. If so, the point is added to the *zipper*. In the next iteration,

the next point is checked. If the next point is to the left of the line segments, then it is known that the last point was not an *extreme point* and it is then deleted from the zipper, and the new point is added to the *zipper* instead. This process iterates until V_0 is reached again.

Algorithm 3.1: Sklansky's algorithm

```
Result: ZPR(V_0, V_q)
1  V_0 ← Z_0, V_1 ← Z_1, j ← 1, q ← 1;
2  while V_q ≠ V_0 do
3      if V_{q+1} is to the right of L(Z_{j-1}, Z_j) then
4          if V_{q+1} is to the right of E(V_{q-1}, Z_j) then
5              j ← j + 1;
6              Z_j ← V_{q+1};
7              q ← q + 1;
8          end
9          else
10             while V_{q+1} is on or to the right of L(Z_{j-1}, Z_j) do
11                 q ← q + 1;
12             end
13         end
14     end
15     else
16         while Z_j ≠ V_0 and Z_{j-1} is not to the right of L(Z_j, V_{q+1}) do
17             j ← j − 1;
18         end
19     end
20     j ← j + 1;
21     Z_j ← V_{q+1};
22     q ← q + 1;
23 end
```

A Figure showing Sklansky's algorithm can be found in Appendix A Figure A.1. Once the convex polygons have been created using this algorithm, they need to be reduced further using the Ramer–Douglas–Peucker(RDP) algorithm for decimating a curve composed of line segments to a similar curve with fewer points [45]. This algorithm takes in a set of n points on a curve $C = (P_1, P_2, ..., P_n)$ and the distance dimension ε where:

$$\varepsilon > 0 \tag{3.14}$$

This algorithm first considers the simplest approximation possible, namely a straight line between P_1 and P_n. The point furthest from the line segment will then be selected as the

first point included in the set of points that make up the simplified curve. This can also be seen in Figure 3.6.

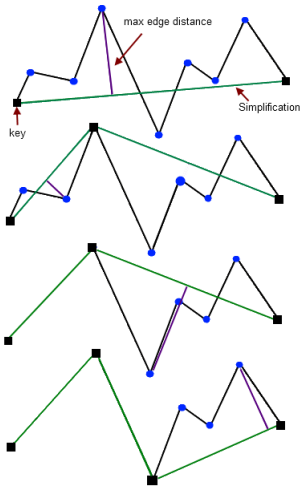


Figure 3.6: The RDP algorithm called recursively on the line segments until all points that are above the ε will be included in the approximation [46].

This recursively finds the point furthest from the new line segment and adds it to the simplified path. If a point is within the ε distance from the simplified line, it is omitted from the next iteration of the algorithm. So by merely giving an input number of points and a certain ε, a simplified curve/polygon can be constructed. Furthermore, the greater the ε distance is the lower the fidelity/resolution the curve/polygon will have. The result of the `costmap_converter` can be seen in Figure 3.7.

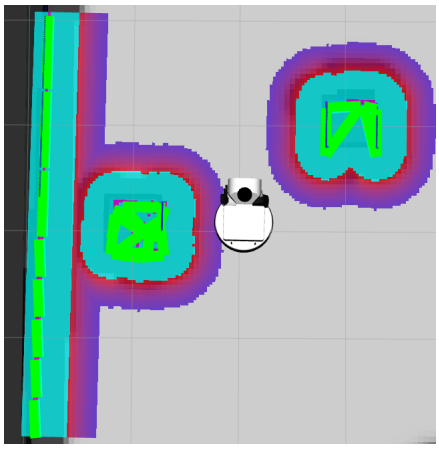


Figure 3.7: A costmap conversion of the local costmap in RViZ, recorded by the robot. The robot is facing upwards, with two cupboards as obstacles in the environment and a wall of the environment. Original costmap in cyan and the polygon obstacles in green

Using this polygon representation of the environment's obstacles, two different obstacle representations were developed. One is a simplified obstacle representation, performed by finding the centroid of the polygons making up the obstacles in the environment and encircling them, effectively turning them into circles. The other is to calculate the closest point in each polygon to the robot, thereby simply knowing how far to stay away from that point at any given time.

3.5.1 Polygon Centroid Representation

The polygon centroid representation method is used to simplify the representation of static obstacles, through taking obstacles from a polygonal shape and turning them into circles. The first step in creating the encircled polygons is to find the area of the given polygon, calculated by Equation 3.15. [47]

$$a_{so} = \frac{1}{2} \sum_{i=1}^{n_{so}} x_{i-1}^{so} y_i^{so} - x_i^{so} y_{i-1}^{so} \tag{3.15}$$

Herein x_i^{so} and y_i^{so} denote the position of the i'th vertex of the polygon, n_{so} being the maximum number of vertices for the given polygon. Once having the area of the polygon, the next step is to calculate the first moments of the polygon, giving the position (x^{cent}, y^{cent}) of the centroid.

$$\begin{aligned} x^{cent} = \frac{1}{a_{so}} \iint_R x^{so}\, dx^{so}\, dy^{so} &= \frac{1}{a_{so}} \int_b -x^{so} y^{so}\, dx^{so} \\ &= \frac{1}{6a_{so}} \sum_{i=1}^{n_{so}} (x_{i-1}^{so} + x_i^{so})(x_{i-1}^{so} y_i^{so} - x_i^{so} y_{i-1}^{so}) \end{aligned} \tag{3.16}$$

$$\begin{aligned} y^{cent} = \frac{1}{a_{so}} \iint_R y^{so}\, dx^{so}\, dy^{so} &= \frac{1}{a_{so}} \int_b -x^{so} y^{so}\, dx^{so} \\ &= \frac{1}{6a_{so}} \sum_{i=1}^{n} (y_{i-1}^{so} + y_i^{so})(x_{i-1}^{so} y_i^{so} - x_i^{so} y_{i-1}^{so}) \end{aligned} \tag{3.17}$$

Herein R being the region of the polygon, b is the border of the polygon and a_{so} again is the area. After the centroid has been found, in order to find the radius that encircles the whole polygon, the distance between the centroid of the polygon, and the vertices is calculated in Equation 3.18.

$$d_i^{cent} = \sqrt{(x^{cent} - x_i^{so})^2 + (y^{cent} - y_i^{so})^2} \tag{3.18}$$

Comparing all the distances d_i^{cent}, the longest distance is used as the radius of the circle

$r^{so} = max(d^{cent_i})$ made to encircle the obstacle.
For smaller obstacles, different centroid calculation is implemented, specifically for obstacles small enough to constitute just one point. The method used in this case is simply defining the point obstacles as their centroid and assigning a minimum size to them to make sure at least a minimum distance is kept from them.
A representation of the polygon centroid method, as implemented in MATLAB, can be seen in Figure 3.8.

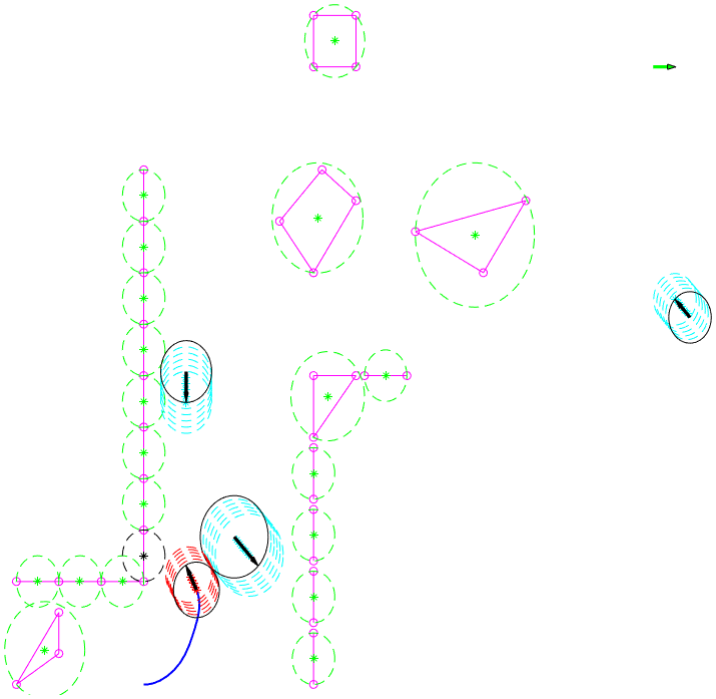


Figure 3.8: The polygon centroid method for static obstacle representation. The circles encompassing the polygons show the altered obstacles (green). The circles with a cyan horizon are moving obstacles and the red horizon belongs to the robot.

3.5.2 Closest Point of Polygon Representation

In this Section, an unprecedented obstacle representation, called the closest point of polygon obstacle method for static obstacles, is explained. This method takes the polygon-shaped obstacles to form the `costmap_converter` plugin and instead of calculating the centroid and encircle the polygon which inflates the real occupied area as was done in the centroid representation (Section 3.5.1), it takes only the closest part of the obstacle at any time into account.
Firstly, the polygon obstacles are evaluated per the number of considered polygons and how many vertices they have. The NMPC local planner only consider a limited number of static obstacles, denoted as n_{so}, from the local costmap. The reason behind this is

because the proposed method is computationally heavy, compared to the centroid polygon representation. Another difference compared to the previously introduced method is, that by considering only the closest point of the polygon to the robot position, the obstacle radius is neglected ($r^{mo} = 0$) and the distance between the obstacle and the robot are directly measured to the side of the polygon as it is shown in Figure 3.9.

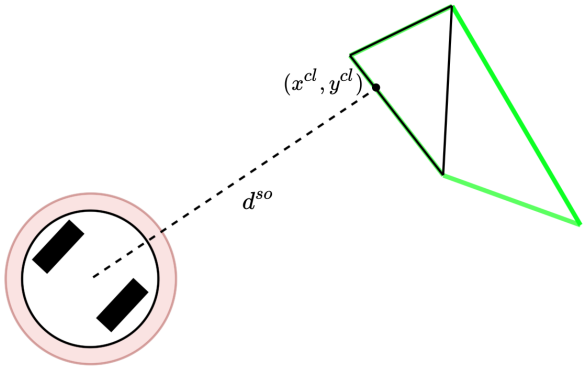


Figure 3.9: Closest point to polygon representation method showing the distance between the closest point and the robot. The green polygon is the polygon provided by the `costmap_converter` while the black triangle is the maximum three sided polygon considered by the NMPC local planner.

The NMPC local planner runs the algorithm, called ClosestPoint2Polygon, presented in Algorithm 3.2 in order to find the closest point of the polygon edge. Using this point to set the constraints for the optimal control problem which keeps the distance between the robot and the closest point of the polygon (x^{cl}, y^{cl}). The method to set these constraints is explained in Section 4.5.2.
The polygon obstacles given to the ClosestPoint2Polygon algorithm has to come in a special form. The obstacles provided by the `costmap_converter` are polygons represented simply by their vertices, which has to be reduced in order to decrease the computational burden. An algorithm called ClosestNTriangle finds the closest n_{so} polygons by the measured distances to the polygon centroids provided by the PolygonCentroid algorithm. This function also calculates the distance between the robot and each of the polygon vertices in order to find the three closest ones, effectively reducing complex polygons to three sided triangles. Sometimes the `costmap_converter` provides point and line shape obstacles. These have to be handled differently compared to polygons with more vertices. The output of the ClosestNTriangle algorithm is the polygons converted into a vector with the following 6 element:

$$\boldsymbol{x}^{so} = [x_1^{so}, y_1^{so}, x_2^{so}, y_2^{so}, x_3^{so}, y_3^{so},] \tag{3.19}$$

In Equation 3.19 the values (x_i^{so}, y_i^{so}) where $i = 1, 2, 3$ are the calculated closest vertices.

In the special cases when the polygons provided by the `costmap_converter` are lines the values of (x_3^{so}, y_3^{so}) are filled up with zeros, since they will be neglected. In the case of the point obstacles only the (x_1^{so}, y_1^{so}) values are filled with the position of the point obstacle while the rest are zeros. The algorithm ClosestNTriangle includes as an extra output S^{so}, called state size, which includes a number 1,2 or 3, defining the polygon complexity. Both the vector $\boldsymbol{x}^{so}$ and the variable S^{so} along with the current robot pose $\boldsymbol{x}^{rob} = \left[x^{rob}, y^{rob}\right]$ are required inputs for Algorithm 3.2 in order to calculate the closest point to the robot.

Algorithm 3.2: ClosestPoint2Polygon algorithm

Input : $\boldsymbol{x}^{so} = [x_1^{so}, y_1^{so}, x_2^{so}, y_2^{so}, x_3^{so}, y_3^{so},]\,, S^{so}, \boldsymbol{x}^{rob} = \left[x^{rob}, y^{rob}\right]$
Output: x^{cl}, y^{cl}

Function ClosestPoint2LineSegment$(x_R, y_R, x_A, y_A, x_B, y_B)$**:**
  $d_{AB_x} \leftarrow x_B - x_A$;
  $d_{AB_y} \leftarrow y_B - y_A$;
  $d_{AR_x} \leftarrow x_R - x_A$;
  $d_{AR_y} \leftarrow y_R - y_A$;
  $sq_{AB} \leftarrow \max(0.001, d_{AB_x}^2 + d_{AB_y}^2)$;
  $dot_{AB-Ar} \leftarrow d_{AB_x}\, d_{AR_x} + d_{AB_y}\, d_{AR_y}$;
  $p_{diff} \leftarrow \max(0, \min(1, \frac{dot_{AB-Ar}}{sq_{AB}}))$;
  $x_{cl} \leftarrow x_A + d_{AB_x} \cdot p_{diff}$;
  $y_{cl} \leftarrow y_A + d_{AB_y} \cdot p_{diff}$;
  return x_{cl}, y_{cl}
if $S^{so} < 3$ **then**
  if $S^{so} < 2$ **then**
    $(x^{cl}, y^{cl}) \leftarrow (y_1^{so}, x_1^{so})$
  else
    $x^{cl}, y^{cl} \leftarrow$ ClosestPoint2LineSegment$(x^{rob}, y^{rob}, x_1^{so}, y_1^{so}, x_2^{so}, y_2^{so})$
  end
else
  $x_{p1}, y_{p1} \leftarrow$ClosestPoint2LineSegment$(x^{rob}, y^{rob}, x_1^{so}, y_1^{so}, x_2^{so}, y_2^{so})$;
  $x_{p2}, y_{p2} \leftarrow$ClosestPoint2LineSegment$(x^{rob}, y^{rob}, x_2^{so}, y_2^{so}, x_3^{so}, y_3^{so})$;
  $x_{p3}, y_{p3} \leftarrow$ClosestPoint2LineSegment$(x^{rob}, y^{rob}, x_3^{so}, y_3^{so}, x_1^{so}, y_1^{so})$;
  $dist_1 \leftarrow \sqrt{(x^{rob} - x_{p1})^2 + (y^{rob} - y_{p1})^2}$;
  $dist_2 \leftarrow \sqrt{(x^{rob} - x_{p2})^2 + (y^{rob} - y_{p2})^2}$;
  $dist_3 \leftarrow \sqrt{(x^{rob} - x_{p3})^2 + (y^{rob} - y_{p3})^2}$;
  $x^{cl}, y^{cl} \leftarrow$ MinSearch$(dist_1, dist_2, dist_3)$;
end

The ClosestPoint2Polygon algorithm classifies the polygons by the number of ver-

tices. For single-point obstacles the closest point is the point itself. For lines (two vertices) the ClosestPoint2LineSegment algorithm finds the closest point to the line segment by considering the vector from the start to the end of the line $\vec{AB}$ and the vector from the start of the line to the robot position $\vec{AR}$. The square of the magnitude of the vector $\vec{AB}$, denoted with sq_{AB}, is calculated. This square is given a minimum value of 0.001 to avoid division by 0 in the case of the polygon vertices coinciding due to some error related to `costmap_converter` giving the first and last vertices as the same position. The scaling factor p_{diff} is then calculated by dividing the dot product of the vectors $\vec{AB} \cdot \vec{AR}$ with the squared magnitude (sq_{AB}). By adding the dot product $\vec{AB} \cdot p_{diff}$ to the starting point of the line, the closest point on the line segment is calculated.
For the three sided polygon obstacles, the ClosestPoint2LineSegment function is simply performed for each line segment constituting the polygon. The distance between the robot and each point is evaluated, to determine which one is the closest point (x^{cl}, y^{cl}). Algorithm 3.2 has been develop and implemented into the NMPC local planner. The MATLAB simulation in Figure 3.10 show how the closest point to polygon representation works in practice.

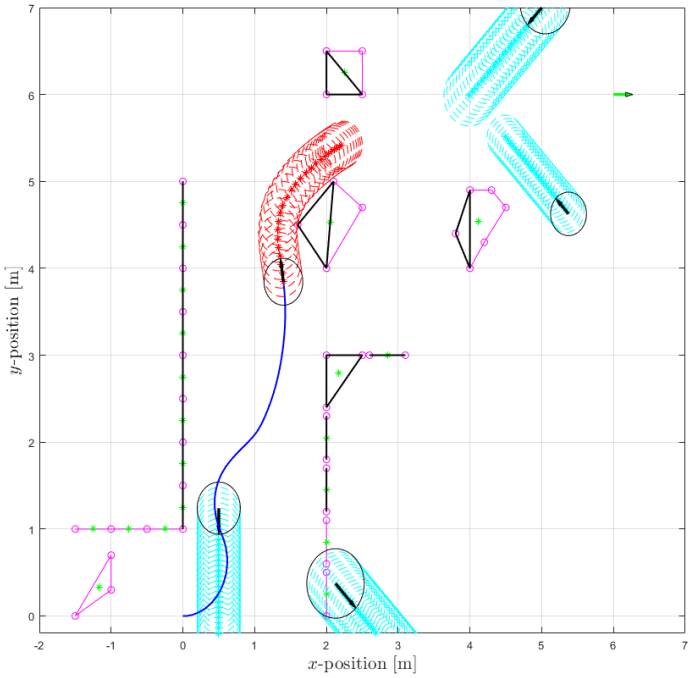


Figure 3.10: An environment with moving obstacles represented by circles (cyan), and static obstacles represented as polygons (magenta), the black polygons being the ones currently considered by the NMPC. The red horizon being the robot prediction horizon.

CHAPTER 4

NMPC LOCAL PLANNER

This chapter focuses on the formulation of a Non-linear Model Predictive Control (NMPC) problem developed for the local planning of the TIAGo robot. The general ideas of stabilization and tracking NMPC formulations are derived from [28]. Firstly, the principal ideas and advantages of an NMPC local planner will be explained in Section 4.1. Secondly, the basic formulation of an optimal control problem is introduced in Section 4.2. In Section 4.3, the basic NMPC algorithm is derived for set point stabilization and trajectory tracking problems. Later in Section 4.4 the discretization of the NMPC is investigated. In addition, in this Section an improved direct approach called the direct multiple shooting method, which was proposed by Bock and Plitt in ([48] is explained, in order to handle non-linear propagation, which makes the system predictions highly non-linear. This method overcomes this issue by lifting of the previously used direct approach, called the direct single shooting method, which is used in the NMPC formulations in Section 4.3. Furthermore, in Section 4.5, constraints on the states and inputs of the optimal control problem are defined along with the static and moving obstacle constraints for dynamic obstacle avoidance. Finally, the Section 4.6 summarizes the previously introduced methods and presents the final formulation used for the NMPC path planning algorithm.

4.1 Introduction of NMPC Local Planner

In this thesis, the NMPC is applied as a local planner in the TIAGo robot's navigation algorithm. The local planner should receive the calculated global trajectory, which is an optimal path calculated by a different type of path planning algorithm like A* or RRT provided by the global planner. The local planner will generate the velocity commands to the TIAGo velocity controller in order to follow the globally desired reference trajectory while satisfying constraints and avoiding obstacles.
A path planner which utilizes an MPC scheme can be solved by integrating the MPC into an advanced global planner (RRT) which also executes the control commands to the vehicle, using a small segment of the predicted trajectory in the MPC's optimal control problem like as it was introduced in [49]. However, the disadvantage of this solution is the computational burden. Instead, the solution proposed in this thesis is an NMPC local planner that takes a small segment of the globally optimal trajectory provided by the global planner algorithm and computes the velocity commands that accounts for constraints while avoiding collision with obstacles.

When moving in an environment with dynamic obstacles, it is desirable to predict the future motion of the robot in order to navigate optimally. The NMPC problem is an advanced optimization-based method which allows finding the most optimal trajectory for the robot's future motion by minimizing a cost function. However, this is not the only reasons why an NMPC Local planner is the focus of this research. Generally, one of the main reasons behind the success of all MPC formulation is the ability to take constraints into account explicitly. This allows consideration of different constraints at each prediction. It follows that by assuming that the future movement of a pedestrian is also predictable, an NMPC local planner would be able to consider these predictions and set the constraints in order to avoid the pedestrian in an optimal and socially acceptable manner.

4.2 Optimal control problem

The idea of a predictive local planning strategy is to utilize an iterative optimization method in order to compute the control inputs, that will ensure an optimal future behaviour of the robot motion model. Optimal control problems(OCP) are used in many industrial applications, for example in design and operation of technical systems such as large scale heating, ventilation and cooling(HVAC) systems, bridges, aircraft and cars. In dynamic optimization or optimal control, solutions are sought for decision-making problems constrained by ordinary differential equations (ODE). A basic optimal control problem constrained by an ODE is defined as follows:

$$\begin{aligned} \underset{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot),p}{\text{minimize}} \quad & \int_0^N L(\boldsymbol{x}(t),\boldsymbol{u}(t),p)dt \\ \text{subject to} \quad & \left.\begin{aligned} &\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t),\boldsymbol{u}(t),p), \\ &\boldsymbol{u}(t) \in \mathcal{U}, \quad \boldsymbol{x}(t) \in \mathcal{X}, \end{aligned}\right\} t \in [0,N] \\ & \boldsymbol{x}(0) \in \mathcal{X}_0, \quad \boldsymbol{p} \in \mathcal{P} \end{aligned} \tag{4.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ is the differential states, $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ is the control inputs, and $\boldsymbol{p} \in \mathbb{R}^{n_p}$ is a vector of free parameters in the model. In this consideration, the OCP has a Lagrange term (L) and an ODE with initial conditions $\mathcal{X}_0$. Furthermore, there are admissible sets for the states $\mathcal{X}$, control $\mathcal{U}$ and parameters $\mathcal{P}$.
The OCP can be efficiently solved by a direct approach, where the OCP in Equation 4.1 is transcribed into a non-linear program (NLP)

$$\begin{aligned} \underset{\boldsymbol{\omega}}{\text{minimize}} \quad & J(\boldsymbol{\omega}) \\ \text{subject to} \quad & g(\boldsymbol{\omega}) = 0, \quad \boldsymbol{\omega} \in \mathcal{W} \end{aligned} \tag{4.2}$$

where $\boldsymbol{\omega} \in \mathbb{R}^{n_\omega}$ is the decision variable, J is the objective function, and $\mathcal{W}$ is the interval set. Genarally, in NLP problems we are looking for the optimal value of $\boldsymbol{\omega}$ which minimizes the objective $J(\boldsymbol{\omega})$:

$$\boldsymbol{\omega}^\star = \underset{\boldsymbol{w}}{\text{argmin}} \tag{4.3}$$

There are several solvers that can be used to solve an NLP problem in order to obtain the optimal decision variable $\boldsymbol{\omega}^\star$ but these are out of scope of this thesis. The developed NMPC local planner solves the NLP problems by a solution called interior point optimization (IPOPT), proposed by A. Wächter in [50]

4.3 Formulation of the NMPC Algorithm

Since the system model introduced in Chapter 3 is a non-linear model, this thesis focuses on designing an NMPC controller. NMPC is an advanced optimization-based method for feedback control of non-linear systems. Similarly to linear MPC formulation, it can be used in a variety of different applications. It is generally used to control complex or multiple input multiple output systems in which predicting the future behaviour of the system is essential. However, it is primarily used for stabilization and tracking problems which were introduced in Section 2.3.
The general idea of the NMPC scheme is the following: at each discrete time instant n we optimize the predicted future behaviour of the system for some finite horizon $k = 0, ..., N-1$ of length $N \geq 2$ and then use the first element of the resulting optimal control horizon as input to the system. The states $\boldsymbol{x}(n)$ are considered in a closed set $\boldsymbol{x}(n) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ with $0 \in \mathcal{X}$ and the control inputs are in a compact set $\boldsymbol{u}(n) \in \mathcal{U} \subset \mathbb{R}^{n_u}$ where $0 \in \mathcal{U}$. The NMPC utilizes the non-linear system model in order to predict and optimize the future system behaviour. The general non-linear control system model in discrete time form is:

$$\boldsymbol{x}^+ = f(\boldsymbol{x}, \boldsymbol{u}) \tag{4.4}$$

where the $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the transition map that assigns the state $\boldsymbol{x}^+ \in \mathcal{X}$ at the next sampling time to each pair of state $x \in \mathcal{X}$ and control value $u \in \mathcal{U}$.
In order to be able to react to the current deviation of $\boldsymbol{x}(n)$ from the reference $\boldsymbol{x}^{ref}$, the control input $\boldsymbol{u}(n)$ has to be defined in feedback form $\boldsymbol{u}(n) = \mu(\boldsymbol{x}(n))$ for some map μ,

mapping the state $\boldsymbol{x}(n) \in \mathcal{X}$ into the set $\mathcal{U}$ of control values.
In other words, the feedback law of the NMPC problem formulated as $\mu : \mathcal{X} \to \mathcal{U}$ with the assumption that the resulting closed-loop system satisfies:

$$\boldsymbol{x}^{+} = f(\boldsymbol{x}, \mu(\boldsymbol{x})) \tag{4.5}$$

where f is the same as in Equation 4.4. In the following Equation 4.5 will be called a nominal closed-loop system.
In the case of the NMPC, which is based on the system model presented in Chapter 3, the model includes the following states:

$$\mathcal{X} = [x \quad y \quad \theta]^T \qquad \mathcal{X} \in \mathbb{R}^3 \tag{4.6}$$

where x and y denote the position of the robot in the local coordinate frame while θ is the difference between the orientation of the map coordinate frame and the local coordinate frame. The control value space $\mathcal{U}$ is described by the following:

$$\mathcal{U} = [v \quad \omega]^T \qquad \mathcal{U} \in \mathbb{R}^2 \tag{4.7}$$

where v and ω are the linear and angular velocities, respectively.
In this thesis, two different problem formulations were considered. The first one is the set point stabilization NMPC, where the reference value is constant. Secondly, a trajectory tracking problem where the NMPC is formulated by a time-varying reference which is detailed further in Section 4.3.2.
In order to explain the concept of the NMPC design, firstly the problem will be considered as a set point stabilization problem.

4.3.1 Set Point Stabilization NMPC

In NMPC formulation we are talking about set point stabilization problems if the reference value is:

$$\boldsymbol{x}^{ref} \equiv \boldsymbol{x}_* \in \mathcal{X} \tag{4.8}$$

where $\boldsymbol{x}_*$ is constant.
The prerequisite for stabilizing the system at $\boldsymbol{x}_*$ by the feedback law is that the reference value $\boldsymbol{x}_*$ has to be the equilibrium of Equation 4.5, the nominal closed-loop system, while satisfying the required condition where such $u_* \in \mathcal{U}$ control value exists, where:

$$\boldsymbol{x}_* = f(\boldsymbol{x}_*, u_*) \tag{4.9}$$

Since NMPC is predictive control, it uses the non-linear system model in order to predict

and optimize future system behaviour. In order to differentiate the real state and control variables $\boldsymbol{x}(n)$, $\boldsymbol{u}(n)$ from the predicted variables, the latter are denoted as $\bar{\boldsymbol{x}}(n)$ and $\bar{\boldsymbol{u}}(n)$. Starting from the current state $\boldsymbol{x}(n)$ for the control sequence $\boldsymbol{u}(0), ..., \boldsymbol{u}(N-1)$ where $N \geq 2$ is the length of the prediction horizon, the prediction trajectory can be constructed as:

$$\bar{\boldsymbol{x}}(0) = \boldsymbol{x}(n), \qquad \bar{\boldsymbol{x}}(k+1) = f(\bar{\boldsymbol{x}}(k), \bar{\boldsymbol{u}}(k)), \qquad k = 0, ..., N-1 \tag{4.10}$$

by iterating with Equation 4.4. By this iterative method, the $\bar{\boldsymbol{x}}(k)$ state prediction is obtained for the state $\boldsymbol{x}(n+k)$ at time t_{n+k} in the future. Also, based on the chosen control sequence $\boldsymbol{u}(0), ..., \boldsymbol{u}(N-1)$ the prediction of the system behaviour is obtained on the discrete-time interval $t_n, ..., t_{n+N}$.
In order to decide the optimal control sequence $\boldsymbol{u}(0), ..., \boldsymbol{u}(N-1)$ that results in states $\bar{\boldsymbol{x}}(k)$ approaching the reference value (set point) $\boldsymbol{x}_*$ for $k = 0, ..., N-1$ a minimization of a cost function is used. This cost function penalizes the state error $\boldsymbol{x}^{err}$ between $\bar{\boldsymbol{x}}(n)$ and $\boldsymbol{x}_*$. Furthermore, it can penalize the deviation of the control values $\boldsymbol{u}(k)$ to the reference control u_*. A common choice for this cost function $\ell(\bar{\boldsymbol{x}}(k), \boldsymbol{u}(k))$ is a quadratic function that satisfies following condition:

$$\ell(\boldsymbol{x}_*, \boldsymbol{u}_*) = 0 \quad \text{and} \quad \ell(\bar{\boldsymbol{x}}, \boldsymbol{u}) > 0 \quad \text{for all } \bar{\boldsymbol{x}} \in \mathcal{X}, \boldsymbol{u} \in \mathcal{U} \text{ with } \boldsymbol{x} \neq \boldsymbol{x}_* \tag{4.11}$$

Such a quadratic function for set point stabilization NMPC local planner is formulated as,

$$\begin{aligned} \ell(\boldsymbol{x}(k), \boldsymbol{u}(k)) &= \|\boldsymbol{x}^{err}(k)\|^2_{\boldsymbol{Q}} + \|\boldsymbol{u}(k)\|^2_{\boldsymbol{R}} + \|\Delta\boldsymbol{u}(k)\|^2_{\boldsymbol{W}} \\ &= \boldsymbol{Q}^T \boldsymbol{x}^{err}(k) \boldsymbol{Q} + \boldsymbol{R}^T \boldsymbol{u}(k) \boldsymbol{R} + \boldsymbol{W}^T \Delta\boldsymbol{u}(k) \boldsymbol{W} \end{aligned} \tag{4.12}$$

where $\|\cdot\|$ denotes the Euclidean norm. The state error $\boldsymbol{x}^{err}(k)$ defined as,

$$\boldsymbol{x}^{err}(k) = \boldsymbol{x}(k) - \boldsymbol{x}^{ref} = [\varepsilon_x \quad \varepsilon_y \quad \varepsilon_\theta]^T \tag{4.13}$$

where ε_x, ε_y and ε_θ are the positions and heading errors. In Equation 4.12 $\Delta\boldsymbol{u}(k)$ is the change in control input and defined as the following:

$$\Delta\boldsymbol{u}(k) = \boldsymbol{u}(k) - \boldsymbol{u}(k+1) \tag{4.14}$$

In Equation 4.12, the weighting cost matrices $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{W} \geq 0$ are positive semi-definite in order to guarantee that the cost is always positive semi-definite. These cost matrices are considered as weights on the state error vector $\boldsymbol{x}^{err}$, control inputs u and the change of control inputs $\Delta\boldsymbol{u}$, respectively.
Given such a cost function ℓ respect to Equation 4.12 along the prediction horizon, $N \geq 2$

formulates the NMPC scheme algorithm. The optimal control problem (OCP) in this algorithm has to respect all admissible control sequences $\boldsymbol{u}(0), ..., \boldsymbol{u}(N-1)$ with $\bar{\boldsymbol{x}}$ generated by Equation 4.10. Here admissible means that there are a set of control sequences $\mathbb{U}^N(\boldsymbol{x}_0) \subseteq \mathcal{U}^N$ which includes constraints depending on the initial value $\boldsymbol{x}_0$.
In the case of the NMPC local planner, several constraints will be considered, which will be explained in Section 4.5. For now, simply assume an unconstrained scenario where $\mathbb{U}^N(\boldsymbol{x}_0) \equiv \mathcal{U}^N$ for all $\boldsymbol{x}_0 \in \mathcal{X}$.
Also, note that here direct single shooting is applied as optimal control method where the optimal state trajectory $\bar{\boldsymbol{x}}(\cdot)$ is not included in the optimization parameters of the OCP, leaving only $\bar{\boldsymbol{u}}(\cdot)$ to be the decision variable of the NLP. In this case, the set point stabilization NMPC scheme performs predictions at each sampling time $t_n, n = 0, 1, 2, ...$ via the solution of the optimal control problem.

Algorithm 4.1: set point stabilization NMPC scheme

1 Measuring the state $\boldsymbol{x}(n) \in \mathcal{X}$;
2 Set $\boldsymbol{x}_0 := \boldsymbol{x}(n)$, solve the optimal control problem

$$\begin{aligned} \underset{\bar{\boldsymbol{u}}(\cdot)\in\mathbb{U}^N(\boldsymbol{x}_0)}{\text{minimize}} \quad & J(\boldsymbol{x}_0, \bar{\boldsymbol{u}}(\cdot)) := \sum_{k=0}^{N-1} \ell(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) \\ \text{subject to,} \quad & \bar{\boldsymbol{x}}(0, \boldsymbol{x}_0) = \boldsymbol{x}_0, \quad \bar{\boldsymbol{x}}((k+1), \boldsymbol{x}_0) = f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) \end{aligned} \tag{4.15}$$

obtain the optimal control sequence by $\boldsymbol{u}^\star(\cdot) \in \mathbb{U}^n(\boldsymbol{x}_0)$;
3 Get NMPC-feedback value $\mu(\boldsymbol{x}(n)) := \boldsymbol{u}^\star(0) \in \mathcal{U}$ and use it in the next sampling period

In Algorithm 4.1, the optimal control sequence $\boldsymbol{u}^\star(\cdot)$ is calculated using the state information $\boldsymbol{x}(n)$ at sampling time t_n assuming that such an $\boldsymbol{u}^\star$ exist. At each sampling time t_n, the future behaviour of the system is predicted over the horizon of $[t_n, t_n + N]$ by computing the optimal input $\boldsymbol{u}^\star(\cdot)$.
The NMPC local planner is running on a fixed rate between each new time sample t_n which results in a set time span between separate path planning. During that time, the first optimal control input $\boldsymbol{u}^\star(0)$ is applied to the system. When a new iteration starts the prediction horizon shifts forward to $[t_{n+1}, t_{n+1} + N]$.
Note that in Algorithm 4.1, the constraints are not considered. The constraints will be detailed further in Section 4.5. The next Section will introduce NMPC local planning problem with the consideration of varying reference.

4.3.2 Trajectory Tracking NMPC

If the NMPC is formulated by considering a time varying reference, the problem is called a tracking problem. In an NMPC with a non-constant reference $\boldsymbol{x}^{ref}(n)$ it is assumed that

$\boldsymbol{x}_* \in \mathcal{X}$ is the equilibrium of the nominal closed-loop system for corresponding control value $u_* \in \mathcal{U}$. In this case, $\boldsymbol{x}^{ref}$ can then be defined as:

$$\boldsymbol{x}^{ref}(n) = \boldsymbol{x}_{\boldsymbol{u}^{ref}}(n, \boldsymbol{x}_0) \tag{4.16}$$

where $\boldsymbol{x}_0 = \boldsymbol{x}^{ref}(0)$ and some suitable reference infinite horizon control sequence $\boldsymbol{u}^{ref}(\cdot) \in \mathbb{U}^{\infty}(\boldsymbol{x}_0)$. When the reference is not constant, the $\boldsymbol{x}^{ref}(\cdot)$ and $\boldsymbol{u}^{ref}(\cdot)$ has to be taken into account when formulating the cost function ℓ. In such a time-varying case, the cost for the reference becomes:

$$\ell(n, \boldsymbol{x}^{ref}(n), \boldsymbol{u}^{ref}(n)) = 0 \quad \forall n \in \mathbb{N}_0 \tag{4.17}$$

and,

$$\ell(n, \boldsymbol{x}, \boldsymbol{u}) > 0 \quad \forall\, n \in \mathbb{N}_0,\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{u} \in \mathcal{U} \text{ with } \boldsymbol{x} \neq \boldsymbol{x}^{ref}(n) \tag{4.18}$$

where $\mathcal{X}$ and $\mathcal{U}$ are the set of states and control inputs, respectively. $\mathbb{N}_0$ is the set of natural numbers, including 0. Applying the same motive as in Section 4.3.1 the quadratic function for a trajectory tracking NMPC local planner is defined as:

$$\ell(n, \boldsymbol{x}, \boldsymbol{u}) = \|\boldsymbol{x}^{err}(n)\|_Q^2 + \|\boldsymbol{u}^{err}(n)\|_R^2 + \|\Delta\boldsymbol{u}(n)\|_W^2 \tag{4.19}$$

where $Q, R, W \geq 0$ are positive semi-definite weighting matrices. The state error $\boldsymbol{x}^{err}(n)$ is defined as:

$$\boldsymbol{x}^{err}(n) = \bar{\boldsymbol{x}} - \boldsymbol{x}^{ref}(n). \tag{4.20}$$

The error on the control input $\boldsymbol{u}^{err}(n)$ is:

$$\boldsymbol{u}^{err}(n) = \boldsymbol{u} - \boldsymbol{u}^{ref}(n) \tag{4.21}$$

and the change in the control input $\Delta\boldsymbol{u}(n)$ defined as:

$$\Delta\boldsymbol{u}(n) = \boldsymbol{u}(n) - \boldsymbol{u}(n+1) \tag{4.22}$$

For each $k = 0, ..., N-1$, the prediction $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ with $\boldsymbol{x}_0 = \boldsymbol{x}(n)$ used in the NMPC algorithm now becomes a prediction of the closed-loop state $\boldsymbol{x}(n+k)$. This prediction should be close to the desired reference state $\boldsymbol{x}^{ref}(n+k)$. Therefore the prediction $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ is used in the cost function defined in Equation 4.19. This leads to the following algorithm for minimizing the set of control sequences $\mathbb{U}^N(\boldsymbol{x}_0)$ in the horizon $k = 0, ..., N-1$.

Algorithm 4.2: trajectory tracking NMPC scheme

1 Measuring the state $\boldsymbol{x}(n) \in \mathcal{X}$;

2 Set $\boldsymbol{x}_0 := \boldsymbol{x}(n)$, solve the optimal control problem

$$\begin{aligned} \underset{\bar{\boldsymbol{u}}(\cdot)\in\mathbb{U}^N(\boldsymbol{x}_0)}{\text{minimize}} \quad & J(n, \boldsymbol{x}_0, \bar{\boldsymbol{u}}(\cdot)) := \sum_{k=0}^{N-1} \ell(n+k, \bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) \\ \text{subject to,} \quad & \bar{\boldsymbol{x}}(0, \boldsymbol{x}_0) = \boldsymbol{x}_0, \quad \bar{\boldsymbol{x}}((k+1), \boldsymbol{x}_0) = f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) \end{aligned} \tag{4.23}$$

obtain the optimal control sequence by $\boldsymbol{u}^\star(\cdot) \in \mathbb{U}^n(\boldsymbol{x}_0)$;

3 Get NMPC-feedback value $\mu(\boldsymbol{x}(n)) := \boldsymbol{u}^\star(0) \in \mathcal{U}$ and use it in the next sampling period

In Algorithm 4.2 the optimal control sequence $\boldsymbol{u}^\star(\cdot)$ is calculated using the state information $\boldsymbol{x}(n)$ at sampling time t_n assuming that such a $\boldsymbol{u}^\star$ exist. At each sampling time t_n, the future behaviour of the system is predicted over the horizon of $[t_n, t_n + N]$ by computing the optimal input $\boldsymbol{u}^\star(\cdot)$.
Similarly, as is the case for the set point NMPC scheme (Algorithm 4.1), direct single shooting is applied as optimal control method where the optimal state trajectory $\bar{\boldsymbol{x}}(\cdot)$ is not included in the optimization parameters of the OCP, leaving only the $\bar{\boldsymbol{u}}(\cdot)$ as the decision variable of the NPL. In addition, remark that the OCP in Algorithm 4.2 is reduced to Algorithm 4.1 if ℓ is not dependent on n. Moreover, Algorithm 4.2 is free from constraints; this is further explained in Section 4.5.

4.4 Discretization of NMPC

The previous sections (Section 4.3.1, 4.3.2) explained how the optimal solution $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ of a discrete-time system (Equation 4.4) from the OCP could be obtained using numerical methods for differential equations, but not how the minimization problem in the OCP can be solved.
In particular, how the OCP problem (Equation 4.1) can be reformulated into an NLP problem (Equation 4.2)
Even though OCP is already considered as a discrete-time problem, the process to turn it into an NLP problem is called discretization.
There are several methods for discretization, but for now, concentrate on direct methods which are by far the most popular class of algorithms in NMPC applications. In these methods, the OCP is transformed into a static optimization problem which can then be solved by NLP algorithms.

4.4.1 Direct optimal control methods

Popular direct approaches or direct optimal control methods can be distinguished into main families. The first one is the sequential approach, represented as direct single shooting method where the state trajectory is not included in the decision variables, leaving the control trajectory a set of free parameters to be determined by the NLP solver.
The second approach is the simultaneous approach where the state trajectory is estimated by polynomials whose coefficients are determined by the NLP solver. One of the first simultaneous approaches is the direct collocation approach which was presented by Lynn in [51].
A hybrid solution between the sequential and simultaneous approaches is the direct multiple shooting method, proposed by Bock and Plitt in ([48], [52]). This method has some important aspect of the simultaneous approach, which makes it able to handle unstable systems, while avoiding storing the whole state trajectory in the problem formulation. For the NMPC local planner, the direct multiple shooting has been considered and implemented.

4.4.2 Direct multiple shooting discretization

The previously used method in Algorithm 4.1 and 4.2 was called direct single shooting method where $\bar{\boldsymbol{u}}(\cdot)$ was the only optimization variable in the NLP. Generally, this means that the discretization occurs only at the first entry of the OCP when $\bar{\boldsymbol{x}}(0, \boldsymbol{x}_0) = \boldsymbol{x}_0$. This is called a shooting point. The main drawback of this method is the non-linear propagation, which means that the integrator function tends to become highly non-linear for long horizons N.
As it was mentioned before, direct multiple shooting is a hybrid solution between the sequential and simultaneous approaches. It is also called lifted single shooting. The principal idea of this method is to include some or all component states $\bar{\boldsymbol{x}}(\cdot)$ as independent optimization variables in the problem. This means that the optimization variables ω in the NLP problem are formulated as:

$$\boldsymbol{\omega} = \left[\bar{\boldsymbol{u}}(0)^T, ..., \bar{\boldsymbol{u}}(N-1)^T, \bar{\boldsymbol{x}}(0, \boldsymbol{x}_0)^T, ..., \bar{\boldsymbol{x}}(N, \boldsymbol{x}_0)^T\right] \tag{4.24}$$

where

$$\bar{\boldsymbol{x}}(k+1, \boldsymbol{x}_0) = f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) \tag{4.25}$$

with the initial condition $\bar{x}(0, \boldsymbol{x}_0) = \boldsymbol{x}_0$.
In order to ensure the optimal solution of the NLP, an extra equality constraint has to be applied at each optimization (shooting) step which satisfies the system dynamic.

$$\bar{\boldsymbol{x}}(k+1, \boldsymbol{x}_0) - f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) = 0, \quad k = 0, ..., N-1 \tag{4.26}$$

The direct multiple shooting is called lifted single shooting because by reformulating the system, by adding the predicted states $\bar{\boldsymbol{x}}$ as decision variables and considering an extra equality constraint (Equation 4.26) where the difference between the actual state $\bar{\boldsymbol{x}}(k+1, \boldsymbol{x}_0)$ and the predicted state $f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k))$ represents the system model and applying it at every prediction step $k = 0, ..., N-1$ or shooting step.
By increasing the number of decision variables the dimensionality of the NLP grows, since it increases the number of optimization variables in the problem, in exchange for a less non linear system model, which often results in a smaller NLP problem that can be solved faster. Also by adding $\bar{\boldsymbol{x}}$ as decision variable it gives an opportunity to initialize by state trajectory.

4.5 Constraints

One of the significant successes of MPC is the ability to take constraints into account explicitly. In this Section, the different considerations of constraints are explained. Firstly, the constraints on the states which are considered to be bounded by the local costmap. Secondly, the constraints on the control inputs that are limited due to the robot technical properties but also has a great influence on the robot motion and performance as a tuning parameter. In Section 4.5.2 the constraints for collision avoidance with static obstacles are detailed. Finally, Section 4.5.3, shows how an NMPC local planner can use the predictions from moving obstacles and apply different constraint at each prediction in order to consider only the admissible trajectories.

4.5.1 Constraints on States and Inputs

It is a common practice to define the constraints both on the states and on the control inputs. To do so, the admissible states are understood as $\boldsymbol{x} \in \mathbb{X}$, where the set of admissible states are $\mathbb{X} \subseteq \mathcal{X}$. Furthermore, $u \in \mathbb{U}(\boldsymbol{x})$ is considered as the admissible control values for $\boldsymbol{x}$, where the set of admissible control values is $\mathbb{U}(\boldsymbol{x}) \subseteq \mathcal{U}$. By introducing these sets and using a control system like in Equation 4.4 it is possible to calculate with the OCP problem where the states and control inputs are constrained by $\mathbb{X}$ and $\mathbb{U}(\boldsymbol{x})$, respectively. Such a control system should have the following control sequence and the corresponding trajectory:

$$\boldsymbol{u}(k) \in \mathbb{U}(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)) \quad \text{and} \quad \bar{\boldsymbol{x}}(k+1, \boldsymbol{x}_0) \in \mathbb{X} \quad \text{for all} \quad k = 0, ..., N-1 \tag{4.27}$$

where $N \in \mathbb{N}$ and $\boldsymbol{x}_0 \in \mathbb{X}$ is an initial value. Here $\boldsymbol{u} \in \mathcal{U}^N$ control sequence with the trajectory $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ is admissible for $\boldsymbol{x}_0$ up to time N if its hold for $k = 0, ..., N-1$. Also, an assumption should be considered where for each $\boldsymbol{x} \in \mathbb{X}$ exist a $\boldsymbol{u} \in \mathbb{U}(\boldsymbol{x})$ where $f(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{X}$ holds. This is an essential assumption in order to ensure feasibility. The

OCP is called feasible for $\boldsymbol{x}_0$ if the set of admissible control sequences $\mathbb{U}^N(\boldsymbol{x}_0)$ for $\boldsymbol{x}_0$ up to N over which the OCP is optimized is nonempty.
In the case of NMPC handling, these constraints are straightforward because these can directly be inserted into Algorithm 4.1 or Algorithm 4.2.

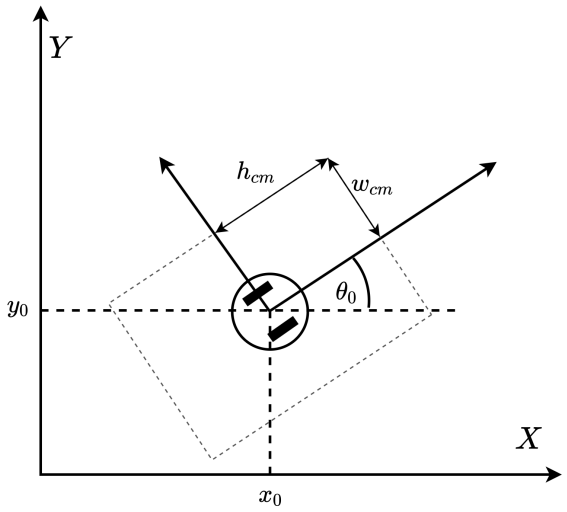


Figure 4.1: Dimensions of the costmap of the robot's current state

In the case of the NMPC local planner, the set of admissible states $\mathbb{X}$ can be considered as unconstrained if there are no margins on the map ($(x,\, y)$ can reach any value) and the robot can turn towards any direction (θ). Another concept is to limit the admissible positions $(x,\, y)$ to the size of the local costmap. This method neglects all the spaces that the sensor measurements do not consider. In this case, the local planner will not consider solutions where the robot ends up in an undetected area. Most of the local planners like DWA or TEB do not consider those areas either. The parameters for the height and width of the costmap are denoted as h_{cm}, w_{cm}, and they are measured from the current position of the robot $(x_0,\, y_0)$. For better understanding, Figure 4.1 shows the boundary of the costmap around the robot's current state $\boldsymbol{x}_0$.
To define the set of admissible states for this application the upper and lower bounds on the state variable x are defined as:

$$\begin{aligned} x_{max} &= x_0 + h_{cm}\, cos(\theta_0) + w_{cm}\, sin(\theta_0) \\ x_{min} &= x_0 - h_{cm}\, cos(\theta_0) - w_{cm}\, sin(\theta_0) \end{aligned} \tag{4.28}$$

where the x_{max}, x_{min} are the upper and lower bound of the state variable x respectively. Similarly the upper and lower bounds $(y_{max},\, y_{min})$ on the state variable y are calculated as:

$$\begin{aligned} y_{max} &= y_0 + h_{cm}\, sin(\theta_0) + w_{cm}\, cos(\theta_0) \\ y_{min} &= y_0 - h_{cm}\, sin(\theta_0) - w_{cm}\, cos(\theta_0) \end{aligned} \tag{4.29}$$

In this concept, the θ is still unconstrained. The set of admissible states in the NMPC local planning problem are defined as the following:

$$\boldsymbol{x} \in \mathbb{X} \quad \text{where} \quad \mathbb{X} := \begin{bmatrix} x_{min} \\ y_{min} \\ -\infty \end{bmatrix} \leq \mathcal{X} \leq \begin{bmatrix} x_{max} \\ y_{max} \\ \infty \end{bmatrix} \tag{4.30}$$

The control values are constrained by the maximum and minimum velocity commands that should be applied to the robot. In this case, the set of admissible control values $\mathbb{U}^N(\boldsymbol{x})$ are the following:

$$\boldsymbol{u} \in \mathbb{U}^N(\boldsymbol{x}) \quad \text{where} \quad \mathbb{U} := \begin{bmatrix} v_{min} \\ \omega_{min} \end{bmatrix} \leq \mathcal{U} \leq \begin{bmatrix} v_{max} \\ \omega_{max} \end{bmatrix} \tag{4.31}$$

Where the v_{max} and v_{min} are the maximum and minimum linear velocities control inputs while ω_{max} and ω_{min} are the maximum and minimum angular velocity control inputs, respectively. Note that the minimum angular velocity control input should be $\omega_{min} = -\omega_{max}$ since the robot should be able to turn to both directions equally. Furthermore, if $v_{min} = 0$ then the NMPC local planner neglects all solutions, where the robot moves backwards. This can be useful since the TIAGo Base and Steel do not proper rear facing sensing capabilities for sensing the area behind the robot, while they are differential driven robots which makes them capable of turning back by rotating in place ($v = 0, \omega \neq 0$). Moreover, since the objective function (Equation 4.19) does not penalize backwards motion $v_{min} < 0$, once the robot started to move backwards it usually continues its motion this way for extended periods, due to costs on turning, which is not preferable due to the sensor arrangement problem.

4.5.2 Static Obstacle Constraints

An essential property of a local planner is that it calculates velocity control inputs that respect obstacles and avoids collision between them and the robot. The global planner calculates an optimal path based on a previously mapped area that the local planner should follow. However, sometimes the environment changes which results in the appearance of new obstacles. In addition, if the reference that the local planner follows is only one point of the globally optimal path, an obstacle-free path is not guaranteed between the robot and the reference point. It is expected of the local planner to overcome these challenges, circumnavigate the obstacles and avoid a collision.

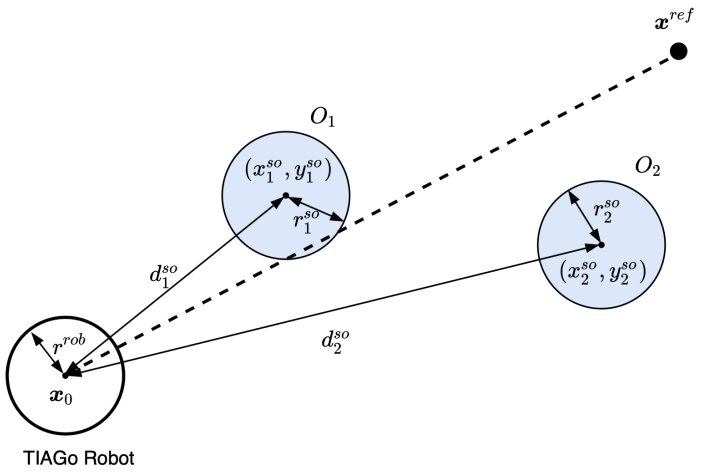


Figure 4.2: Scenario with two static obstacles (O_1 and O_2)

NMPC is suited to handle obstacles as constraints and directly insert them into the OCP. In this report two different static obstacle representation were considered: the polygon centroid representation (Section 3.5.1) and the closes point of polygon representation (Section 3.5.2). In both cases, the final goal is the same: keeping the current and predicted states far enough from specific points of the polygon-shaped static obstacles by neglecting the unwanted states from the set of admissible states $\mathbb{X}$.
For better understanding, a scenario was created, which illustrates a simplified problem in Figure 4.2. In this scenario, two obstacles are considered, O_1 in the position (x_1^{so}, y_1^{so}) and O_2 in the position (x_2^{so}, y_2^{so}) as point obstacles with the radius r_1^{so} and r_2^{so} respectively. These obstacles are placed in the area between the current state $\boldsymbol{x}_0 = [x_0, y_0, \theta_0]^T$ of the robot and the reference state $\boldsymbol{x}^{ref}$.
The set point stabilization NMPC local planner should predict a collision-free trajectory $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ from $\boldsymbol{x}_0$ up to the horizon N if it holds for $k = 0, ..., N-1$ by minimizing the state error $\boldsymbol{x}^{err}$ in the cost function from Equation 4.12. In order to minimize the distance between $\boldsymbol{x}_0$ and $\boldsymbol{x}^{ref}$ while respecting the obstacles, further constraints have to be defined, ensuring that the robot cannot move into areas occupied by static obstacles.
The distances d_1^{so} and d_2^{so} to the obstacles play an essential role in collision avoidance. These distances can be calculated by the Euclidean distance between the trajectory $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0) = \left[\bar{x}_k, \bar{y}_k, \bar{\theta}_k\right]^T$ and the obstacle positions as follows:

$$d_{k,i}^{so} = \sqrt{(\bar{x}_k - x_i^{so})^2 + (\bar{y}_k - y_i^{so})^2} \tag{4.32}$$

where the notation $\cdot_{k,i}$ denotes the k'th prediction of the NMPC scheme and the i'th obstacle consideration. These distances should not be smaller than the sum of the robot

radius and the radius of the obstacles.

$$d_{k,i}^{so} \geq r^{rob} + r_i^{so} \tag{4.33}$$

By combining Equation 4.32 and Equation 4.33, the following inequality constraints can be defined:

$$G^{so}(k,i) := -\sqrt{(\bar{x}_k - x_i^{so})^2 + (\bar{y}_k - y_i^{so})^2} + r^{rob} + r_i^{so} \geq 0 \tag{4.34}$$

for all $k = 0, ..., N-1$ and $i = 0, ..., n_{so}$.
As mentioned before, the NMPC local planner implementation considers two different static obstacle representation. The first one is the polygon centroid representation, where the centroid position of the i'th polygon obstacle (x_i^{cent}, y_i^{cent}) is calculated by Equation 3.16 and 3.17. In addition, the minimum radius r_i^{so} that encircles all the vertices of the i'th polygon from the centroid is defined (Equation 3.18). This representation is equivalent to the scenario from Figure 4.2.
The second representation is the closest point to the polygon method, which differs from the previous example. In this method, the closest point on the polygon edge has to be defined in relation to the robot position. There are several difficulties with this concept. First of all, the closest point on the polygon is a varying parameter between each prediction, since by predicting a trajectory $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0)$ at each $k = 0, ..., N-1$ the closest point of the i'th polygon ($x_{k,i}^{so}, y_{k,i}^{so}$) might be in a different position at each prediction. Another difficulty is that the closest point between a point and a polygon calculated by the Algorithm 3.2 presented in Section 3.5.2, can be defined only by an algorithm which includes conditional statements since it is dependent on the number of vertices of the polygon. This is why every polygon is reduced into a three-sided polygon by the ClosestNTriangle algorithm presented in Section 3.5.2.
By the mentioned assumption, the closest point of the polygon ($x_{k,i}^{so}, y_{k,i}^{so}$) at each prediction k can be calculated by the ClosestPoint2Polygon (Algorithm 3.2) to the predicted trajectory by $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0) = \left[\bar{x}_k, \bar{y}_k, \bar{\theta}_k\right]$ and the distance between them can be defined as:

$$d_{k,i}^{so} = \sqrt{(\bar{x}_k - x_{k,i}^{so})^2 + (\bar{y}_k - y_{k,i}^{so})^2} \tag{4.35}$$

In this method the radius of the polygon is $r_i^{so} = 0$, so a consideration of a safety boundary S_b was necessary to keep some distance between the robot and the closest point. In this case the constraints can be defined as follows:

$$G^{so}(k,i) := \sqrt{(\bar{x}_k - x_{k,i}^{so})^2 + (\bar{y}_k - y_{k,i}^{so})^2} + r^{rob} + S_b \geq 0 \tag{4.36}$$

4.5.3 Moving Obstacle Constraints

Moving obstacles are handled by the NMPC local planner similarly to the static obstacles. For a better understanding, consider the following scenario in Figure 4.3.

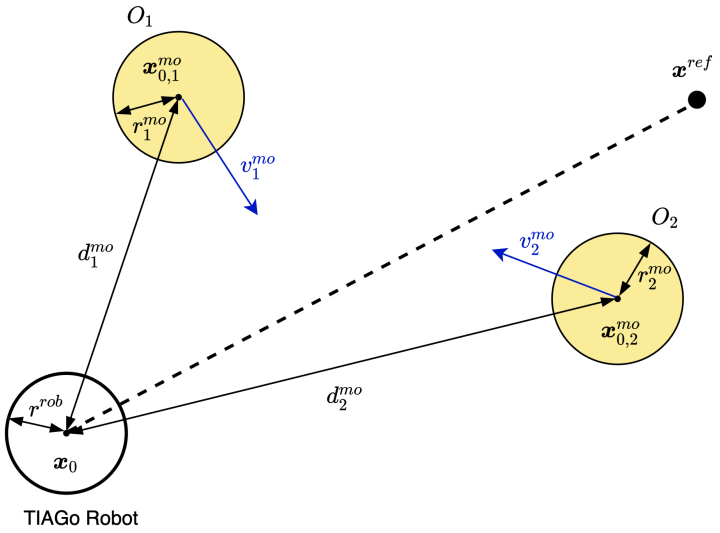


Figure 4.3: Scenario with two moving obstacles (O_1 and O_2)

In Figure 4.2, two obstacles are considered; O_1 and O_2 in the measured state $\boldsymbol{x}_{0,1}^{mo} = \left[x_{0,1}^{mo}, y_{0,1}^{mo}\right]^T$ and $\boldsymbol{x}_{0,2}^{mo} = \left[x_{0,2}^{mo}, y_{0,2}^{mo}\right]^T$ respectively. Each obstacle has a constant radius r_1^{so}, r_2^{so} and moving with a constant speed v_1^{mo}, v_2^{mo} and orientation θ_1^{mo}, θ_2^{mo}. These obstacles are placed in the area between the current state $\boldsymbol{x}_0 = [x_0, y_0, \theta_0]^T$ of the robot and the reference state $\boldsymbol{x}^{ref}$.
At every sampling time t_n, the initial states of the moving obstacles prediction are updating in regards to their measured state $\bar{\boldsymbol{x}}_{0,i}^{mo} = \boldsymbol{x}_{0,i}^{mo}$. After that, by applying a forward simulation, the future position of the moving obstacles are predicted:

$$\bar{\boldsymbol{x}}_{k,i}^{mo} = \begin{bmatrix} \bar{x}_{k,i}^{mo} \\ \bar{y}_{k,i}^{mo} \end{bmatrix} = \begin{bmatrix} \bar{x}_{0,i}^{mo} \\ \bar{y}_{0,i}^{mo} \end{bmatrix} + T_s \begin{bmatrix} cos(\theta_i^{mo}) \\ sin(\theta_i^{mo}) \end{bmatrix} \tag{4.37}$$

for all $k = 0, ..., N-1$ and $i = 1, ..., n_{mo}$ where n_{mo} is the number of considered moving obstacles. It is essential that T_s step size is the same as the one in the discretization of the system model in Equation 3.10, since this guarantees that at each prediction instant k the predicted elapsed time between each iteration is equivalent. The minimum distance between the trajectory $\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0))$ and the predicted trajectories of moving obstacles $\bar{\boldsymbol{x}}_{k,i}^{mo}$ has to comply with the following inequality:

$$d_{k,i}^{mo} \geq r^{rob} + r_i^{mo} + S_b \tag{4.38}$$

where the r^{rob} is the robot radius, and Smo_b is the safety boundary for moving obstacles. Each distance can be calculated as the Euclidean distance between the two trajectories:

$$d_{k,i}^{mo} = \sqrt{(\bar{x}_k - \bar{x}_{k,i}^{mo})^2 + (\bar{y}_k - \bar{y}_{k,i}^{mo})^2} \tag{4.39}$$

The constraints for the moving obstacles in the OCP are formulated as following:

$$G^{mo}(k,i) := -\sqrt{(\bar{x}_k - \bar{x}_{k,i}^{mo})^2 + (\bar{y}_k - \bar{y}_{k,i}^{mo})^2} + r^{rob} + r_i^{mo} + S_b \geq 0 \tag{4.40}$$

for all $k = 0, ..., N-1$ and $i = 0, ..., n_{mo}$. These constraints can be directly added to the NMPC algorithms in Algorithm 4.1 or Algorithm 4.2.

4.6 Summary of NMPC local planner

The following algorithm summarizes NMPC path planning algorithm:

Algorithm 4.3: NMPC path planning algorithm

Input : $\boldsymbol{x}(n), \boldsymbol{x}^{mo}(n), \boldsymbol{x}^{so}(n), \boldsymbol{x}^{ref}$

Output : $\boldsymbol{u}(n)$

Parameter: $N, T_s, h_{cm}, w_{cm}, v_{max}, v_{min}, \omega_{min}, \omega_{max}, n_{so}, n_{mo}, S_b, d_{gt}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{W}$

while *norm(*$\boldsymbol{x}_0 - \boldsymbol{x}^{ref}$*)* $> d_{gt}$ **do**

$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}(n)$

$\bar{\boldsymbol{x}}^{mo}_{i=0,\dots,n_{mo}} \leftarrow$ moving obstacle forward simulation from $\boldsymbol{x}^{mo}(n)$ (Equation 4.37)

if *Centroid Polygon Method* **then**

$x^{cent}, y^{cent}, r^{so} \leftarrow$ Calculate centroids $\forall \boldsymbol{x}^{so}(n)$ (Equation 3.16, 3.17, 3.18)

$\boldsymbol{x}^{so}_{i=0,\dots,n_{so}}, r^{so}_{i=0,\dots,n_{so}} \leftarrow$ Closest n_{so} to $\boldsymbol{x}_0$

else

$\boldsymbol{x}^{so}_{i=0,\dots,n_{so}}, S^{so}_{i=0,\dots,n_{so}} \leftarrow$ ClosestNTriangle$(\boldsymbol{x}_0, \boldsymbol{x}^{so}(n), n_{so})$

end

Solve OCP:

$$\underset{\substack{\bar{\boldsymbol{u}}(\cdot)\in\mathbb{U}^N(\boldsymbol{x}_0)\\ \bar{\boldsymbol{x}}(\cdot)\in\mathbb{X}}}{\text{minimize}} \quad J(\boldsymbol{x}_0, \bar{\boldsymbol{u}}(\cdot)) := \sum_{k=0}^{N-1} \ell(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k))$$

subject to

$$\bar{\boldsymbol{x}}(0, \boldsymbol{x}_0) = \boldsymbol{x_0}$$

$$\bar{\boldsymbol{x}}(k+1, \boldsymbol{x}_0) - f(\bar{\boldsymbol{x}}(k, \boldsymbol{x}_0), \bar{\boldsymbol{u}}(k)) = 0$$

$$\bar{\boldsymbol{u}}(k) \in \mathbb{U}^N(\boldsymbol{x}_0) \text{ (Equation 4.30)} \qquad \forall k \in [0, N-1]$$

$$\bar{\boldsymbol{x}}(k) \in \mathbb{X} \text{ (Equation 4.31)} \qquad \forall k \in [0, N]$$

$$G^{mo}(k, i) \geq 0 \text{ (Equation 4.40)} \qquad \forall k \in [0, N] \quad \forall i \in [0, n_{mo}]$$

$$G^{so}(k, i) \geq 0 \text{ (Equation 4.34, 4.36)} \qquad \forall k \in [0, N] \quad \forall i \in [0, n_{so}]$$

obtain the optimal control sequence by $\boldsymbol{u}^\star(\cdot)$

$\boldsymbol{u}^\star(n) \leftarrow \boldsymbol{u}^\star(0)$

end

CHAPTER 5

SIMULATION

This chapter evaluates the different NMPC local planner designs from Chapter 4 based upon the difficulty of implementation and general performance in a simulated environment. Firstly, in Section 5.1 the different methods are implemented in MATLAB, then the simulated controller designs are evaluated. The ones that perform the best will then be implemented in Python and ROS and simulated using Gazebo in Section 5.2.

5.1 MATLAB Simulation

The NMPC local planner has first been implemented and developed in the MATLAB programming platform. Many iterations of the NMPC have been developed over the course of the semester. As extensions of the original MPC concept or as new concepts that would perform better in certain scenarios, for example, set point stabilization or trajectory tracking. In addition, MATLAB has been used as a test-bench to test different scenarios and for adding constraints and costs to the objective function. This section will result in two advanced planning algorithms that can take both moving and static obstacles into account.

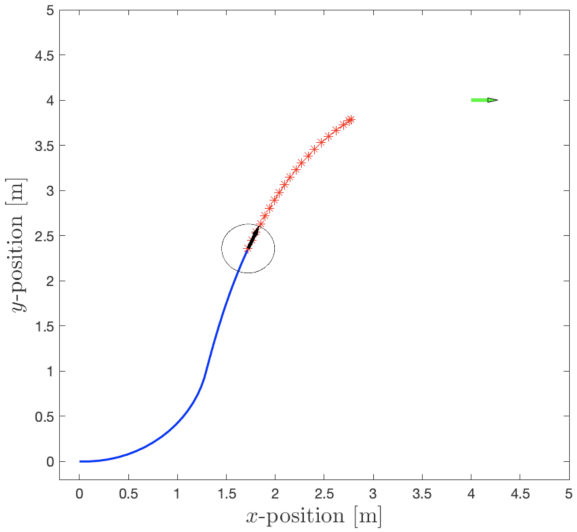
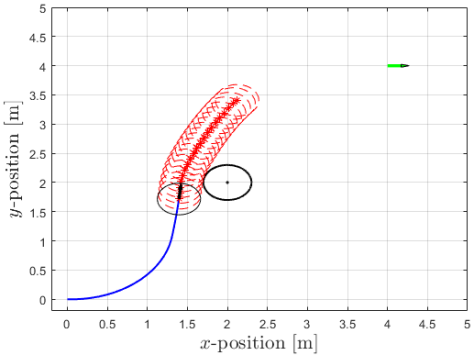
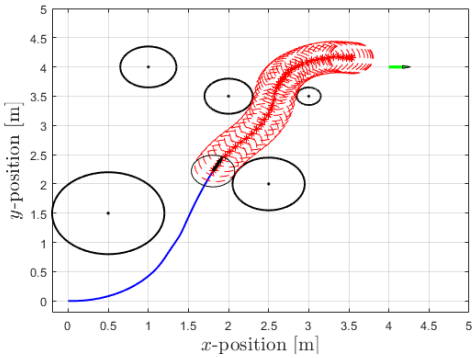


Figure 5.1: The robot drew as a circle with a heading moving towards the goal with the prediction horizon in red. The already travelled path is marked in blue.

The first NMPC developed is equivalent with the simplified example presented in Section 4.3.1 which only considers that the robot state has to reach a specific target state or a goal; this is also known as set-point stabilization. A simulation of this iteration of the NMPC can be seen in Figure 5.1.
The first improvement upon this simple set-point stabilization (Algorithm 4.1 was 'lifting'. Instead of using the direct single shooting discretization, a more advance method, called direct multiple shooting, also known as 'lifted' single shoot, was implemented for more information about the two methods, see Section 4.4.2. This method makes the NLP problem formulation computationally more expensive but decreases the non-linearity of the system model, which makes the NLP problem more sparse.
The next added functionality was static obstacles defined in the constraints of the NMPC. This was done with one obstacle, as seen in Figure 5.2a and later with multiple static obstacles, as seen in Figure 5.2b.



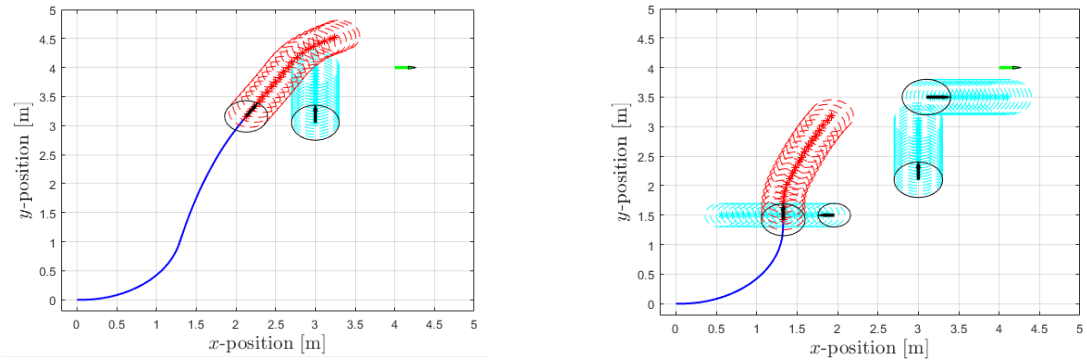
(a) One static obstacle obstructing the way of the robot.



(b) Multiple static obstacles obstructing the way of the robot.

Figure 5.2: NMPC path planning with static obstacles consideration as constraints, simulated in MATLAB. The blue line is the driven trajectory, the red circles are the predicted states with the robot footprint and the black circles with points in the middle are the static obstacles

The next objective was to implement moving obstacles into the NMPC. Unlike the static obstacles, these are the time-varying constraints whose predicted movement has to not coincide with the predicted movement of the robot. Much like the implementation of the static obstacles, firstly the constraints for one moving obstacle was implemented and then for multiple obstacles once the correct formulation had been derived. The robot together with the moving obstacles, can be seen in Figure 5.3.



(a) One moving obstacle obstructing the way of the robot.

(b) Multiple moving obstacles obstructing the way of the robot.

Figure 5.3: NMPC in MATLAB with moving obstacles as time-varying constraints,

In Figure 5.3, the predictions of the movement of the obstacles are illustrated with a cyan horizon and a red horizon for the robot. The constraints for the static obstacles and the time-varying constraints of the moving obstacles have to be combined and formulated in one NMPC that can consider both types of obstacles. This can be seen in Figure 5.4.

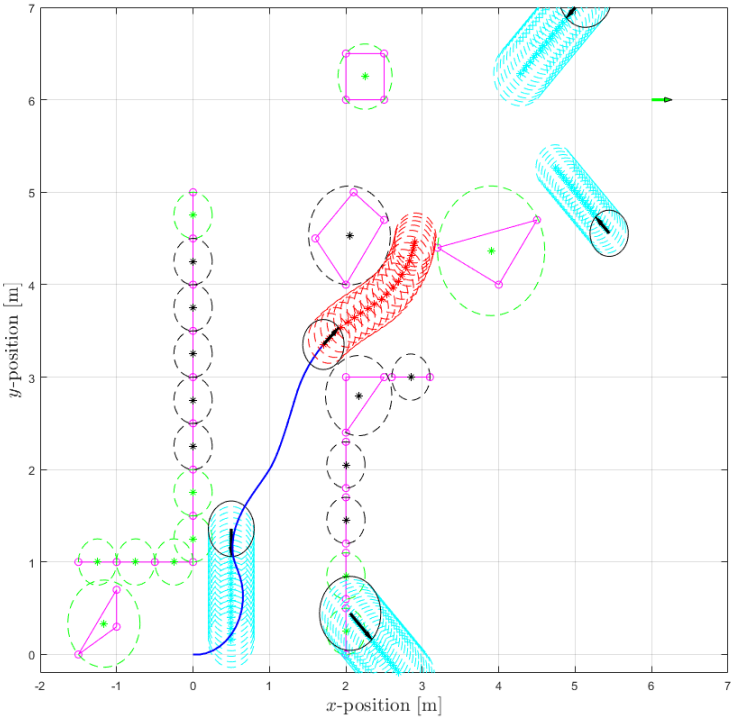


Figure 5.4: Set-point stabilization NMPC with static and moving obstacles. Static obstacles are represented as circles.

In Figure 5.4, the ten nearest static obstacles are considered in the NMPC while all the

moving obstacles are considered. The static obstacles currently considered by the NMPC are marked with black circles, and the rest of the static obstacles are marked with green. As mentioned in prior figures, the prediction horizon is marked in red, and the predictions of the moving obstacles are marked in cyan. In this scenario, the original obstacles that are polygons are represented as circular obstacles. For more information about this obstacle representation see Section 3.5.1. A simulation of this NMPC with a different static obstacle representation has also been developed. Another obstacle representation, called closest point to polygon method (Section 3.5.2, simplifies the complex polygons into three types/cases of obstacles, namely single-points, lines or three-sided polygons (triangles). In this method the NMPC use a complicated algorithm (Algorithm 3.2) in order to only consider the closest point to the simplified obstacles at each prediction. This simulation can be seen in Figure 5.5.

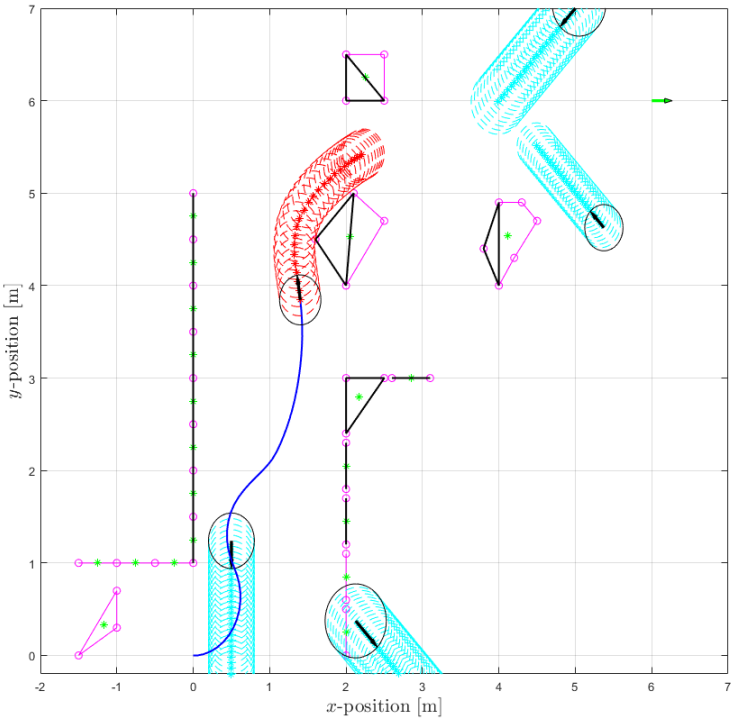
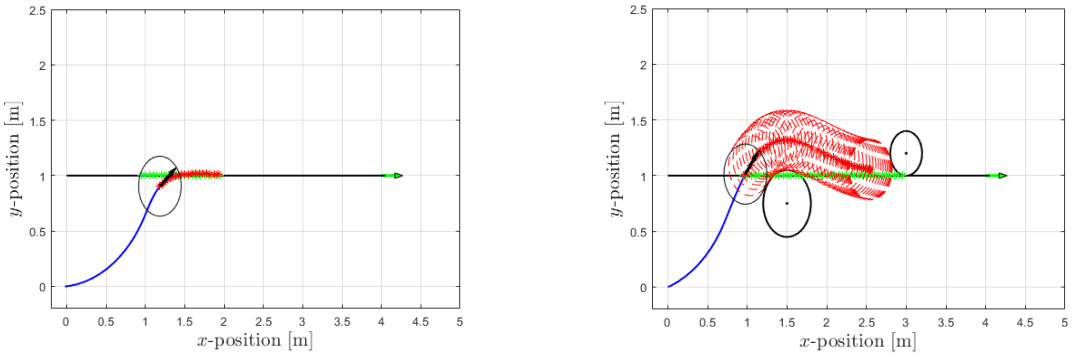


Figure 5.5: Set point stabilization NMPC with static and moving obstacles using the closest point to polygon static obstacle representation

The goal is kept constant throughout the simulations when using set-point stabilization. However, a time-varying reference can be useful when trying to, for example, follow a specific trajectory rather than merely finding a path to a given goal. To investigate how a trajectory tracking NMPC from Section 4.3.2 can be used for local planning, it was

implemented and investigated in MATLAB. The reference trajectory consists of a set of poses with linear and angular velocities; this set is the same length as the prediction horizon. The reference trajectory moves along the global plan in time. To implement this trajectory tracking, a simple line was first used as the global plan that the robot has to follow. This can be seen in Figure 5.6a.



(a) The robot has to follow a trajectory along a straight line.

(b) Obstacles are now introduced in the constraints.

Figure 5.6: NMPC with trajectory tracking along a straight line.

In Figure 5.6 the reference trajectory can be seen in green, and the prediction horizon in red. Once the reference trajectory reaches the goal, the local planner starts using set-point stabilization to converge to the goal. Moving obstacles can also be introduced to the environment, resulting in Figure 5.7.

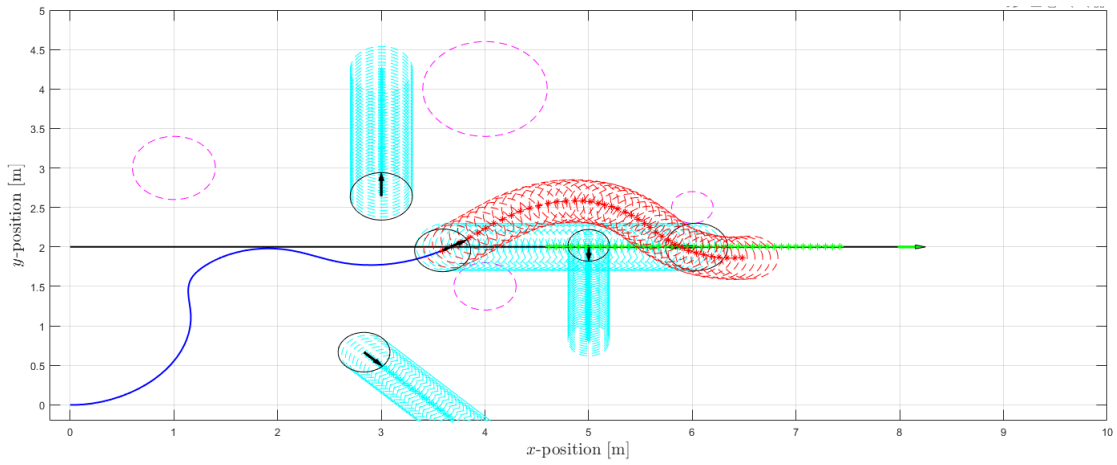


Figure 5.7: Trajectory tracking NMPC with moving obstacles(cyan) and static obstacles(magenta).

More complex global paths can also be followed using trajectory tracking. In fact, any path can be followed using trajectory tracking, given that a set of reference states and

control inputs are given. The NMPC local planner following a circular trajectory can be seen in Figure 5.8.

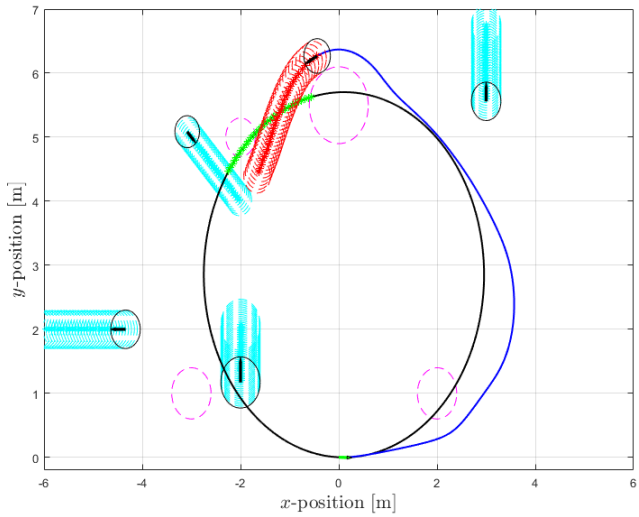
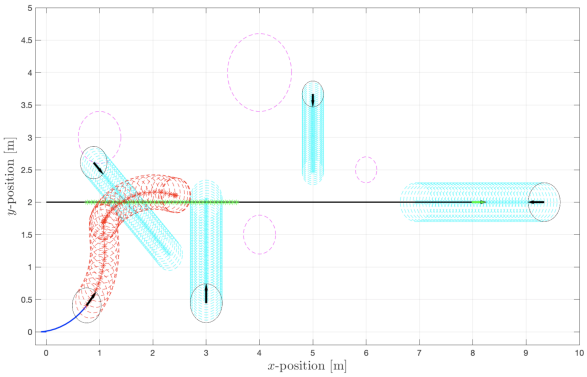
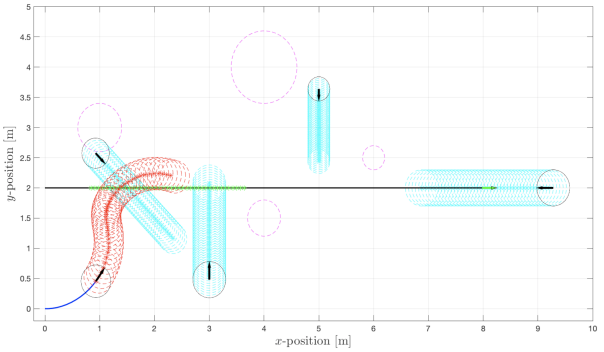


Figure 5.8: The NMPC following a circular trajectory while avoiding moving and static obstacles. The reference trajectory is marked with green on the circle

Lastly, the behaviour of the NMPC in both trajectory tracking and set-point stabilization has been quite erratic. This is due to the fact that there is no constraint or cost on the acceleration of the robot. This means that the robot will move as fast as possible while there are no obstacles in the way. This resulted in sudden stops when the robot was near an obstacle and very high acceleration when the robot starts to move again.



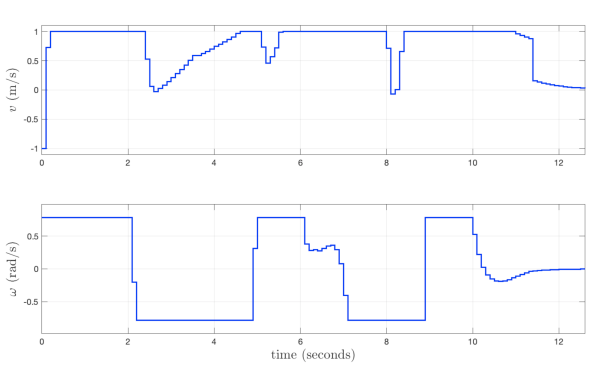
(a) The local planner trajectory tracking without a cost on acceleration. It can be seen that the robot does, indeed stop in the prediction waiting for the obstacle to pass by.



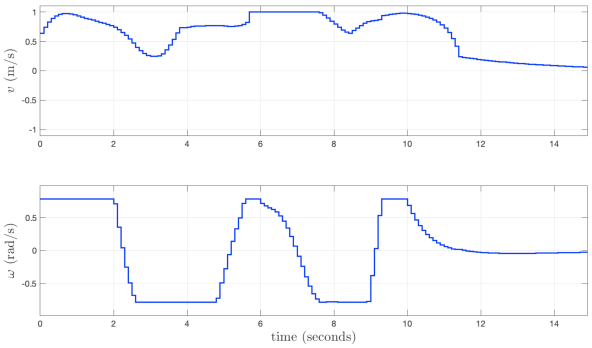
(b) The local planner trajectory tracking with a cost on acceleration. Instead of stopping at the obstacle, the robot slows down and speeds up again once the obstacle has is cleared from the predicted trajectory.

Figure 5.9: NMPC with trajectory tracking along a straight line before and after introducing a cost on the acceleration.

An example of this can be seen in the trajectory tracking scenario seen in Figure 5.9a. Therefore a cost has been put on the acceleration such that the robot accelerates and decelerates slower, resulting in smoother applied control inputs. The same scenario with cost on acceleration can be seen in Figure 5.9b.
The control outputs from the local planner have also been smoothened by introducing the cost on the acceleration. The control output has been recorded from start to goal in the scenario presented in Figure 5.9 and can be seen in Figure 5.10.



(a) The control output of the local planner without cost on the acceleration. The sudden spikes in linear and angular velocity are apparent.



(b) The control output of the local planner with cost on the acceleration. The sudden spikes in linear and angular velocity are smoothened such that the robot does not stop while moving towards the goal.

Figure 5.10: NMPC with trajectory tracking along a straight line before and after introducing a cost on the acceleration.

While the local planner without acceleration cost reaches the goal faster than the local planner with cost on the acceleration, however, it makes two stops while moving towards that goal. This more aggressive behaviour in relation to movement is less desired for a robot that has to be integrated into an environment with people as it might be uncomfortable to interact with. This is the final development of the NMPC local planner in MATLAB. The development has resulted in an NMPC local planner that uses set-point stabilization to converge to the goal and an NMPC local planner that uses a mixture of trajectory tracking and set-point stabilization to converge to the goal. The NMPC local planner with set point stabilization is chosen for continued work due to the fact that the local planner using trajectory tracking needs a reference trajectory not only containing position and heading, but also the linear and angular velocities. The global planners used on the TIAGo robot use the A* search algorithm that gives a set of poses to the final goal, but no information about intermediate velocities with these poses. Hereby providing a set of poses that can be treated as intermediate final goals for set-point stabilization, while having no velocities inhibits these poses being used for trajectory tracking. Hence trajec-

tory tracking NMPC is not implemented to any of the global planners that are installed on the TIAGo robot.

5.2 Gazebo Simulation

PAL robotics has made a quite comprehensive simulation environment for testing new packages for the TIAGo robot. The simulation environment consists of a complete robot model, sensor models and different Gazebo worlds for testing scenarios. The local planners chosen from the MATLAB simulation have been implemented in Gazebo before being implemented on the actual robot. The Gazebo simulations run all the same topics as the real robot and make use of the same localization and planning methods such as `amcl` and `Globalplanner`. This makes implementation easier due to only minor differences between the Gazebo simulation and the actual robot. Most of the differences were discovered in the change from MATLAB to Gazebo and ROS. One of the major differences was keeping track of the frames in which the data come in and the transformations needed to represent the data correctly. This problem arose, for example, in the shape of having to transform obstacles detected by the Lidar from `odom` frame to `map` frame, such that the obstacle data is static in relation to robot movement.

5.2.1 Simulation Scenarios

The two simulation scenarios are described in this section. These scenarios are done in the simulation to test the performance of the two local planners. First, the planners were tested in a static environment such that the differences in obstacle representation can be assessed. Then a test with multiple moving obstacles is done to assesses the capabilities of the local planner when in a highly dynamic environment. This test is only done once since both local planners use the same formulation for moving obstacles.

5.2.1.1 Environment with only static obstacles

In this simulation test, the robot has to navigate through a room with static obstacles consisting of shelves, tables and walls. Both the local planner will have the same starting position and goal. The test can be seen in Figure 5.11.

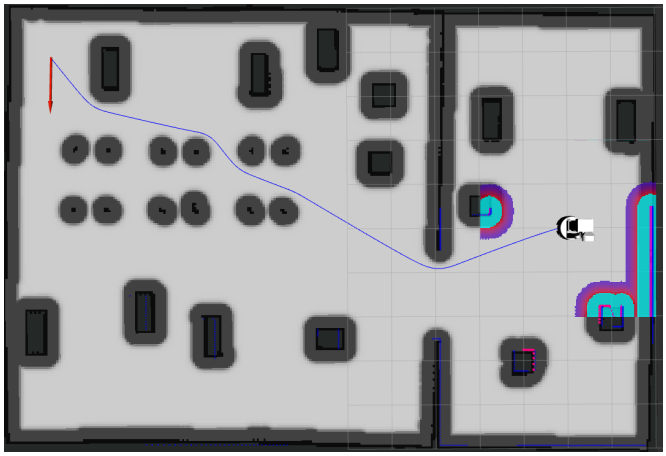
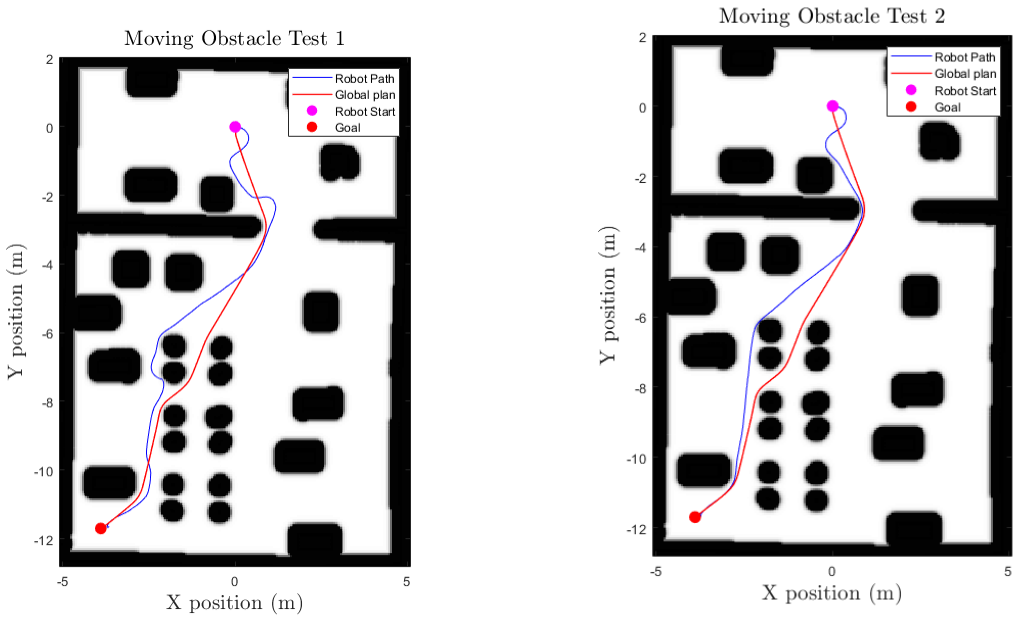


Figure 5.11: The map with the initial position and the goal marked as a red arrow indicating the final orientation.

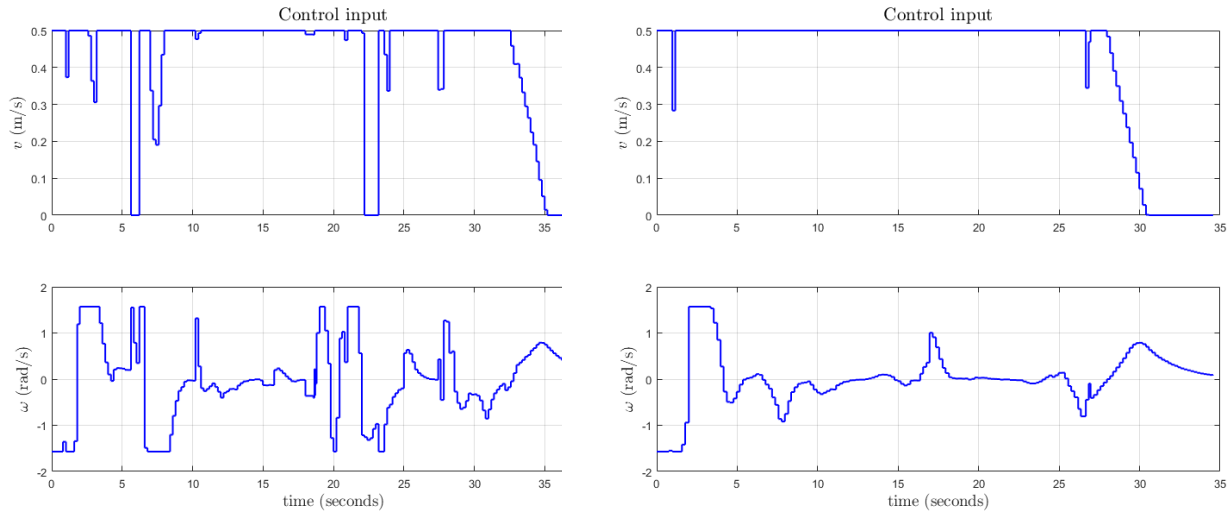
In Figure 5.11, the test scenario can be seen with the initial position being the robot position, and the goal marked as a red arrow indicates the final orientation. The global plan can also be seen as the blue line in the Figure. The two local planners will be tested based on distance travelled, time taken to reach the goal as well as the overall smoothness in the control outputs. In Figure 5.12, the resulting paths that the robot took with each of the two local planners.



(a) Map of the first test with circular obstacles. (b) Map of the second test with polygons as obstacles.

Figure 5.12: Obstacle representation and safety boundary alongside each other.

In Figure 5.12, the initial global plan can be seen in red, and the actual path the robot moved in blue. The global planner does not take into account the tabletops in the global costmap because the mapping is done using a laser scanner that can only detect the legs of the tables. The initial global plan can, therefore, be seen going through one of the tables. However, as the tables move into the local costmap zone, and the robot can detect the table with the RGB-D camera, the global planner remaps the global plan to move around the table. It can be seen in Figure 5.12a that the robot makes some unnecessary movements. This is due to the varying size of the circular obstacles. However, in Figure 5.12b, the robot moves straighter and closer to the obstacles due to their polygon representation. The unnecessary movements become more apparent when looking at the control outputs from the local planner that can be seen in Figure 5.13.

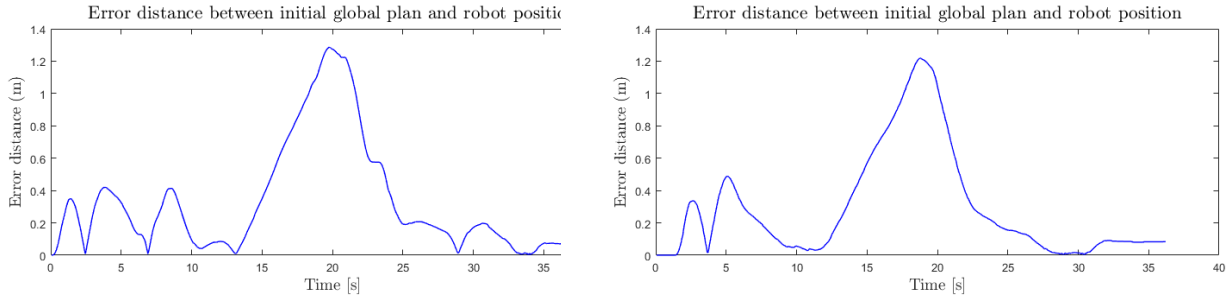


(a) Control outputs of test 1.

(b) Control outputs of test 2.

Figure 5.13: The path of the local planner with centroids and the global plan superimposed on the global costmap.

In test 1, with circular obstacles, the robot stops at certain points to readjust because of the constant change in the size of the obstacles. The planner also changes direction multiple times when it is not needed. In test 2, with the polygon obstacles, the local planner runs at an almost constant speed and only changes the angular velocity when it is necessary to turn. The distance to the initial global plan can be seen in Figure 5.14.



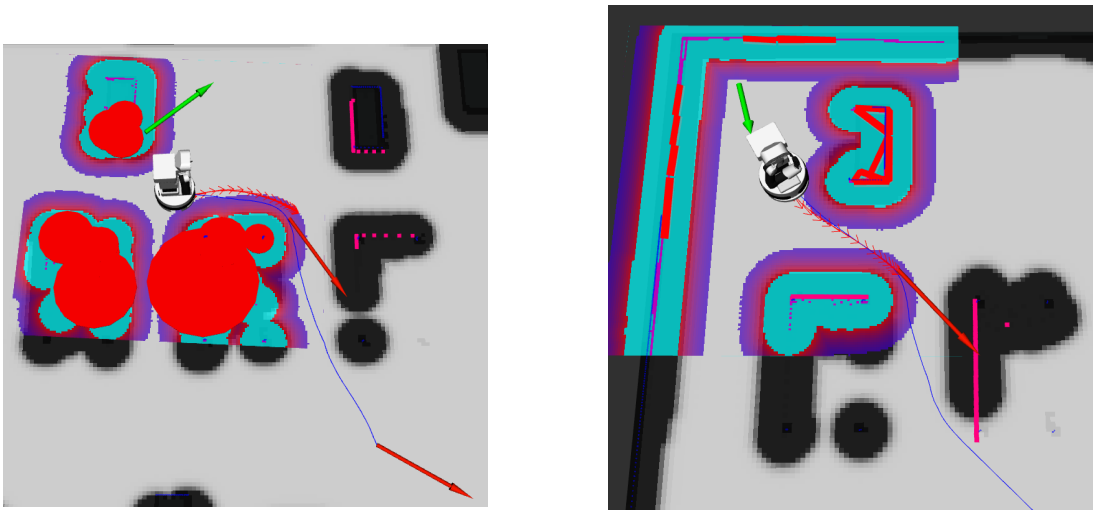
(a) Distance from the global plan in test 1. (b) Distance from the global plan in test 2.

Figure 5.14: The distance from the global plan over time.

In Figure 5.14, it can be seen that test 2 converges to the goal faster than test 1. Due to a minimum velocity of 0 as well as a high cost on changes in acceleration for both linear and rotational velocity, a slight overshoot of the global path results, likely turning to proper goal position after reaching the goal. More results and details concerning the test are available in Table 5.1.

Test nr.	Test parameters	Distance travelled [m]	Min. distance from obs. [m]	Avg. distance from obs. [m]	Time [s]
1	Ts = 0.25 and N = 20	15.43	0.274	0.685	36.6
2	Ts = 0.25 and N = 20	14.25	0.365	0.775	30.57

Table 5.1: Test parameters and results.



(a) Static obstacles represented as circles. (b) Static obstacles represented as points, lines or three-sided polygons.

Figure 5.15: Obstacle representation and safety boundary alongside each other.

From the two tests recorded and through the implementation process into the simulation environment, it has become apparent that the polygon representation of the obstacles is

superior to the circular obstacles. This is mainly due to the sensitivity of the circular representation to irregular polygons and lines. Moving close to walls can create large circles that fill up far more space than the actual size of the obstacles. However, if the obstacles are regular polygons or points, the representation is more accurate.
In Figure 5.15, the obstacle representations can be seen in red on the costmap. In addition to the obstacle representations, the local goal and prediction horizon can be seen as a large red arrow and a set of small arrows, respectively. The global plan and goal can also be seen with the plan being the blue line leading to the global goal at the end of the line represented as a large red arrow. In conclusion, the polygon obstacles representation overall performs better than the circular obstacle representation. However, both methods will be implemented on the robot for real-life testing. Next is the moving obstacles test where static obstacles are not considered.

5.2.1.2 Environment with only moving obstacles

In this simulation test, the local planner has to avoid moving obstacles while moving towards a goal that is 8 meters away from the initial position of the robot. There are, in total, eight moving obstacles that have to be taken into account in this test. The obstacles will have varying speed, heading and size. A table containing the parameters for each obstacle can be found in Table 5.2.

Obstacle nr.	x-position [m]	y-position [m]	Heading [rad]	Speed [m/s]	Radius [m]
1	-2	2.66	-1.57	0.5	0.3
2	-0.05	3.44	-1.57	0.5	0.35
3	-3.02	-2.01	1.57	0.5	0.2
4	-0.09	-3.56	1.57	0.4	0.4
5	2.91	3.95	-2.355	0.3	0.25
6	3.85	2.84	-2.355	0.3	0.25
7	3.97	-3.03	-2.355	0.3	0.25
8	1.35	-3.41	-2.355	0.3	0.25

Table 5.2: The obstacle parameters in the moving obstacle test.

The obstacles can be seen in the map in Figure 5.16 with numbering for each of them.

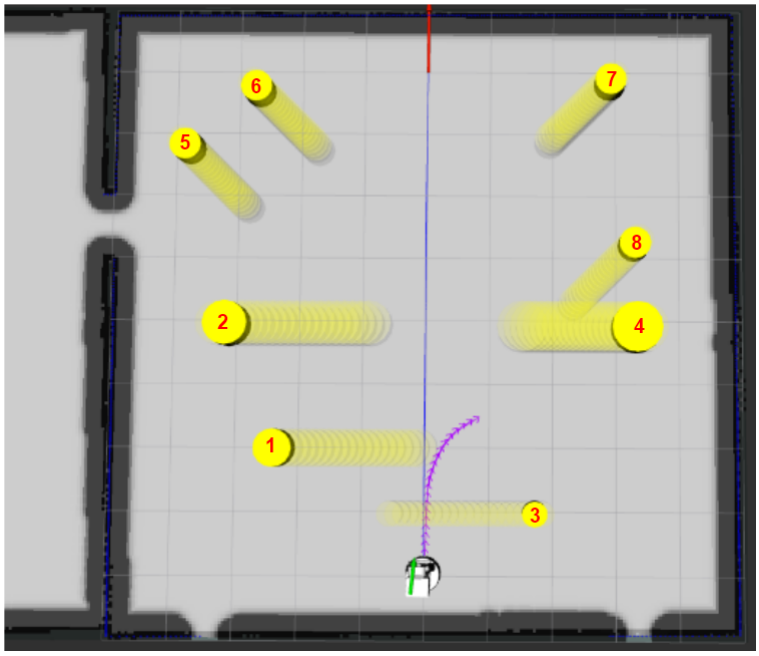


Figure 5.16: The moving obstacle scenario, obstacles visualized with their prediction horizons ahead of them.

As presented in Figure 5.16, the test carried out in an empty room such that there are no static obstacles that have to be avoided. This scenario also gives insight on how the robot would perform in a highly dynamic environment with people moving around it. This test is only carried out virtually due to issues with the robot hardware and with localization. Therefore the results of this test can be found in the Chapter 7 of the report, namely in Section 7.3.

CHAPTER 6
IMPLEMENTATION

The HAMR navigation package has been implemented into the PAL robotics TIAGo software environment. The TIAGo robot is using the Robot Operating System (ROS) as its framework for communication between different pieces of software and sensors, and much more. This Chapter will describe the implementation process and what measures had to be taken to make the package work with the actual robot. Firstly, some of the keynotes about the ROS framework will be described since the ROS framework will be used as heavily in the following sections. Then the navigation stack on the TIAGo robot will be described, and finally, the HAMR navigation package will be described and how it fits into the general structure on the TIAGo robot.

6.1 Robot Operating System(ROS)

In this Section, some of the general concepts of the robotics framework will be outlined. These will be used when describing the package structure in ROS and the communication between these nodes/processes within the packages.

6.1.1 ROS Packages

Workspaces, as well as packages, follow a specific structure when build and run in ROS. In Figure 6.1, the structure of a workspace can be seen with a pack inside.

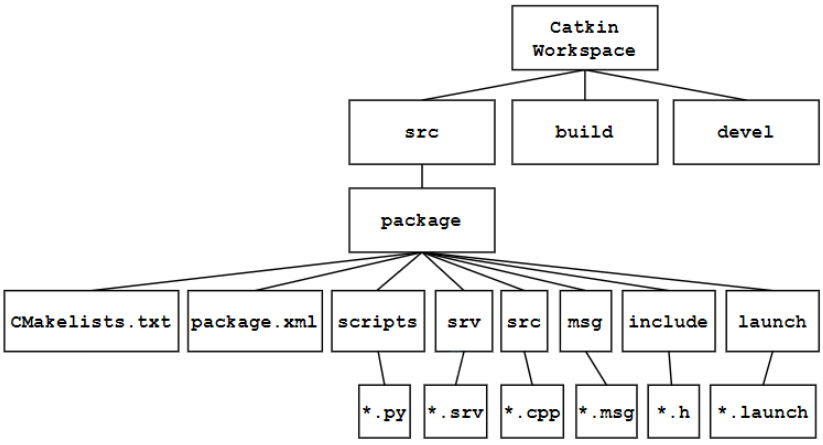


Figure 6.1: The structure of a ROS workspace with a package.

Within the source (src) folder of the workspace, all the packages are stored. A package must have a `CMakelists.txt` which contains the description of how to build the package. A package also needs a `package.xml` which contains information about package names, version numbers, authors and dependencies.

6.1.2 ROS Nodes, Topics & Services

ROS processes are referred to as nodes in a graph-like structure that is connected by edges known as topics. These ROS nodes can pass messages to each other through the use of topics or services. All communication through topics is managed by the ROS master each node has to be registered at the Master when setting up node-to-node communication. This type of communication is often used for one way communication, such as sensor data messages or control signals. Services do not go through the ROS master when sending messages between nodes. Instead, the ROS master sets up peer-to-peer communication between all the nodes after they have been registered with the ROS master. ROS services work per request and reply and can be useful for two-way communication between nodes. A simple illustration of the ROS topic communication can be seen in Figure 6.2.

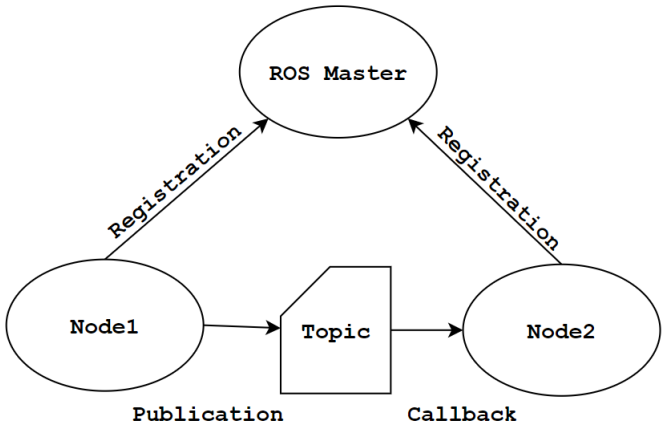


Figure 6.2: Communication between two nodes registered to the ROS master through a topic

The publisher publishes messages at some desired rate to the topic then the subscriber 'listens' to this topic and every time it receives a message, it runs the callback function. The subscriber often handles most of the processing of the received data inside the callback function.

6.1.3 Transformations & Frames

Another important topic within ROS and robotics, in general, is coordinate frames and the transformations between them. The TIAGo robot has many different coordinate

frames. Some examples could be the camera frames in the head of the robot and the `base_footprint` frame. ROS has a package called `tf` that is used to keep track of all the coordinate frames over time. The `tf` package allows for transformations of points and vectors between the different coordinate frames. As mentioned before the `tf` package can not only transform between different coordinate frames but also transforming in time. Firstly, a regular transformation from some a to some frame c through some intermediate frame b:

$$T_a^c = T_a^b \, T_b^c \tag{6.1}$$

Now imagine an observation was made at time t_0 and it is now time t_1 there is now a missing term in the Equation 6.1. A transformation between time t_0 and t_1 has to be added in between the two transformations as such:

$$T_{a@t_0}^{c@t_1} = T_{a@t_0}^{b@t_0} \, T_{b@t_0}^{b@t_1} \, T_{b@t_1}^{c@t_1} \tag{6.2}$$

This way, coordinates can be transformed through time using the `tf` package [53]. All frames are managed in a `tf` tree with the `/map` or `/world` frame as the root in it. The `tf` tree for the TIAGo robot can be found in the Github repository [54]. In Figure 6.3, the TIAGo robot can be seen in the ROS visualization tool `Rviz` with all its frames.

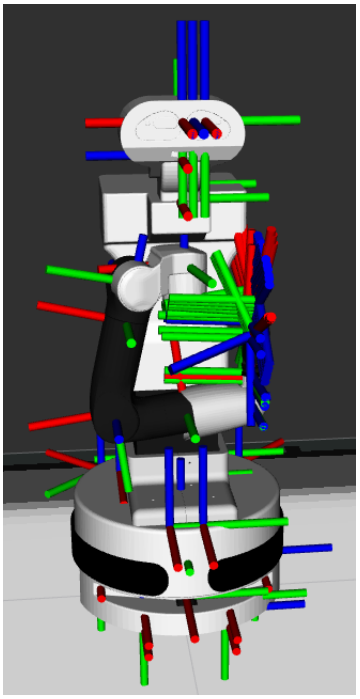


Figure 6.3: The TIAGo robot with all its coordinate frames.

With this short introduction to the concepts of ROS, the next section will continue with

a description of the TIAGo navigation stack.

6.2 TIAGo Navigation Stack

The navigation stack takes in the information from odometry, sensor streams and goal pose and outputs velocity commands that actuate the robot towards the goal position. These primary components are the `move_base`, `Map_server`, sensor sources, odometry source, `tf` transforms described in the previous section and finally the `amcl`. An illustration of the TIAGo navigation stack can be seen in Figure 6.4.

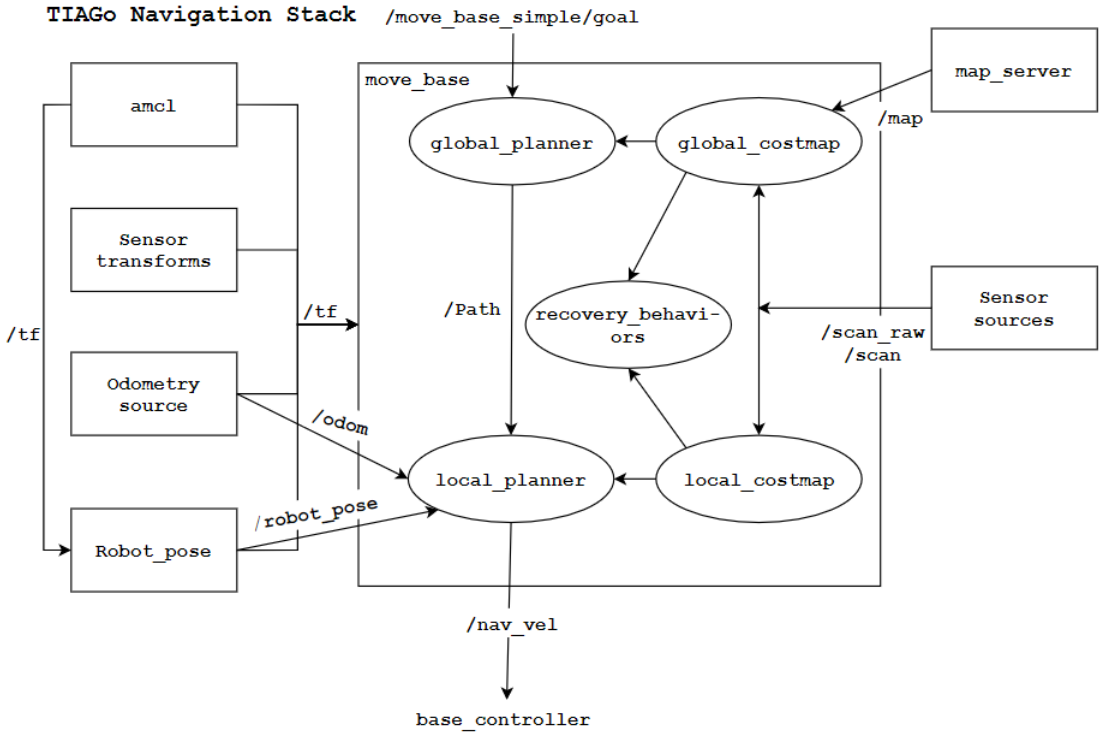


Figure 6.4: Diagram of the TIAGo navigation stack.

The `move_base` carries out the planning (global and local), costmap definition, bypassing obstacles and recovery features when the robot is stuck. The output of this structure is the velocity commands that actuate the robot. The `Map_sever` feeds the map to the `move_base`. This map can either be downloaded if the area is already mapped or the area can be mapped by the robot itself using gmapping. The primary sensor sources on the TIAGo robot is the RGB-D camera in the head and the lidar in the base. The odometry source comes from the encoders in the wheels of the TIAGo robot. These give a temporary position estimate while the map updated using the `amcl`. Adaptive Monte Carlo Localization (`amcl`) is a probabilistic localization system for a robot moving in 2D.

The `amcl` plugin is based on using particle filtering for position estimation. This localization system helps the robot cope with error in position and orientation accumulating by integrating encoder ticks.

6.2.1 Deployment of Packages

In order to develop packages on the TIAGo robot, a PAL SDE development computer based on the Linux Ubuntu 16.04 LTS distribution is needed. This development computer is capable of building a workspace on the computer itself and afterwards it can be moved to the TIAGo robot via Secure Shell(ssh) connection. When the TIAGo robot boots up, it creates two sources of packages to its ROS environment. One is the ROS software distribution `erbium`, and the other is a fixed location at `/home/pal/deployed_ws`, which is where the deployment tool installs the packages. The `deploy` function follows the rules of the `CMakelists.txt`, so everything needed must be added to it when the workspace is built. These are the main features of the TIAGo navigation stack and how to deploy packages to that environment. The next section will describe how this navigation stack has been modified in this project.

6.3 HAMR Navigation Package

This section will specifically outline the structure of the navigation package developed in this project as well as the pieces of software used to interface with the TIAGo software environment. There are four packages created in this project. Three of these nodes, namely the Moving Obstacle Publisher, Local Goal Publisher, and the Costmap Conversion Publisher are used to feed the input from the TIAGo navigation stack to the `CasADi` MPC node. The `CasADi` MPC node is then responsible for sending control signals to the actuators. A modified version of the TIAGo navigation stack can be seen in Figure 6.5.

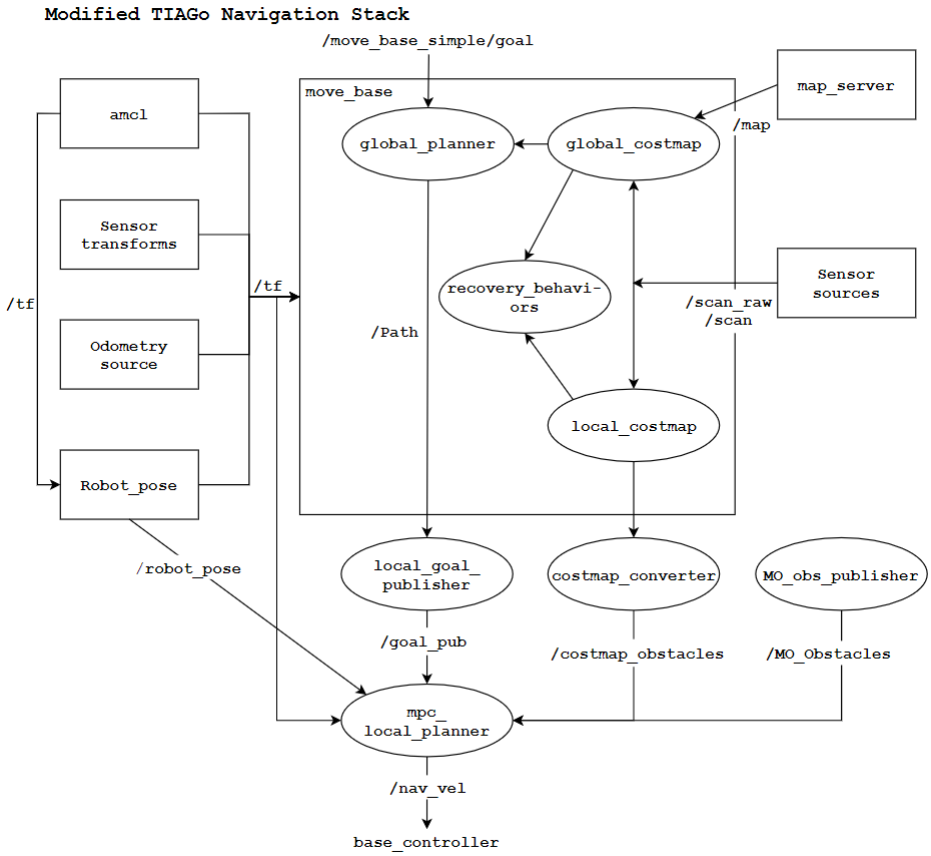


Figure 6.5: Modified navigation stack with the local planner outside the `move_base`.

As seen in Figure 6.5, the local planner the local goal position from the local goal publisher. The `local_goal_publisher` finds the closest pose to a set distance of 1.2 meters from the robot in the global plan. The `local_goal_publisher` then publishes this pose to the robot at the same rate as the global plan, which is approximately 2 Hz. The `costmap_converter` publishes the obstacles in the `local_costmap` to the robot at a variable rate depending on how much the costmap updates over time. A completely separate node from the `move_base` is the `MO_obs_publisher` this publishes virtual obstacles which are only handled in the MPC. This is mainly used for testing the MPC and is merely a placeholder for a perception package, which should have the same output. Lastly, the local planner gets feedback from the `Robot_pose`, which publishes the estimated position of the robot based on the `amcl` and the odometry. With these nodes, the local planner does not directly interact with the `move_base`; however, in principle, it works the same way. If the local planner had to directly interact and be a part of `move_base`, it would have to be written as a plugin in `C++`. Having the local planner outside the `move_base` allows for the planner to be written in any language adding flexibility to the system. This is beneficial since most documentation for the solver used in this project, namely `CasADi`,

is written in Python. A simplified graph of this is shown in Figure 6.6.

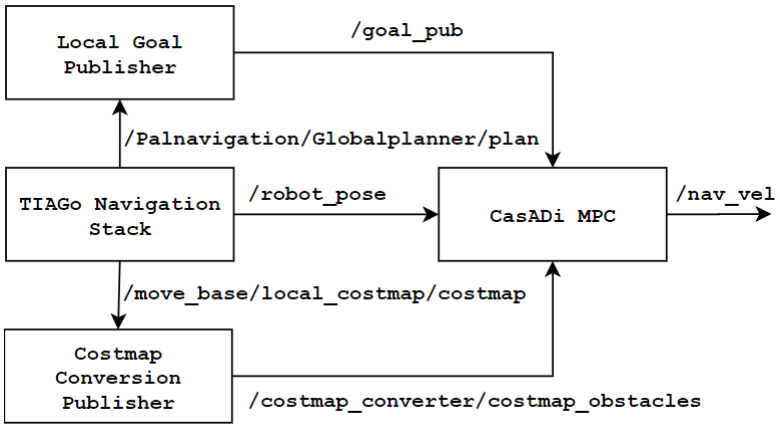


Figure 6.6: Simplified graph of the navigation package.

An RQT-graph of the entire node network, including all implemented nodes from this project, can be found on the Github repository [54]. An overview of the topics used in the navigation package can be seen in Table 6.1.

Name	Message Type	Update Frequency
`/robot_pose`	`/PoseStamped`	50 Hz
`/plan`	`/Path`	2 Hz
`/goal_pub`	`/PoseStamped`	2 Hz
`/costmap_obstacles`	`/ObstacleArrayMsg`	Variable
`/costmap`	`/OccupancyGrid`	Variable
`/nav_vel`	`/Twist`	5 Hz
`/MO_Obstacles`	`/ObstacleArrayMsg`	50 Hz

Table 6.1: List of the topics mainly used in the navigation package.

The core of the modified navigation stack is the NMPC local planner script. This has mainly been developed in Python as mentioned earlier. Furthermore the NMPC has made used of `CasADi` for defining and solving the NLP problem, the next section will describe the process formulating an OCP in `CasADi`.

6.4 OCP with `CasADi`

`CasADi` is an open-source software tool for numerical optimization in general and optimal control (i.e. optimization involving differential equations) in particular [55]. In order to formulate and solve the OCP and NLP problems of the NMPC local planner, the model

has to be established in CasADi. This is done using CasADi's symbolic framework that allows for the model of the robot and the functions to be constructed using symbolic expressions [56]. A model can be established by first constructing the state and control inputs to the system. This can be seen in the Listings 1 and 2.

```
x = ca.SX.sym('x')
y = ca.SX.sym('y')
theta = ca.SX.sym('theta')
states = ca.vertcat(x, y, theta)
```

Listing 1: The state vector.

```
v = ca.SX.sym('v')
omega = ca.SX.sym('omega')
controls = ca.vertcat(v, omega)
```

Listing 2: The input vector.

Once the state and input vector has been established using CasADi's SX data type used to represent matrices whose elements consist of symbolic expressions. The right-hand side of the state equations is then established, and a mapping function is used to map the previous state and control into the new state. The right-hand side of this project:

```
rhs = ca.vertcat(v * ca.cos(theta), v * ca.sin(theta), omega)
```

The mapping function that is part of the CasADi framework as well can be written as:

```
mapping_func = ca.Function('f', [states, controls], [rhs])
```

This mapping function can then describe the evolution of the system based on what integrator/discretization is used. As mention before in this project RK4 integrator is used. Once the model is set up, constraints are constructed for the states, inputs, static obstacles and moving obstacles. To understand how the constraints are fed into the NLP solver, the form of the parametric NLP that CasADi solves is formulated.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x,p) \\ \text{subject to:} \quad & x_{lb} \leq x \leq x_{ub} \\ & g_{lb} \leq g(x,p) \leq g_{ub} \end{aligned} \tag{6.3}$$

where lb and ub denote the lower and upper bound of the constraints, respectively. The parameter p is a vector that contains all the values for the current state, goal, obstacle parameters etc. $g(x,p)$ is a non-linear constrained function that has to be bounded. An example of $g(x,p)$ is the position of a moving obstacle relative to the robot, x is the state of the robot and p is the parameters of the moving obstacle. The optimization variables, the objective function, $g(x,p)$ and the parameter vector p are used to set up the NLP problem. This is done using the CasADi function `nlpsol()`. A specific solver also has to be picked, the open-source Interior-Point Optimization(IPOPT) method that is included in the CasADi installation is used in this project. The setup in python then becomes:

```
nlp_prob = {'x': OPT_variables, 'f': obj, 'g': const_vect, 'p': P}
solver = ca.nlpsol('solver', 'ipopt', nlp_prob)
```

Once the solver has been constructed, the parameters can be given to solve the problem. The parameter vector, current bounds(lbx, ubx, lbg, ubg) and current state can be given to the solver.

```
sol = solver(x0=x0k, lbx=lbw, ubx=ubw, lbg=lbg, ubg=ubg, p=p)
```

The `sol` array then contains the predicted states and control inputs for the optimal path to the goal. This is how `CasADi` is used to formulate and solve the OCP in this project. The next Chapter will investigate the performance of the two solvers developed in this project through a series of tests.

CHAPTER 7

TESTING & RESULTS

Testing of the system is carried out to evaluate the performance of the navigation package developed by the project group. This evaluation is carried out partially as a comparison with the original navigation system in place for the PAL moving base(PMB) robot. This comparison is the foundation for evaluating how well the robot fulfills the task of navigating a dynamic environment including static and moving obstacles. To compare the navigation system already in place on the PMB, with the MPC based navigation package, a static environment test is carried out.On top of this another test involving moving obstacles is carried out to determine the performance of the NMPC local planner, with respect to moving obstacles. Lastly a scenario is set up to simulate a possible real-life interaction between the robot and a human/moving obstacle in an implementation environment.

7.1 Mapping

For testing the developed robot navigation system the control lab at AAU's Fredrik Bajers vej 7C building is used. A ground plan of the laboratory can be seen in Figure 7.1, from which the main room (2-104) was used.

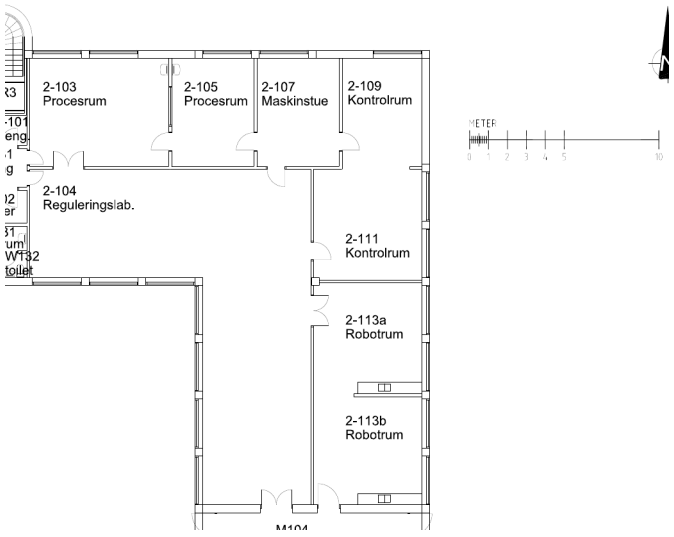
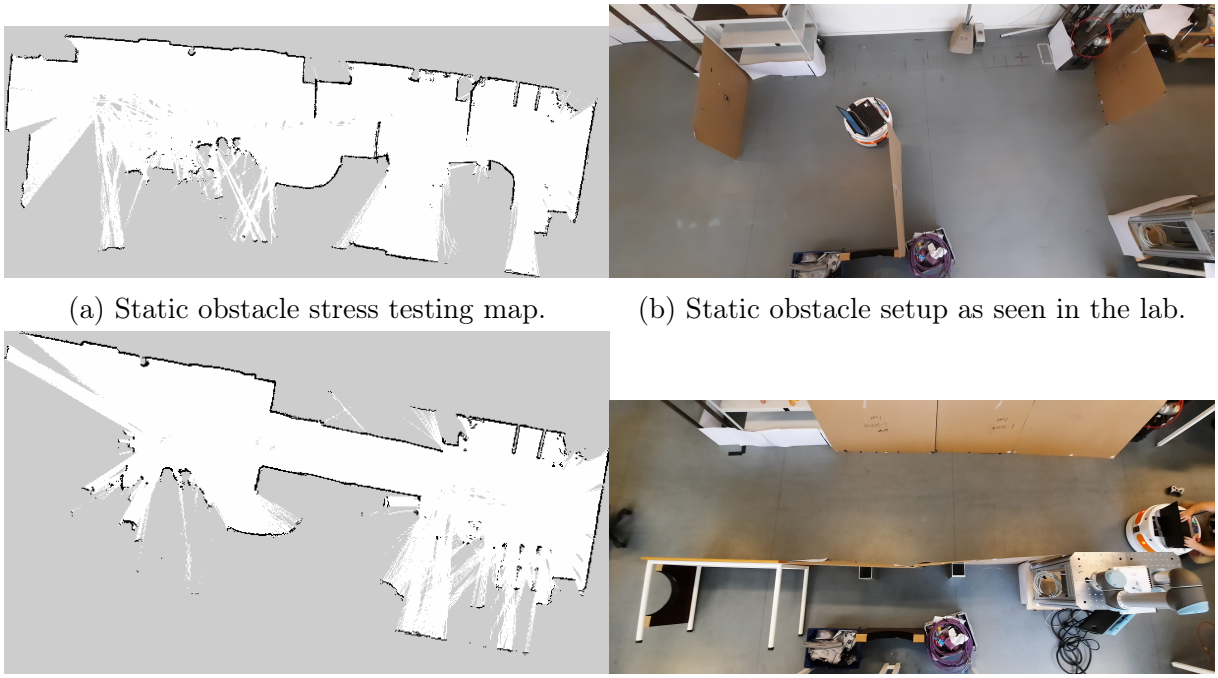


Figure 7.1: Plan of the Control and Automation Laboratory.

Using the PMB's built-in mapping functionalities the lab was mapped out for the different

planned tests. The maps that were created can be seen in Figure 7.2.



(a) Static obstacle stress testing map.

(b) Static obstacle setup as seen in the lab.

(c) Hallway scenario map.

(d) Hallway Scenario as seen in the lab.

Figure 7.2: The two main maps used for robot implementation testing

7.2 Static Obstacle Avoidance

This test is carried out to compare the capability of static obstacle avoidance for the different developed planners in this thesis. The procedure for the static obstacle testing is as follows: The robot has to move through a small mapped area with static obstacles, the mapped area can be seen in Figure 7.2a. The static obstacles consist of walls made of cardboard and paper walls along the sides of the workspace. There will be a constant starting position and end position throughout all tests of this scenario. The tests are carried out for the two local planners developed by the research group and one proprietary planner developed by PAL robotics. Firstly, the two planners are tuned with the parameters of time-step size (T_s) and horizon length (N). Afterwards tests are conducted multiple times for each planner to test reliability and performance.
Before testing for performance of the NMPC local planner, tuning was performed to figure out a fitting interval of tuning parameters to compare performance with the PAL local planner that is already implemented on the robot. During this it was determined a lower max turning speed (ω_{max}) was needed, from the initial $\pi/2$ down to $\pi/4$, to eliminate the chance for overshooting rotational movement and to reduce the likelihood that the MPC

prioritizes turning. During the tuning procedure of this test an initial focus was upon the horizon of the robot, as the path in question consists of multiple twists it is important that the horizon be long enough to circumnavigate possible obstacles between robot and local goal effectively.

7.2.1 Testing

For evaluating the planners made by the group an initial number of tests are carried out to compare tuning parameter performance, followed by a number of test runs with set parameters for consistency of planner behaviour. These tests are evaluated based on the distance from the global goal, time to complete given navigation task, overall driven distance as well as the given control outputs(`nav_vel`), these work as metrics for optimality of the route taken. An example of the RViZ visualization of these tests can be seen in Figure 7.3 [1].

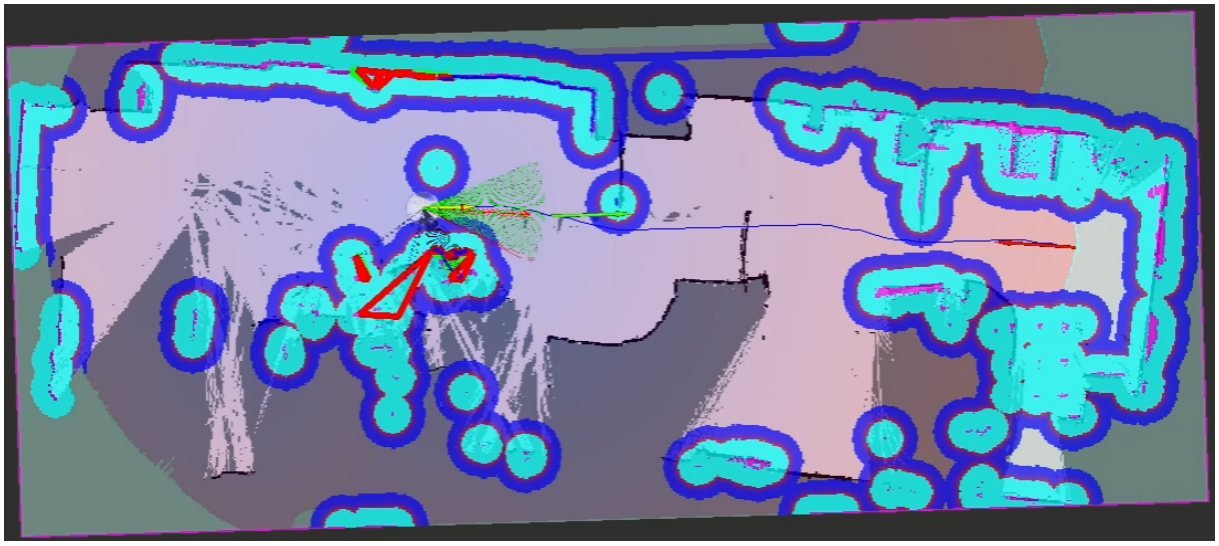


Figure 7.3: The lab setup and system information as vizualized in RViZ, a recording of which can be seen here

7.2.1.1 NMPC Local Planner with Polygon Centroid Representation

The NMPC Local Planner with Polygon Centroid Representation was tested for different time step values; 0.1, 0.25 and 0.5 as well as different maximum robot velocities 0.3 m/s and 0.5m/s. One of the first tests carried out with the centroid planner can be seen in Figure 7.5, showing the resulting behaviour of a close quarters environment, arising especially from the fact that obstacle sizes are increased using this method.

[1] https://youtu.be/ajY194H95yk

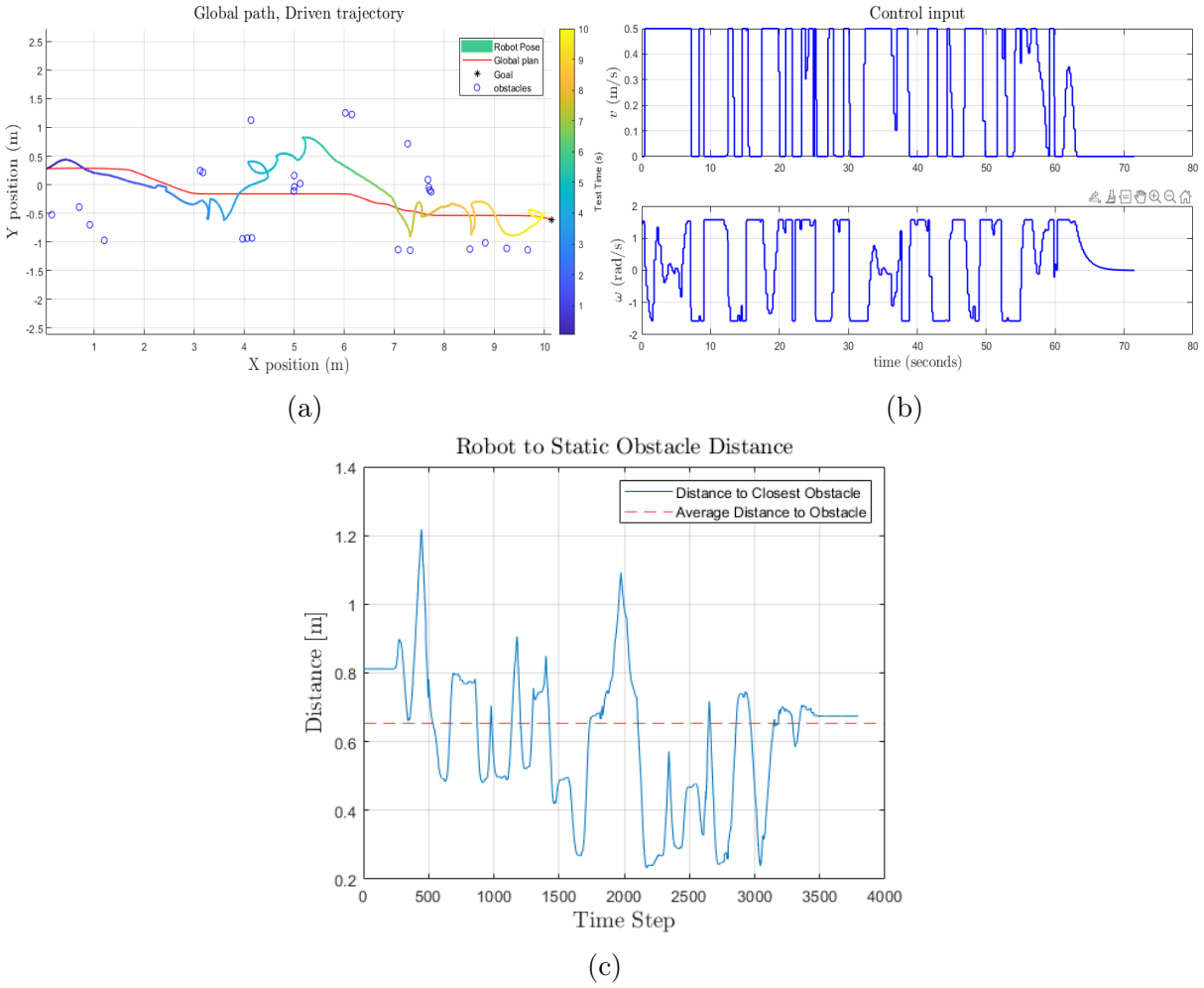


Figure 7.4: Values for test of centroid planner with values of $T_s = 0.1$ and $v_{max} = 0.5$

From this can be seen that tuning is necessary, also why no more tests were carried out with such a short horizon, since the robot repeatedly gets into situations where it cannot pass obstacles. After initial tuning and from one of the later tests can be seen smoother behaviour, in Figure 7.5. Smoother performance, though still the robot arrives in some situations where it has to turn almost all the way around to get out of, as can be seen by the linear velocity outputs going to 0.

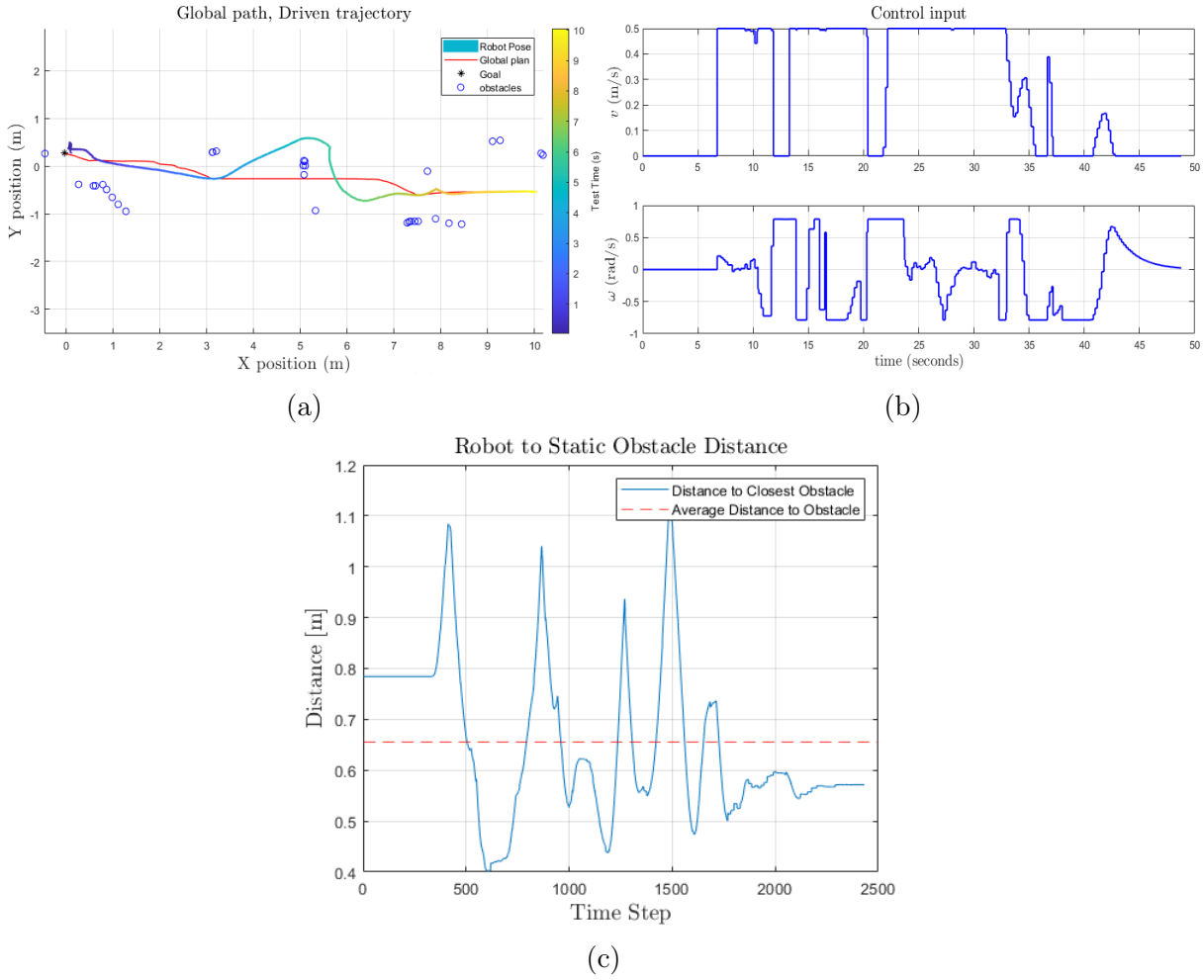


Figure 7.5: Values for test of NMPC planner with centroid polygon representation with values of $T_s = 0.5$ and $v_{max} = 0.5$

For the different parameters tested average results were calculated as can be seen in Table 7.1. From this can be seen a significant decrease in travelled distance for longer predictions arising by the larger time step.

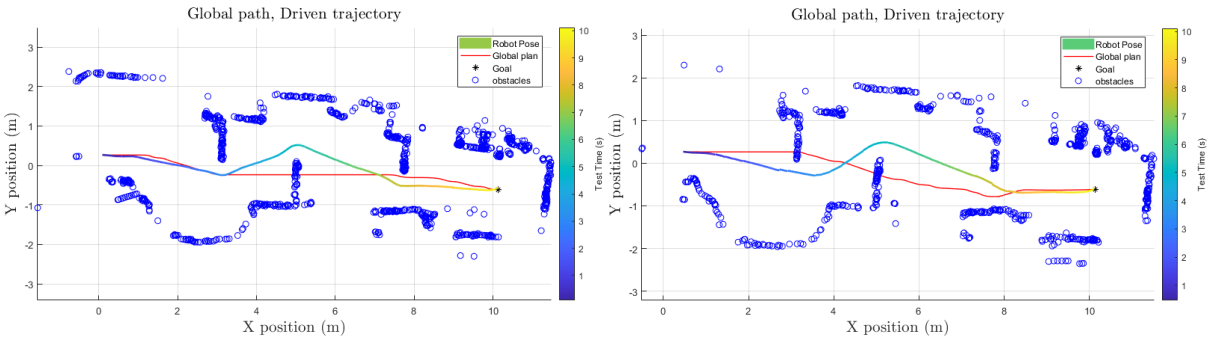
Timestep [s]	Robot Max Velocity [m/s]	Global Plan Distance [m]	Robot Travelled Distance [m]	Travelled distance - Global distance difference [m]	Average distance Robot to Global plan [m]	Time [s]	Distance to Obstacles [m]
0.25	0.3	10.1835	14.8928	4.7093	0.2195	62.0774	0.6111
0.25	0.5	10.2420	14.7534	4.5114	0.3901	44.5088	0.6482
0.5	0.5	10.2383	11.2465	1.0081	0.1507	28.9006	0.6982

Table 7.1: Average values for tests carried out at given Timestep values

7.2.1.2 NMPC Local Planner with Polygon Representation

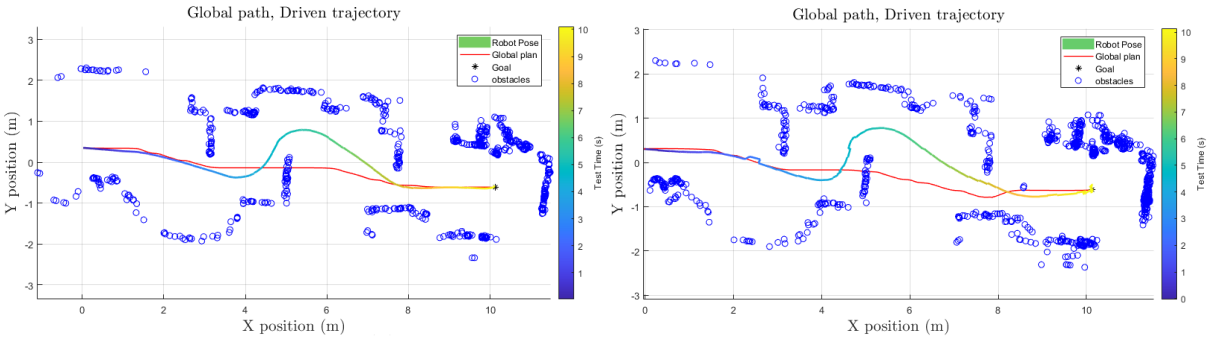
For the polygon test 4 different robot velocities (v_{max}) were tested for; 0.3 m/s, 0.5 m/s, 0.75 m/s and 1 m/s. As well as time step size (T_s); 0.1, 0.25 and 0.5. The values for

these tests for the Polygon test can be seen in Appendix B.1. An example of each of the velocities tested for can be seen in Figures 7.6 and 7.7.



(a) low speed, 0.3 m/s, testnr.: 40

(b) medium-low speed, 0.5 m/s, testnr.: 44

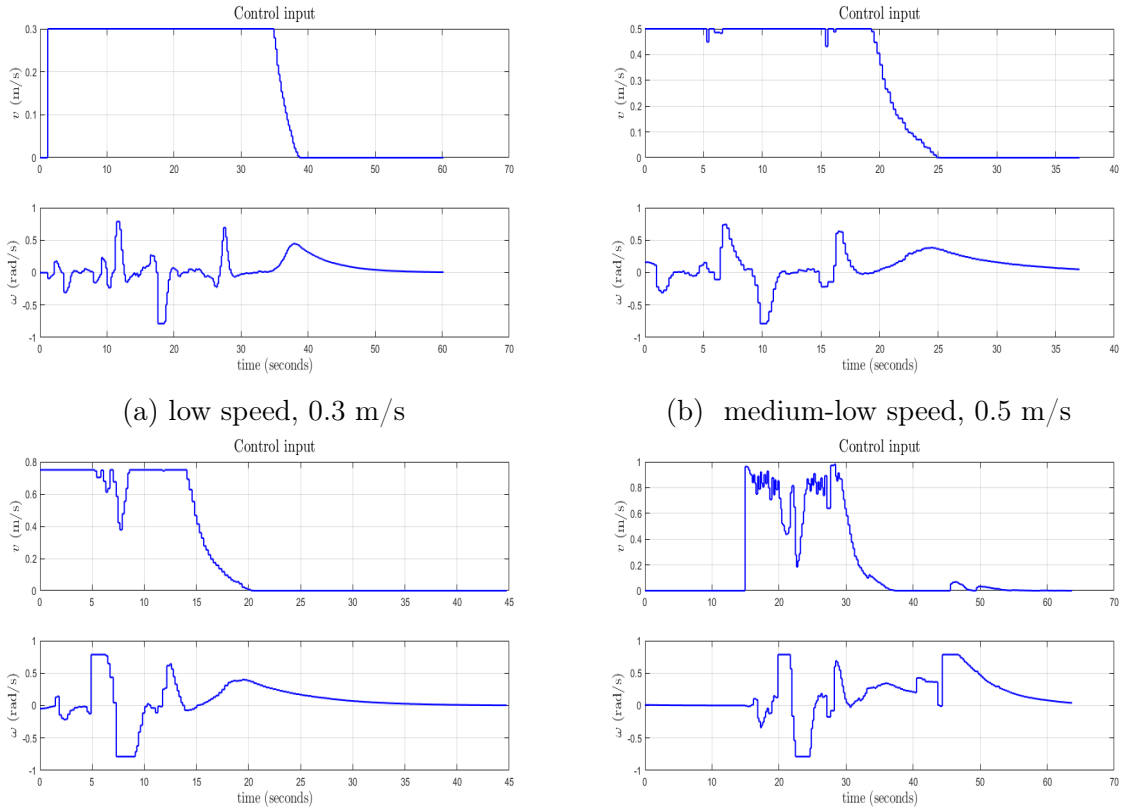


(c) medium speed, 0.75 m/s, testnr.: 45

(d) high speed, 1 m/s, testnr.: 47

Figure 7.6: Example in difference in performance of different speed values with same horizon parameters.

Figure 7.6 shows in all tests a common trend of the global planner to disregard the middle wall in the environment as seen in Figure 7.2c of the laboratory layout. Also as can be seen from the robot paths that as the maximum linear velocity of the robot increases it can lead to overshooting, as seen in Figure 7.6c. A video of the exact behaviour of the robot can be seen here [57].
The behaviour of the local planner can be analyzed more precisely from the control outputs, these are shown in Figure 7.7.



(a) low speed, 0.3 m/s

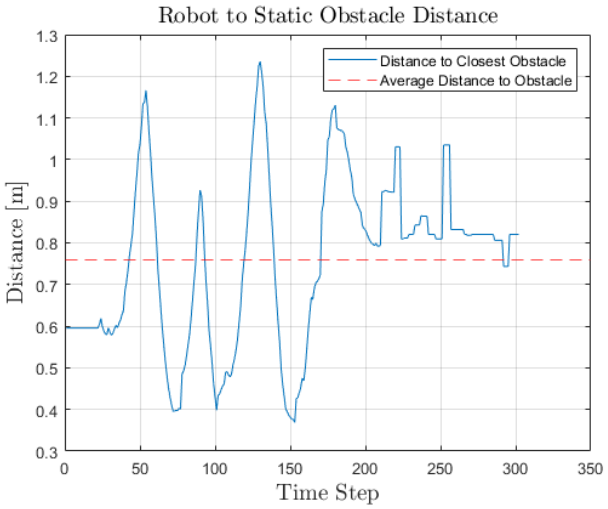
(b) medium-low speed, 0.5 m/s

(c) medium speed, 0.75 m/s

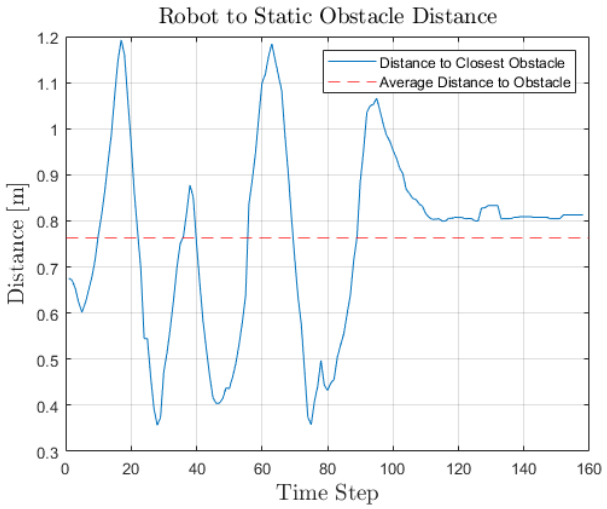
(d) high speed, 1 m/s

Figure 7.7: Four different parameters were tried in the static obstacle test.

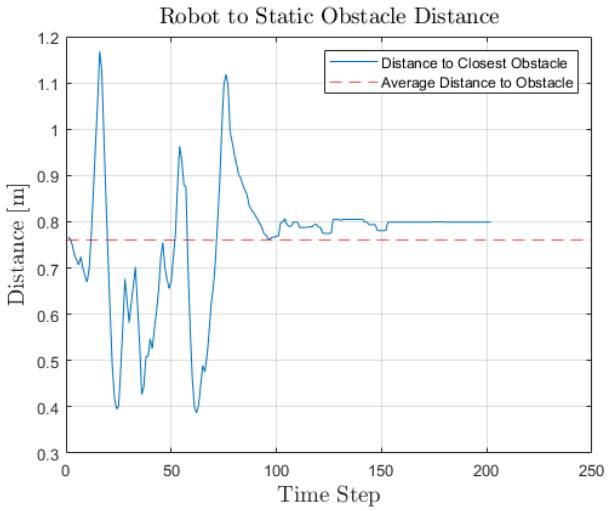
In Figure 7.7 it can be seen for the slower tests they have more smooth performance, given the pretty much constant speed throughout the test. In the higher speed tests rougher control input curves are seen because the robot has to slow down after too high speeds cause it to overshoot the obstacles. Also the distance between the robot and obstacles have been taken into account, as it should remain relatively stable over the course of these tests, to see if different parameters have any impact on how well the robot manages to keep away from the obstacles.



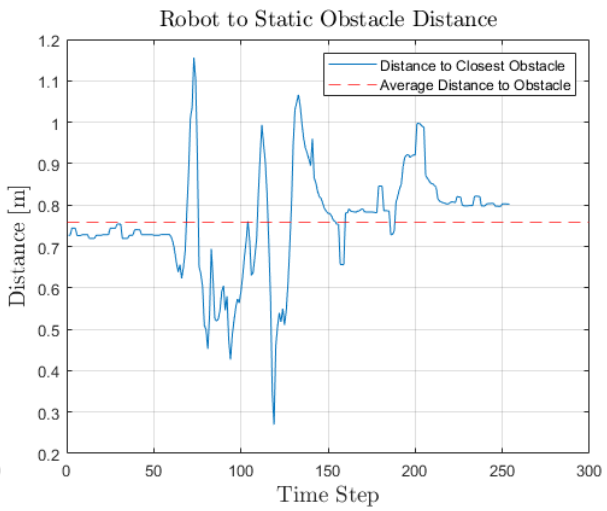
(a) low speed, 0.3 m/s



(b) medium-low speed, 0.5 m/s



(c) medium speed, 0.75 m/s



(d) high speed, 1 m/s

Figure 7.8: Obstacle distance for four different velocities tested for SO performance

Above can be seen the impact of changes in velocity on system performance. From Figure 7.8d that with a robot velocity of 1m/s the robot overshoots its targets and therefore violates the safety distance set for the robot, in this case a robot radius of 0.27 and a safety boundary of 0.1, meaning the minimum distance should be around 0.4. Along with these multiple tests were carried out for different timesteps, some averaged results for which are shown in Table 7.2.

Timestep [s]	Global Plan Distance [m]	Robot Travelled Distance [m]	Distance between Travelled and Global plan [m]	Time [s]	Distance to Obstacles [m]
0.1	10.2349	10.9297	0.1928	50.4802	0.6812
0.25	10.2225	10.7488	0.1661	41.9704	0.7148
0.5	10.2135	11.1124	0.1267	35.6688	0.7247

Table 7.2: Average values for tests carried out at given Timestep values with robot velocity 0.3 m/s

For these timestep consistency tests a minimum of 10 runs through the static obstacle course were performed. Showing that for higher timesteps the robot adheres more closely to the global plan, being able to consider more obstacle parts in the planned horizon.

7.2.1.3 PAL Local Planner

The PAL local planner was tested and used as a benchmark for the local planners designed in this project. Throughout all the tests the PAL local performed the task with a high max speed of 0.8m/s. For the PAL local planner no parameters were changed. The test runs carried out are illustrated in Figure 7.9.

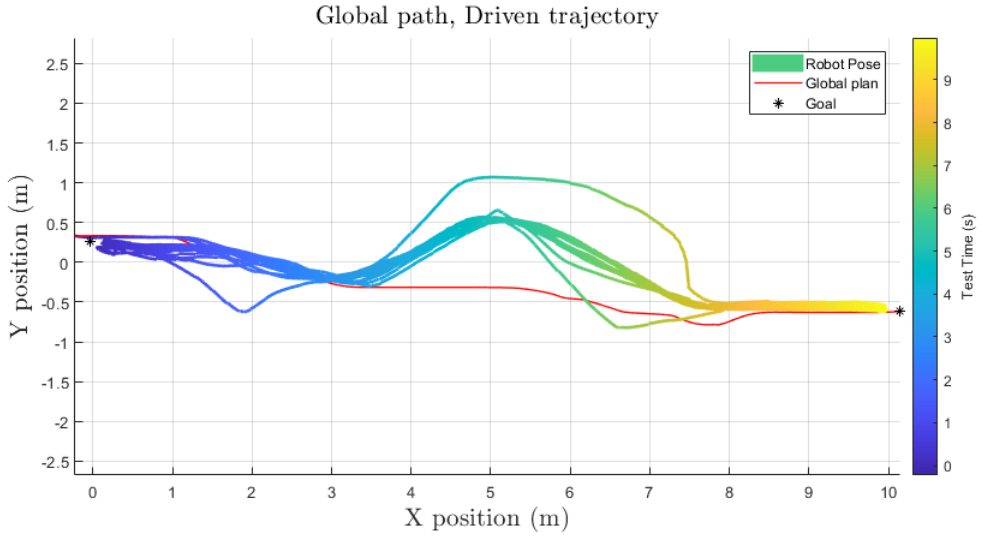


Figure 7.9: All the different paths taken by the local planner used by the PMB.

A total of 13 runs back and forth through the testing environment were carried out. Shown in Figure 7.9, all but two tests providing a very smooth similar performance, the latter two tests having an extra introduced obstacle, shown by the two outlier paths. A differing global path shown for the path towards the right and one for the path to the starting position at the left. Gathering again the difference between the initial global goal and the final robot movement. Data seen in B.1. The values mostly showing as well

uniformity over the different runs. For the different planner tests a number of tests were undertaken, data gathered for each, averaged as can be seen in Table 7.3

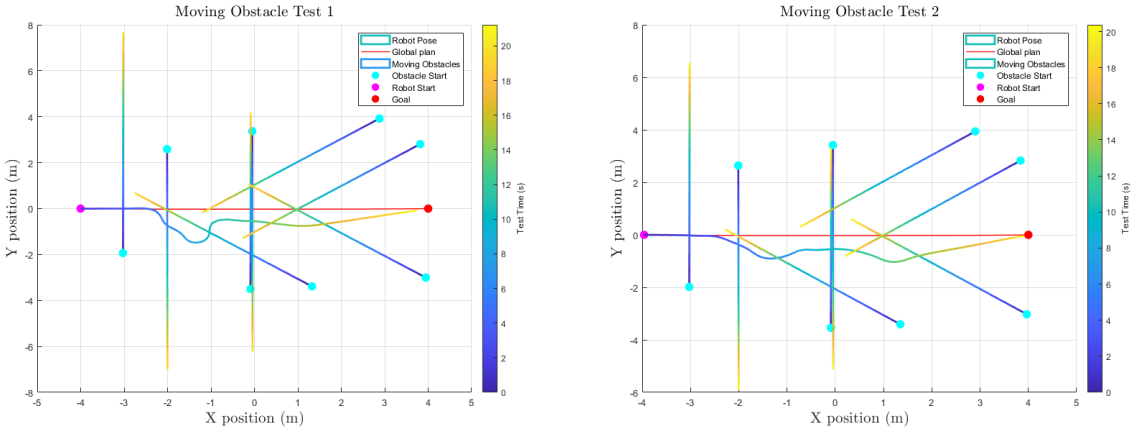
	V_max	Ts [s]	N	global path length	robot path length	robot path global path error	Average distance to global plan	End Time	Average Distance to Obstacles
PalNav	0.8			11.4027	10.9038	0.7090	0.1246	23.8819	
Centroid	0.3	0.1		10.2086	14.7694	4.5608	0.2094	66.9201	0.5624
	0.3	0.25		10.2484	11.9192	1.6708	0.1964	46.5279	0.7144
	0.5	0.5		10.2420	14.7534	4.5114	0.3901	44.5088	0.6482
	0.5	0.5		10.2383	11.2465	1.0081	0.1507	28.9006	0.6982
Polygon	0.3	0.1	20	10.2349	10.9297	0.6948	0.1928	50.4802	0.6812
	0.3	0.25	20	10.2173	10.7406	0.5233	0.1574	42.0602	0.7202
	0.3	0.5	20	10.2142	10.8482	0.6339	0.1417	42.7113	0.7114
	0.5	0.5	20	9.8403	10.5173	0.6770	0.1605	25.1127	0.7621
	0.75	0.5	20	10.2753	11.1609	0.8856	0.1015	21.7980	0.7246
	1	0.5	20	10.3367	11.8899	1.5532	0.1051	40.7326	0.7326

Table 7.3: Averaged results for the static obstacle tests per test parameters.

From the above table, noticeable results are the average times to complete the tests. From the tests conducted at with a maximum robot velocity of 0.5m/s and 0.75m/s a performance comparable to the PAL planner is achieved. As well a close or even smaller resulting error to the global path is achieved. In Appendix B.1 can be seen few outlier results with some exceptional metrics, these have not been included in the calculation of average test performance. The following sections describes the test carried out in simulation of avoidance of moving obstacles in an open environment with multiple moving actors.

7.3 Moving Obstacle Avoidance

One of the main functionalities of the developed NMPC local planners is to circumnavigate moving obstacles of varying sizes and speeds. This test is performed in simulation due to technical difficulties with the wheels on the TIAGo robot. An empty room is used as the map for testing, with the robot moving from one end of the room to the other. Keeping the goal and starting point fixed in the map. The varying parameters in this case will be the horizon (N) and the time step (Ts). Eight moving obstacles will be introduced in the space and the robot will have to avoid all of them to get to the goal. The space with the obstacles has been shown before in Rviz as seen in Figure 5.16.

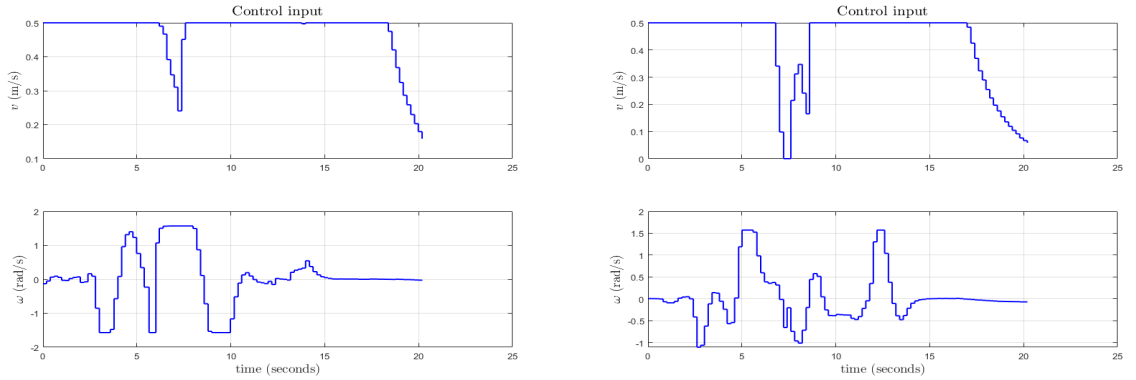


(a) The positions for test 1 in time.

(b) The positions for test 2 in time.

Figure 7.10: Plots showing the movement of the robot and obstacles in time with the global plan.

The two first tests moved through the space successfully without collision and minor unnecessary movement their. The parameters for these tests can be found in Table 7.4. Both paths are relatively similar in terms of position and time it took to get to the goal. This is also apparent in the control outputs in Figure 7.11.

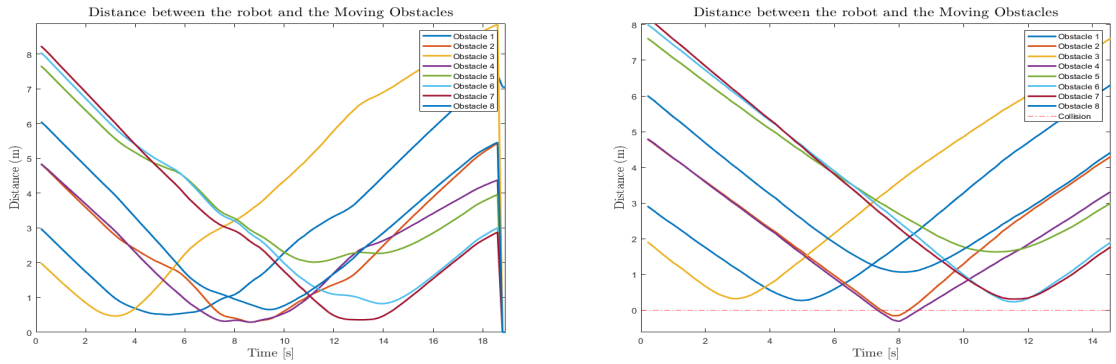


(a) The control outputs for test 1.

(b) The control outputs for test 2.

Figure 7.11: Plots showing the control outputs: linear velocity and angular velocity.

Test 2 in Figure 7.11b changes direction for longer than test 1 and slows down for longer than test 1. These are however minor differences and the two tests perform quite similarly. Both tests in Figure 7.10 also had good average distance from the obstacles as well as an acceptable minimum distance from the obstacles. However when changing the time step in test 3 to $Ts = 0.5$ the robot collided with obstacle 2 and 4 as seen in Figure 7.12b(These obstacles can be seen with numbering in Figure 5.16).



(a) Distances represented for test 2

(b) Distances represented for test 3

Figure 7.12: Plots showing the distances to each obstacle in meters with S_b, rr and r_{obs} accounted for.

One of the likely causes of the collision is that the time step is simply too long and the robot predicts too far into the future trying to make up for obstacles much further ahead than the robot itself. Since there is no cost for obstacles closer all obstacles within the prediction horizon are treated equally and the local planner tries to consider too many thing at once. There is also the opposite end of the scale as seen in test 4 in Table 7.4 the horizon and time step can be too short for the robot to take the obstacle properly into account before its too late to react.

Test nr.	Test parameters	Collision	Avg. dist to obs. [m]	Min. dist. to obs. [m]	Distance travelled [m]	Time [s]
1	Ts = 0.25 and N = 20	No	0.329	0.105	9.36	21.26
2	Ts = 0.15 and N = 20	No	0.336	0.28	8.59	20.41
3	Ts = 0.5 and N=20	Yes	0.295	-0.315	7.96	15.14
4	Ts = 0.25 and N = 15	No	0.279	0.115	10.64	24.17
5	Ts = 0.15 and N = 15	Yes	0.259	-0.241	10.70	25.6
6	Ts = 0.5 and N = 15	Yes	0.337	-0.284	7.94	18.37
7	Ts = 0.25 and N = 25	No	0.257	0.152	11.75	24.59
8	Ts = 0.15 and N = 25	Yes	0.238	-0.293	8.01	18.5
9	Ts = 0.5 and N = 25	Yes	0.226	-0.306	7.98	15.4

Table 7.4: These are the parameters and results from the 9 tests done relating to moving obstacle avoidance.

Further interpretation of the results will be can be found in Chapter 8. The next section will describe the hallway scenario with moving obstacles and static obstacle together.

7.4 Hallway scenario

This test is carried out to see how well the robot planner can handle the real life scenario of navigating through a hallway including a human actor, introduced as a moving obstacle. For the purposes of the test, avoidance and navigation through the hallway is prioritized.

The setup for the test can be seen in Figure 7.13, made such that the robot will have to circumnavigate the moving obstacle.

Figure 7.13: The hallway scenario setup.

The procedure for the test is as follows: The robot has to move through a narrow corridor with a moving obstacle in its path. The robot then has to maneuver out of the obstacles way before they collide. This will be tested with different time steps for the horizon as well as different speeds of the robot and the obstacle. This will be tested for the polygon MPC planner and the PAL planner.

7.4.1 Testing

Testing was carried out with the PAL local planner as well as the Polygon obstacle representation local planner, to see how well a local planner without moving obstacle avoidance compares to one with. For the PAL planner a moving person was moving down the hallway against the robot, while for the polygon MPC a simulated moving obstacle was used. The hallway tests were carried out with obstacle velocities of 0.2m/s, 0.43m/s and 0.6m/s. These values were chosen based on a study of hospital patients, per the scenario of guiding visitors or patients around hospitals. In this study by Graham et al. a mean walking speed was found of 0.43 m/s for ambulatory adults aged 65 and up [58].

7.4.1.1 PAL Local Planner

For comparison the PAL local planner was run through the hallway scenario, with same starting and end point as the developed polygon obstacle MPC planner. The main difference being that the PAL planner simply reacts to obstacles in its vicinity and tries to avoid them, no actual recognition of their movement. The path taken by the runs taken with the Pal planner can be sen in Figure 7.14.

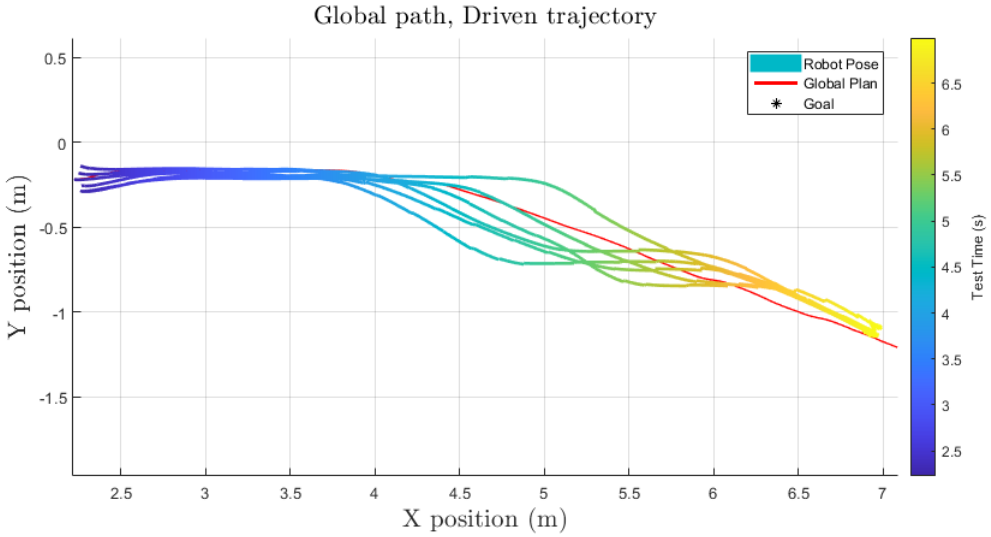


Figure 7.14: The paths resulting from testing the PAL planner.

6 different runs through the hallway were tested, the first one with a person moving through the hallway in the right side and the rest having the person in the left side, right side illustrated in Figure 7.13.
The PAL planner tests were run with a group member walking against the robot, most of the time experiencing the robot colliding with them or getting close enough for them requiring to stop to avoid collision.

7.4.1.2 NMPC Local Planner with Polygon Representation

During the tuning procedure for the hallway scenario the first parameters that were changed was the obstacle size as well as the robot safety boundary. These were changed to make sure the robot could fit besides the obstacle moving within the corridor. Quite some tests were performed with different sizes, ending up with an obstacle of merely 20cm in diameter and a robot safety boundary of 0.05m around the robot. Though it was found that the main reason for early faulty performance was a long prediction horizon, as the obstacle would overlay the current goal of the robot, making the problem of solving for a path infeasible. The size of the obstacle could likely be increased after this discovery, though it was ruled that the robot movement represented the avoidance of the obstacle with these parameters as intended and they were kept.
In Figure 7.15 can be seen an isolated example of how the path travelled compares, red showing the initial global path and blue the actual path taken by the robot, green being the moving obstacle with a radius of 0.2 m and the black circles are the static obstacles seen by the robot.

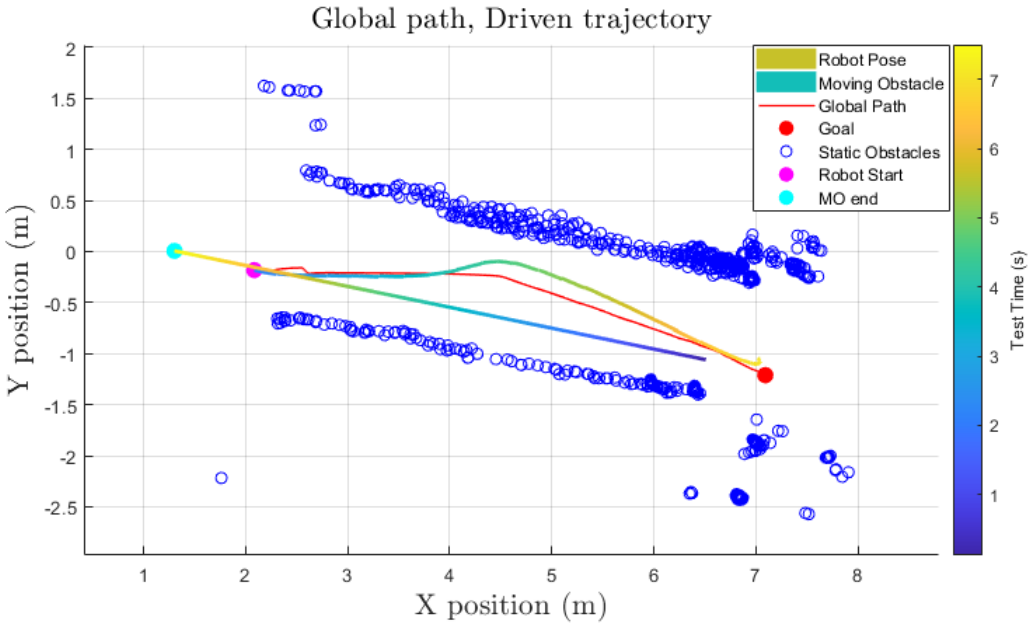


Figure 7.15: The environment and position as seen by the robot, test run at 0.5m/s for Robot velocity, timestep of 0.25 and a horizon of 10 samples

Also compared are the control inputs provided to the system by the MPC, in Figure 7.16 can be seen an example with a maximum robot velocity maxed out at 0.5. On the angular control input can be seen in the spike of negative angular velocity, how the robot turns back unto the global path after initially turning off it to allow the moving obstacle to pass.

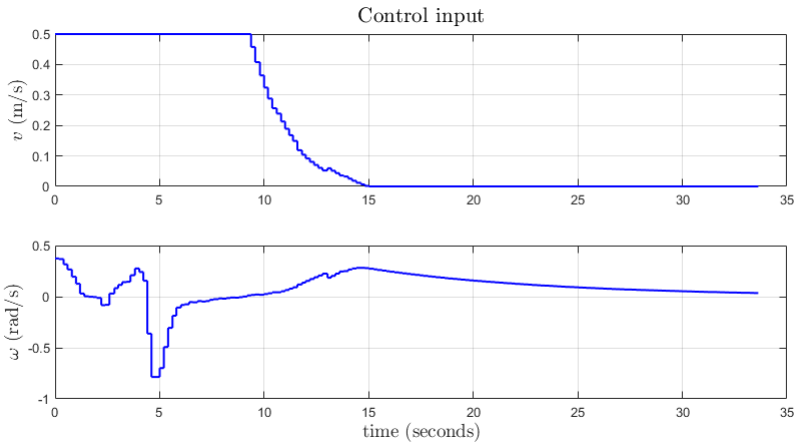


Figure 7.16: Control inputs for the system for the test run at 0.5 Robot velocity, timestep 0.25 and 10 horizon

From the results shown in Appendix B.2 can be seen that the robot travelled distance, as well as the average error to the global plan is reduced with a longer horizon. To visualize

the interaction between the robot moving through the corridor and the moving obstacle, the distance between the two actors is visualized in Figure 7.17.

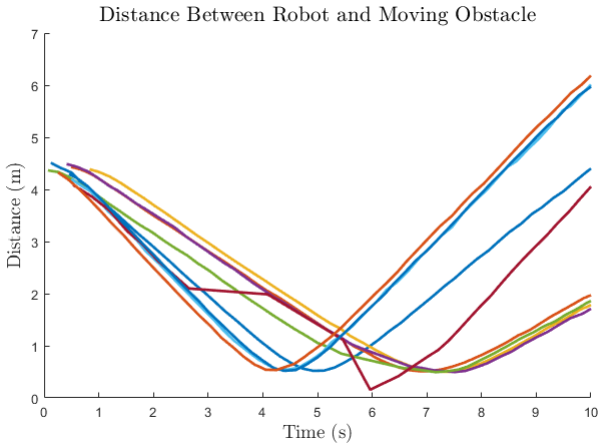


Figure 7.17: Distance to moving obstacle for tests 21 through 29.

It can be seen that in these tests (21-29) the distance minimum holds with the safety boundary set by the robot. The distance being calculated as the length between the center of the robot and the center point of the moving obstacles. From the figure can be seen one of the tests giving a faulty distance measure, upon investigation of the data recorded it can be seen that this is caused by path infeasibility, the robot getting run over by the moving obstacle. This is also shown here [2] The following chapter will describe some of the insights gained through this project and explain some of the further conclusions upon the data gathered from the tests performed.

[2]https://youtu.be/FQIK5868tsQ

CHAPTER 8

DISCUSSION

The main goal of the system is to provide comparable performance to current planners with the added functionality of avoiding moving obstacles. Through a series of tests the system has shown to be capable of avoiding static and moving obstacles given that the NMPC is tuned well. Given the results from the averaged static obstacle testing seen in Table 7.3 the NMPC local planner with polygon obstacle representation have shown to yield comparable results to that of the PAL local planner given similar maximum velocity. Given $v_{max} = 0.75$ for the NMPC local planner and a $v_{max} = 0.8$ for the PAL local planner, the NMPC local planner converges to the goal on average 2.08 seconds faster than the PAL local planner, which is a 9.1% difference. However distance travelled by the NMPC local planner is 0.287 meters longer than that of the PAL local planner, which is a 2.6% difference. The NMPC local planner with closest point to polygon obstacles representation also outperforms the version with polygon centroid obstacle representation on average in time to traverse as well as distance travelled before reaching the goal this can also be seen in Table 7.3. The NMPC local planner with polygon obstacles performed the best in the static obstacle tests with either a maximum velocity of 0.5m/s or 0.75m/s. Above 0.75m/s the robot started to overshoot on the trajectories resulting in undesirable effects where the robot tries to recover from the overshooting. In the majority of static obstacle tests run the average distance to the obstacles have been kept well above the safety boundary of 0.15m to the obstacles. No collisions were encountered during this test, example for each velocity can be seen in Figure 7.8. Although no collision were encountered due to overshooting of the NMPC local planner with polygon obstacles the safety boundary was violated for a brief amount of time as seen in Figure 7.8d.

Two important things to note about the hallway test scenario. Firstly, multiple tests were not carried out using the centroid obstacle representation because it simply did not perform well with walls due to the inherent flaws in the representation when dealing with irregular polygons and lines. Secondly, the robot velocity has been kept constant at 0.5m/s for the hallway scenario, because of the good results yielded from the static obstacle tests. The best results from the hallway scenario resulted from a combination of high resolution horizon $N = 20$ and a time step of $T_s = 0.25$ yielded the best results with an average time to traverse of 13.81 seconds and with no collisions or failures. A significant oversight in the implementation was discovered in that the NMPC local planner has no way to deal

with local goals obscured by moving obstacles, resulting in problem infeasibility. One way of overcoming this problem is to ensure that the local goal will not be inside the moving obstacle or it's predictions it to modify the local goal publisher. However due to time constraints of the Thesis this is not an option. Through tuning this problem can be overcome. Having a horizon that does not predict as far into the future will yield good results as seen with the parameters mentioned earlier.

A moving obstacle test was also conducted in the in Chapter 7. This moving obstacle test was a stress test for the NMPC local planner it was therefore expected that some of the tests would fail. 9 tests were done with the local planner varying the same test parameters as in the hallway scenario and the static obstacle tests. 4 tests succeeded the test while 5 failed the test. The 4 tests that succeeded all had time steps shorter than $T_s = 0.5$ from which test 1 and test 2 were clearly the best. This is an expected result because predicting 0.5 seconds per step in a highly dynamic environment will result in loss of accuracy because a lot can happen in 0.5 seconds in a highly dynamic environment. Similar to the hallway scenario a high resolution of the horizon $N = 20$ whilst keeping the time step T_s relatively low. There are also cases where the horizon and the time step are too low and the robot cannot predict far enough into the future to compute a solution that avoids all the obstacle while converging to the goal this can for example be seen in test 4 with a horizon of $N = 15$ and a time step of $T_s = 0.15$.

During the physical testing some limitations were identified and has to be taken into account when analyzing the data. Firstly, the A^* planner in place on the robot, in the static obstacle test had issues with the map per some of the obstacles not being properly recognized as solid obstacles throughout the testing iterations. This can be observed in Figure 7.6 where the global plan goes through the center wall. This however is a minor problem for the NMPC local planner as it can detect the obstacles once they are within the local costmap bounds. Secondly, Per the implementation of the local planner the current `robot_pose` is taken as the true position of the robot, received by the Adaptive Monte Carlo Localization(AMCL) running on the robot. However this `robot_pose` is only an estimate of the current robot position. To truly test the local planner with an exact position it would be possible to use the Vicon system in the Control and Automation Laboratory in order to pinpoint the robot pose to centimeter or even millimeter precision. And lastly the largest problem with the robot is a mechanical fault causing one of the wheel to not move when control signals are sent to it. This causes the AMCL to completely lose track of the robot and that results in a test failure. Due to this mechanical fault starting to arise multiple times the moving obstacle physical test had to be done in simulation only.

CHAPTER 9

CONCLUSION

The purpose of this Thesis was to investigate whether it was possible to create a local planner using Non-linear Model Predictive Control to predict the motion of moving obstacles and avoid them. Based upon this task, three key questions were formulated in the Problem Statement (Section 2.5). This conclusion will seek to summarize the answers found to these questions, through the research done in this thesis.

The first question was: *How can an NMPC be used as a local planner?* This was investigated through a series of papers and research which used MPC for local planning along with development through a series of local planners designed in MATLAB. Different techniques using MPC for path following and using MPC integrated with RRT* to reach a specific goal were described in the papers and research investigated. In this Thesis, a set of different techniques were employed to broaden the field for MPC used for local planning, specifically local planning with moving obstacle avoidance. The techniques employed in the thesis were, namely, set-point stabilization and trajectory tracking. Through simulations of the NMPC local planners utilizing the techniques mentioned above, it became apparent that implementing the NMPC local planner with trajectory tracking would prove difficult. The difficulties arose with the need for a global planner that could produce reference trajectories with linear and angular velocities at each point in time. Therefore the continued thesis work revolved around the set-point stabilization local planner. During testing the local planner using polygon obstacle representation performed similarly to the native PAL local planner already implemented on the TIAGo robot. This is furthermore verified in Chapter 8 where the NMPC local planner and the PAL local planner with similar parameters had an average difference in time to traverse of 2.08 seconds, which is a 9.1% difference, and the average difference robot path length of 0.2871 meters, which is a 2.6% difference, in the static obstacle tests.

The next key question formulated was: *How can an NMPC local planner be made to consider static and moving obstacles?* This question can be split into two answers, namely considerations for moving obstacle and considerations for static obstacles. Firstly, moving obstacles were investigated specifically in relation to Human-Robot Interaction (HRI). Many of the papers investigated used costmap-like models for representing humans. These models were either used for representing the uncertainty of a prediction and/or a position or for representing a social zone with cost associated with distance. Due to time limitations

of the Thesis, a more straightforward representation with predictive capabilities has been developed throughout the Thesis. This representation defines humans as points with a radius, heading and velocity moving through time. These obstacles are then formulated in the NMPC as time-varying constraints. Using this method, the distance between the robot and the humans was defined by a constant safety boundary S_b. This method of representing moving obstacles has been tested both in pure simulation and in real-life with virtual obstacles, specifically the moving obstacle test in Gazebo and the physical hallway scenario. Using this representation of moving obstacles the robot was able to move through an environment with 8 moving obstacles without collision given the right tuning parameters. Improvements to the moving obstacle representation are discussed in Section 9.1. Secondly, static obstacle representations have been investigated. In MATLAB, the local planner had been designed with two different representations, namely a centroid obstacle representation, similar to the moving obstacles with a point and a radius, and a polygon representation that considers the distance from the closest point on a polygon to robot. Both of these techniques required polygons as initial input, so a package named `costmap_converter` was used to transform the obstacles in the costmap into polygons that could then be used in the NMPC formulation of the static obstacles. These two representations have been tested and compared both in simulation and in real life. It can be concluded based upon the results analyzed in Chapter 8 that the local planner with the polygon representation is faster and more consistent than the centroid representation in terms of time to traverse a path as seen in Table 7.3.

Another question sought to be answered was: *How can an NMPC local planner be implemented on the robot as a part of a navigation package?* The NMPC local planner developed in this Thesis was integrated into a ROS package that has then been deployed on the TIAGo robot. The ROS package was implemented outside the `/move_base` package because of the possibility to develop the local planner in Python. Even though the local planner has not been implemented as a `C++` plugin in the `/move_base`, it still performed well in the test scenarios described in Chapter 7 with an update frequency of 5Hz. In these tests, local planner frequency was not a limiting factor due to the robot and the moving obstacles moving at relatively low speeds between 0.3m/s to 1m/s. For testing and experimentation, this local planner frequency is sufficient; however, in a finished product, it would be desirable to have a higher local planner frequency. This is discussed further in Section 9.1.

To summarize the three questions: It was possible to formulate the local planner using NMPC with various obstacle representations. The results from the local planner was comparable to the results of the native PAL local planner on the robot. The moving obstacle representation has worked well in both simulation and real-life tests if the NMPC was tuned well. Furthermore it was possible to implement the navigation package into the

existing TIAGo navigation stack with relative ease. And thus this concludes the overall remarks of the Thesis.

9.1 Future Work

To improve upon the NMPC based local planner developed in this project some key points are proposed:

- Currently all obstacles only considered as hard constraints in the NMPC. By modifying the const function by adding an extra term that penalizes the close distances to the obstacle would result a much more stable control system
- The method of keeping a certain distance between the robot and humans lends itself to adding a cost to this distance, effectively creating a costmap-like model defining the moving obstacles. This weighting could as well be extended to shape the moving obstacle costmap per the facing direction and velocity of the person.
- Considering costmaps for individual people in an environment could be expanded upon through a proximity based obstacle re-representation as social zones, increasing the comfort with which the robot can navigate in populated environments.
- The `costmap_converter` plugin has an experimental algorithm for moving obstacle tracking and prediction based on the changes of the costmap that could be implemented.
- The proposed NMPC design can be further improved by various well known control methods. For example considering terminal constraints by adding a Mayer term to the optimal control problem would further guarantee stability. An other improvement would be to consider parts of calculated optimal control sequences instead of applying only $\boldsymbol{u}^{\star}(0)$ in those cases when the OCP runs into an infeasiblity problem.
- Several researches show that IPOPT can be tackled by other NLP solvers considering non-linear system model
- Required processing power could be reduced significantly by conversion of the MPC script to a more efficient language such as C++, possibly even generated by the use of `CasADi` code generation, creating specific code capable of calculating the optimization of NLP problems more efficiently. Auto-generated code compiled with code optimization flags by `CasADi` can reach 4 to 10 times faster code execution than `CasADi` virtual machines.

BIBLIOGRAPHY

[1] Thomas Jespersen. Kugle - modelling and control of a ball-balancing robot. ,, 04 2019. doi: 10.13140/RG.2.2.31490.73928. URL https://projekter.aau.dk/projekter/en/studentthesis/kugle--modelling-and-control-of-a-ballbalancing-robot(4450f15c-7d96-41c6-b436-af2296e1bc01).html.

[2] Kerstin Dautenhahn. Methodology & themes of human-robot interaction: A growing research field. *International Journal of Advanced Robotic Systems*, 4(1):15, 2007. URL https://journals.sagepub.com/doi/pdf/10.5772/5702.

[3] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013. doi: 10.1016/j.robot.2013.05.007. URL https://www.sciencedirect.com/science/article/pii/S0921889013001048.

[4] Edward Twitchell Hall. *The hidden dimension*, volume 609. Garden City, NY: Doubleday, 1966. URL https://www.academia.edu/5668023/Edward_T._Hall-_The_Hidden_Dimension.

[5] Trung-Dung Ngo Xuan-Tung Truong. Dynamic social zone based mobile robot navigation for human comfortable safety in social environments. *Internal Journal of Social Robotics*, 8:663–684, 2016. doi: 10.1007/s12369-016-0352-0. URL https://link.springer.com/article/10.1007%2Fs12369-016-0352-0.

[6] J. Gaa M. Kollmitz, K. Hsiao and W. Burgard. Time dependent planning on a layered social cost map for human-aware robot navigation. *2015 European Conference on Mobile Robots (ECMR)*, pages pp. 1–6, 2015. doi: 10.1109/ECMR.2015.7324184. URL https://ieeexplore.ieee.org/document/7324184.

[7] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979. URL https://lis.csail.mit.edu/pubs/tlp/collision-free-planning-cacm.pdf.

[8] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. IEEE, 1985. URL https://ieeexplore.ieee.org/document/1087247.

[9] Bruce Krogh. A generalized potential field approach to obstacle avoidance control. In *Proc. SME Conf. on Robotics Research: The Next Five Years and Beyond, Bethlehem, PA, 1984*, pages 11–22, 1984.

[10] C Thorpe and L Matthies. Path relaxation: Path planning for a mobile robot. In *OCEANS 1984*, pages 576–581. IEEE, 1984. URL https://ieeexplore.ieee.org/document/1152243.

[11] Bruce Krogh and Charles Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1664–1669. IEEE, 1986. URL https://ieeexplore.ieee.org/abstract/document/1087444.

[12] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5):1179–1187, 1989. URL https://ieeexplore.ieee.org/abstract/document/44033.

[13] Yoram Koren, Johann Borenstein, et al. Potential field methods and their inherent limitations for mobile robot navigation. In *ICRA*, volume 2, pages 1398–1404, 1991. URL https://ieeexplore.ieee.org/document/131810.

[14] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3): 278–288, 1991. URL https://ieeexplore.ieee.org/document/88137.

[15] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146)*, volume 2, pages 1572–1577. IEEE, 1998. URL http://www-personal.umich.edu/~johannb/Papers/paper73.pdf.

[16] Iwan Ulrich and Johann Borenstein. Vfh/sup*: Local obstacle avoidance with look-ahead verification. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2505–2511. IEEE, 2000. URL https://ieeexplore.ieee.org/document/846405.

[17] Shuzhi Sam Ge and Yan Juan Cui. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, 16(5):615–620, 2000. URL https://ieeexplore.ieee.org/document/880813.

[18] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002. doi: 10.1023/A:1020564024509. URL https://link.springer.com/article/10.1023/A:1020564024509.

[19] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of IEEE international conference on robotics and automation*, volume 4, pages 3375–3382. IEEE, 1996. URL https://ieeexplore.ieee.org/abstract/document/511023.

[20] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997. URL https://ieeexplore.ieee.org/document/580977.

[21] Thibault Kruse, Patrizia Basili, Stefan Glasauer, and Alexandra Kirsch. Legible robot navigation in the proximity of moving humans. In *2012 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 83–88. IEEE, 2012.

[22] Rachel Kirby, Reid Simmons, and Jodi Forlizzi. Companion: A constraint-optimizing method for person-acceptable navigation. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 607–612. IEEE, 2009. URL https://ieeexplore.ieee.org/document/5326271.

[23] Emrah Akin Sisbot, Luis F Marin-Urias, Rachid Alami, and Thierry Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5): 874–883, 2007. URL https://ieeexplore.ieee.org/document/4339546.

[24] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998. URL https://journals.sagepub.com/doi/pdf/10.1177/027836499801700706.

[25] Kikuo Fujimura and Hanan Samet. A hierarchical strategy for path planning among moving obstacles (mobile robot). *IEEE transactions on robotics and Automation*, 5 (1):61–69, 1989.

[26] A Tsoularis and Chandra Kambhampati. On-line planning for collision avoidance on the nominal path. *Journal of Intelligent and Robotic Systems*, 21 (4):327–371, 1998. URL https://link.springer.com/content/pdf/10.1023/A:1007919920684.pdf.

[27] Animesh Chakravarthy and Debasish Ghose. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(5):562–574, 1998. URL https://ieeexplore.ieee.org/document/709600.

[28] Lars Grüne and Jürgen Pannek. Stability and suboptimality without stabilizing constraints. ,, pages 113–163, 2011. ISSN 0178-5354. doi: 10.1007/978-0-85729-501-9_6.

[29] Rolf Findeisen, Lars Imsland, Frank Allgower, and Bjarne A Foss. State and output feedback nonlinear model predictive control: An overview. *European journal of control*, 9(2-3):190–206, 2003.

[30] Timm Faulwasser. Optimization-based solutions to constrained trajectory-tracking and path-following problems. ,, 2012.

[31] Rolf Findeisen. Nonlinear model predictive control: a sampled data feedback perspective. ,, 2005.

[32] MM Kale and AJ Chipperfield. Stabilized mpc formulations for robust reconfigurable flight control. *Control Engineering Practice*, 13(6):771–788, 2005.

[33] James B Rawlings, Nishith R Patel, Michael J Risbeck, Christos T Maravelias, Michael J Wenzel, and Robert D Turney. Economic mpc and real-time decision making with application to large-scale hvac energy systems. *Computers & Chemical Engineering*, 114:89–98, 2018.

[34] Daniel Limón, Teodoro Alamo, Francisco Salas, and Eduardo F Camacho. On the stability of constrained mpc without terminal constraint. *IEEE transactions on automatic control*, 51(5):832–836, 2006.

[35] Mohsen Ahmadi Mousavi, Zainabolhoda Heshmati, and Behzad Moshiri. Ltv-mpc based path planning of an autonomous vehicle via convex optimization. In *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, pages 1–7. IEEE, 2013.

[36] Lieboud Van den Broeck. Time optimal control of mechatronic systems through embedded optimization (tijdsoptimale controle van mechatronische systemen door middel van ingebedde optimalisatie). ,, 2011.

[37] K. Sekiguchi Y. Nishio, K. Nonaka. Moving obstacle avoidance control by fuzzy potential method and model predictive control. *2017 11th Asian Control Conference (ASCC)*, pages 1298–1303, 2017. doi: 10.1109/ASCC.2017.8287358. URL `https://ieeexplore.ieee.org/document/8287358`.

[38] Daniele De Simone, Nicola Scianca, Paolo Ferrari, Leonardo Lanari, and Giuseppe Oriolo. Mpc-based humanoid pursuit-evasion in the presence of obstacles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5245–5250. IEEE, 2017.

[39] Dongbing Gu and Huosheng Hu. Receding horizon tracking control of wheeled mobile robots. *IEEE Transactions on control systems technology*, 14(4):743–749, 2006.

[40] Hongyan Guo, Chen Shen, Hui Zhang, Hong Chen, and Rui Jia. Simultaneous trajectory planning and tracking using an mpc method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle. *IEEE Transactions on Industrial Informatics*, 14(9):4273–4283, 2018.

[41] Kiattisin Kanjanawanishkul and Andreas Zell. Path following for an omnidirectional mobile robot based on model predictive control. In *2009 IEEE International Conference on Robotics and Automation*, pages 3341–3346. IEEE, 2009.

[42] Steven M. LaValle. Differential models, 2011. URL `http://planning.cs.uiuc.edu/ch13.pdf`.

[43] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In ,, pages 226–231. AAAI Press, 1996.

[44] S. Y. Shin and T. C. Woo. Finding the convex hull of a simple polygon in linear time. *Pergamon JournaLs Ltd. Pattern Recognition Societ*, 19(6):453–458, 1985. URL `https://deepblue.lib.umich.edu/bitstream/handle/2027.42/26420/0000507.pdf;jsessionid=CE7FF57F5D36AFC5FDF5A918CDD9CA9E?sequence=1`.

[45] Mahes Visvalingam and James Duncan Whyatt. The douglas-peucker algorithm for line simplification: Re-evaluation through visualization. *Comput. Graph. Forum*, 9: 213–228, 1990.

[46] Elmar de Koning. Douglas-peucker, 2011. URL `http://psimpl.sourceforge.net/douglas-peucker.html`.

[47] Carsten Steger. On the calculation of moments of polygons. Technical report, Technical Report FGBV–96–04, Forschungsgruppe Bildverstehen (FG BV ..., 1996.

[48] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

[49] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. Trajectory planning for robots in dynamic human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4293–4298. IEEE, 2010.

[50] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[51] LL Lynn, Elliot S Parkin, and Raymond L Zahradnik. Near-optimal control by trajectory approximation. tubular reactors with axial dispersion. *Industrial & Engineering Chemistry Fundamentals*, 9(1):58–63, 1970.

[52] Hans Georg Bock, E Eich, and Johannes P Schlöder. *Numerical solution of constrained least squares boundary value problems in differential-algebraic equations*. Sonderforschungsber., Univ., 1987.

[53] Tully Foote. tf: The transform library. *Open Source Robotics Foundation*, 2013. URL `http://wiki.ros.org/Papers/TePRA2013_Foote?action=AttachFile&do=get&target=TePRA2013_Foote.pdf`.

[54] Simon N. Callisen Martin B. Jensen and Balazs Reiser. Github repository, 2020. URL `https://github.com/aau-hamr-lab/TIAGo-hamr-navigation`.

[55] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.

[56] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[57] Martin B. Jensen Simon N. Callisen, Balázs Reiser. So polygon test 2020 06 03 16 01 38, 2020. URL `https://www.youtube.com/watch?v=9TVRQ0NPsaw&feature=youtu.be`.

[58] James E Graham, Steve R Fisher, Ivonne-Marie Bergés, Yong-Fang Kuo, and Glenn V Ostir. Walking speed threshold for classifying walking independence in hospitalized older adults. *Physical therapy*, 90(11):1591–1597, 2010. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2967707/`.

LIST OF FIGURES

LIST OF TABLES

APPENDIX A

SKLANSKY'S MODIFIED ALGORITHM

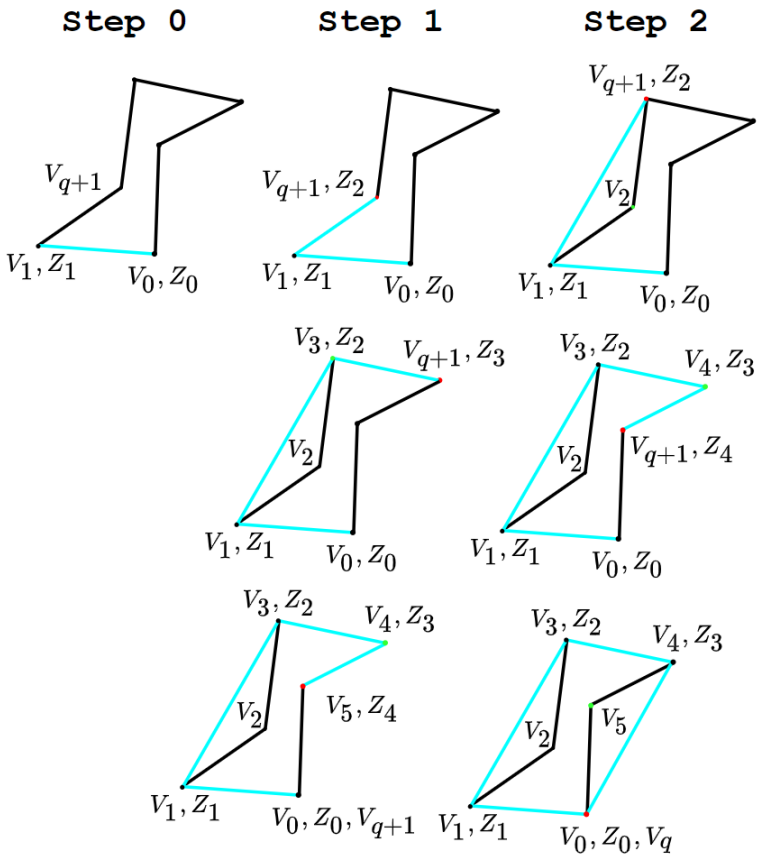


Figure A.1: An illustration of Sklansky's algorithm with a small set of points.

APPENDIX B
TESTING RESULTS

B.1 Static Obstacle Stress Test Results

Testnr. & Controller	N	Ts	Robot Velocity	Global Plan Length	Robot path Length	Robot path - global path error	Average Distance to global plan	Time to Traverse
PalNav: 1				10.5370	11.0614	0.5244	0.1171	24.3926
2				10.1998	11.0741	0.8743	0.0921	27.9182
3				10.1474	10.7078	0.5604	0.0645	17.9874
4				10.1224	10.6736	0.5511	0.1134	19.3225
5				10.0745	10.5281	0.4536	0.0661	16.7914
6				10.1134	10.5638	0.4504	0.1426	21.3927
7				10.1232	10.5915	0.4682	0.1226	20.6902
8				10.1564	11.5000	1.3437	0.1077	48.0845
9				10.2483	10.6697	0.4215	0.1246	17.0992
10				10.1294	10.7390	0.6097	0.0961	17.5785
11				10.0803	10.7700	0.6897	0.0913	18.9061
12				10.2976	11.0840	0.7864	0.1092	31.5026
13				10.3030	11.7861	1.4831	0.3728	28.7985
Centroid: 1		0.1	0.5	10.2066	17.5470	7.3404	0.2566	67.6948
2		0.1	0.3	10.1813	15.1889	5.0075	0.2045	67.9525
3		0.25	0.3	10.1100	12.4853	2.3752	0.2024	46.7747
4		0.25	0.3	10.2360	14.3500	4.1140	0.2143	65.8876
5		0.25	0.3	10.2104	11.0586	0.8482	0.1367	40.8943
6		0.25	0.3	10.4247	12.2136	1.7889	0.2501	51.9147
7		0.25	0.5	10.1933	12.4948	2.3014	0.2296	34.2175
8		0.25	0.5	10.2318	22.8067	12.5749	1.0290	78.7243
9		0.25	0.5	10.2546	11.5743	1.3197	0.1748	41.0586
10		0.25	0.5	10.2882	12.1376	1.8494	0.1271	24.0349
11		0.25	0.5	10.1709	10.7169	0.5460	0.1042	25.5318
12		0.25	0.5	10.2265	12.2356	2.0091	0.1755	38.3363
13		0.25	0.5	10.1911	10.9670	0.7759	0.1069	23.1690
14		0.25	0.5	10.2527	11.3517	1.0990	0.2415	33.7907
15		0.25	0.5	10.2251	10.8939	0.6688	0.1282	25.5398
16		0.25	0.5	10.2482	10.9384	0.6902	0.1673	25.4372
17		0.25	0.5	10.3489	10.8894	0.5406	0.1147	23.3574
18		0.25	0.5	10.2430	11.9786	1.7355	0.1676	36.0426
Polygon: 1	0.3	0.1	20	10.2999	11.0298	0.7299	0.2568	47.2049
2	0.3	0.1	20	10.2517	11.7422	1.4905	0.1348	101.2029

Testnr. & Controller	N	Ts	Robot Velocity	Global Plan Length	Robot path Length	Robot path - global path error	Average Distance to global plan	Time to Traverse
3	0.3	0.1	20	10.2948	10.8917	0.5969	0.1875	41.9858
4	0.3	0.1	20	10.2500	11.1129	0.8629	0.1662	71.5060
5	0.3	0.1	20	10.2567	11.2028	0.9461	0.1931	43.2233
6	0.3	0.1	20	10.2073	11.5847	1.3774	0.1586	73.0834
7	0.3	0.1	20	10.2803	10.7984	0.5182	0.1582	48.1163
8	0.3	0.1	20	9.8330	10.8512	1.0182	0.1504	40.6808
9	0.3	0.1	20	10.2625	8.3560	-1.9065	0.3943	56.3305
10	0.3	0.1	20	10.3434	11.1130	0.7696	0.1701	51.3113
11	0.3	0.1	20	10.2952	11.4918	1.1966	0.1961	51.7564
12	0.3	0.1	20	10.2365	11.0678	0.8313	0.1812	47.7132
13	0.3	0.1	20	10.2517	11.1450	0.8933	0.2136	57.2724
14	0.3	0.1	20	10.2759	11.0346	0.7587	0.1116	111.1375
15	0.3	0.1	20	10.1436	10.1805	0.0369	0.2223	44.3225
16	0.3	0.1	20	10.2672	11.3259	1.0587	0.1197	42.0462
17	0.3	0.1	20	10.2440	10.8771	0.6332	0.2636	40.6506
18	0.3	0.25	20	10.2350	10.9429	0.7079	0.1651	49.0303
19	0.3	0.25	20	10.3089	10.7437	0.4348	0.2404	40.8850
20	0.3	0.25	20	10.1540	10.3543	0.2003	0.1282	36.6189
21	0.3	0.25	20	10.2872	10.7547	0.4675	0.1497	42.9196
22	0.3	0.25	20	10.1561	10.7081	0.5520	0.1111	39.0527
23	0.3	0.25	20	10.2827	10.5264	0.2437	0.1956	39.3234
24	0.3	0.25	20	10.1559	10.8748	0.7189	0.1256	49.2788
25	0.3	0.25	20	10.2520	10.8820	0.6300	0.1843	42.6407
26	0.3	0.25	20	10.1238	10.8781	0.7543	0.1170	38.7929
27	0.3	0.25	20	10.2691	10.8233	0.5541	0.2442	41.1616
28	0.3	0.5	20	10.1545	10.8784	0.7239	0.1290	42.8958
29	0.3	0.5	20	10.2777	10.8373	0.5595	0.1888	43.3825
30	0.3	0.5	20	10.1410	10.7657	0.6247	0.1021	37.9787
31	0.3	0.5	20	10.2836	10.9113	0.6276	0.1471	46.5883
32	0.5	0.5	20	9.8403	10.5173	0.6770	0.1605	25.1127
33	0.75	0.5	20	10.2303	11.3676	1.1373	0.0819	20.5067
34	0.75	0.5	20	10.3203	10.9542	0.6339	0.1212	23.0893
35	1	0.5	20	10.3366	12.9108	2.5742	0.0983	48.7619
36	1	0.5	20	10.3368	10.8690	0.5322	0.1120	32.7033

Table B.1: Data Gathered from running the static obstacle stress test.

B.2 Hallway Test

Testnr. & Controller	N	Ts	Robot Velocity	Obstacle Velocity	Global Plan Length	Robot path Length	Average Distance to global plan	Time to Traverse	Minimum Distance To Moving Obst. / Collision	
PalNav:1	N/A		0.8	N/A	4.985755	5.273188	0.036368	10.006123		
2			0.8		5.006918	5.084404	0.040659	9.798106		
3			0.8		5.011056	5.211132	0.016125	10.406404		
4			0.8		5.049714	5.296439	0.035529	12.990999		
5			0.8		5.012859	5.325864	0.032475	11.377043		
6			0.8		5.014061	5.149848	0.029393	9.216872		
Polygon: 1	0.5	0.25	20	0.43	5.167372	5.852483	0.052619	14.698828	0.4838	YES
2	0.5	0.5	20	0.43	5.208394	6.062019	0.375842	3.624748	0.2413	YES
3	0.5	0.5	20	0.43	5.223907	11.031980	0.023860	14.591687	0.4945	YES
4	0.5	0.5	10	0.43	5.094775	10.860470	0.062871	14.402799	0.4903	YES
5					5.216681	5.515018	0.041564	17.179003	0.5158	NO
6	0.5	0.25	10	0.43	5.213243	5.642186	0.080274	15.245204	0.5195	NO
7	0.5	0.25	10	0.2	5.074556	5.328053	0.039735	14.197330	0.5101	NO
8	0.5	0.5	10	0.2	5.033884	5.351431	0.057048	18.200160	0.5007	NO
9	0.5	0.5	20	0.2	5.173414	5.783427	0.060030	15.009254	0.4966	YES
10	0.5	0.25	20	0.2	5.072094	5.994115	0.099257	14.185422	0.4944	YES
11	0.5	0.25	20	0.6	5.109662	5.545254	0.043849	13.804583	0.5449	NO
12	0.5	0.5	20	0.6	5.197066	5.651408	0.040176	25.177992	0.1559	YES
13	0.5	0.5	10	0.6	5.081707	5.640632	0.075551	14.960675	0.5201	NO
14	0.5	0.25	10	0.6	5.095165	5.824872	0.136024	14.385966	0.5352	NO
15	0.5	0.5	20	0.6	5.193369	3.959708	0.166189	21.580598	0.5172	NO
16	0.5	0.5	20	0.6	5.173404	1.668924	0.085326	15.220656	0.7512	NO
17	0.5	0.5	20	0.6	5.054360	2.234901	0.098995	7.308320	0.0799	YES
18	0.5	0.5	20	0.6	5.057601	8.811474	0.708745	14.243881	11.7282	NO
19	0.5	0.5	20	0.6	5.287672	2.841163	0.081149	7.119133	0.1686	YES
20	0.5	0.25	20	0.6	5.702262	6.167772	0.163853	12.606640	0.5077	NO
21	0.5	0.5	10	0.6	5.085177	5.138287	0.128437	14.217596	0.4864	YES
22	0.5	0.5	15	0.6	4.780771	2.327948	0.112024	5.909939	0.0882	YES

Table B.2: Hallway Test Data

APPENDIX C

TESTING PROCEDURES

The procedures carried out for establishing connection to the robot, software deployment and test procedures for this project.

C.1 General Procedures and Information

The first step, universal for all the tests, is to turn on and connect to the TIAGo robot. First the robot must be turned at which the on/off button will light up in a pushed down position. The system components are then turned on by the electric switch which will light up green when on and the display will turn on. The user interface for this procedure can be seen in Figure C.1. Before using the robot the emergency stop button needs to be in the free position (not pushed down).

Figure 4: User panel

Number	Name / Short description
1	Emergency Stop
2	Information Display
3	On/Off button
4	Electric switch

Figure C.1: This is the user panel of the TIAGo robot.

The startup procedure as described prior is the same for both the PAL TIAGo moving base and TIAGo Steel, except for an initial setup procedure for the Steel of moving the torso, head and gripper around a bit. In contrast the moving base simply has LED's along its sides that light up and blink once when it is ready for operation. For the operation of the robots a joystick is associated, one should click the blue X button(2) on the right side of the controller to shift priority from on-board control to the joystick, pressing the start button(1) might also be necessary if priority is transferred from a fresh startup. After this the joysticks (4 and 5) should be enabled for use to maneuver the robot base.

Figure C.2: The controller layout while testing

After the startup the robot is available for establishing connectivity, for this either an ethernet connection can be established or connection to the on-robot Wi-Fi network. In the case of the moving base the Wi-Fi name should be pmb2-1. After connection is established a simple ssh connection can be launched by the command 'ssh root@pmb2-33c' the with the password 'palroot'. The system by default has 2 users; pal and root with the passwords pal and palroot respectively At this point any ROS packages should be easily deployable within the ROS environment on the robot, using the pal_deploy package

C.2 Static Obstacle Avoidance

C.2.1 Real-life test setup

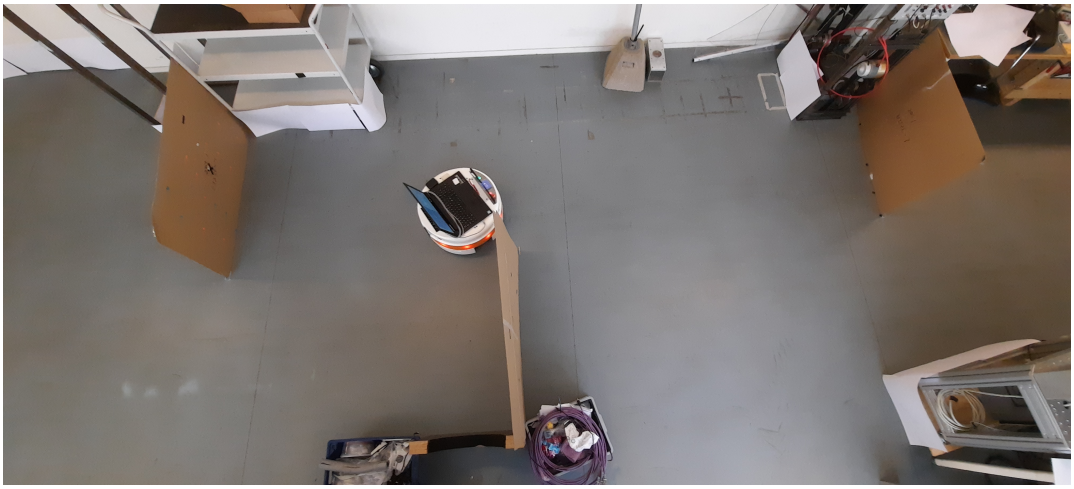


Figure C.3: The test setup for the static obstacle avoidance.

C.2.2 Test procedure

For testing upon the robot all the following nodes directly on the robot after having deployed necessary packages and copied over script files. For the test we have two ROS packages running, the local goal received from pathlist and the obstacle data received from costmap_converter_publisher

```
rosrun pathsplit pathlist
rosrun costmap_converter_publisher publish_costmap_conversion
```

The MPC local planner is then run, for the polygon and centroid obstacle representations they each have a separate script. They are respectively:

```
python CasadiMPC_Polygonavoidance.py
python CasadiMPC_Centroidavoidance.py
```

The goal to the robot should be published separately using rostopic publish command:

```
rostopic pub -r 10 /move_base_simple/goal geometry_msgs/PoseStamped
"header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0"
```

With the position fields specific to the given goal position desired for the robot. At this point the MPC will be running, waiting for the moving obstacles to be declared, the moving obstacle publisher has been created such that it starts setting up the topic for the moving obstacles and after another input defines where they are. Therefore the recording of test data should be launched after the moving obstacle publisher is started but before the obstacles have been defined. The obstacle publisher is run by:

```
python MO_publisher.py
```

After which a prompt appears for user input, now is a great time to set up recording of ROS topics since all the topics used in implementation have been created. With recording simply performed by:

```
rosbag record -a
```

Recording all the topics currently running within the ROS structure on the robot. Now all that is left is to start the MO_publisher, by inputting 'Q' it is possible to define new points through the 'publish points' option in RViZ for the `MO_publisher.py` script, 4 of which are required, after which the robot should move on its own to the given position.

C.3 Moving Obstacle Avoidance

This section will describe the test procedure for the moving obstacles avoidance test.

C.3.1 Test procedure

Since this test has be made purely in simulation the Gazebo simulation has to be run with the correct world that will be used for this test. This is done with the following roslaunch file is run with some certain parameters defined:

```
roslaunch tiago_2dnav_gazebo tiago_navigation.launch public_sim:=true
world:="simple_office"
```

For the test we have two ROS packages created in this project, the `pathsplit` that gives the local goal to the robot from the global plan and the `costmap_converter_publisher` that publishes all the obstacles from the `costmap_converter`:

```
rosrun pathsplit pathlist
rosrun costmap_converter_publisher publish_costmap_conversion
```

After these packages has been run the Moving obstacle publisher is run with the obstacle parameters saved in a numpy array that can be used for each test.

```
python MO_publisher.py
```

Then the MPC local planner is run. There is two different scripts for the two static obstacles representations. For the polygon representation:

```
python CasadiMPC_Polygonavoidance.py
```

or for the circular obstacle representation:

```
python CasadiMPC_Centroidavoidance.py
```

Once all the packages for the local planner has be run, the rosbagging is used to save all the data posted to the relevant topics. The following command is run in order to do this:

```
rosbag record -a -x "/xtion/(.*)|/gains/(.*)"
```

This command record all topics except the topics related to the xtion camera and the robot arm gains since this data is redundant to this thesis. Once the logging of the topics has begun the global goal has to be published for the robot to start moving toward the goal.

```
rostopic pub -r 10 /move_base_simple/goal geometry_msgs/PoseStamped
"header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 4.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0"
```

The tests have simply been recorded until the robot has reached the goal published. Once this is done all process/nodes are terminated and the data is saved in a rosbag file for later analysis. Between each test the parameters in the MPC have been adjusted in the order seen in Table 7.4.

C.4 Hallway Scenario

This test is carried out to test the capabilities for moving obstacle avoidance in a real life scenario.

C.4.1 Real-life test setup

Here in Figure C.4 can be seen the testing environment for the hallway with moving obstacle test.

Figure C.4: The test setup for the hallway scenario.

C.4.2 Testing Procedure

Firstly an SSH connection should be established upon the robot, and the `pathsplit` and `costmap_converter_publisher` package should be deployed unto the robot by using the deploy `pal_deploy` package. The deployment being done from the development computer, having an established connection to the robot platform.

```
rosrun pal_deploy deploy.py --user root pmb2-33c
```

Along with these then the MPC script file `CasadiMPC_Polygonavoidance.py` should be copied unto the robot along with the moving obstacle publisher `MO_publisher.py`.
From here on out the procedure is the same as for the static obstacle avoidance scenario. An extension upon it can be implemented by using the "L" key prompt to load a previously saved set of initial positions for the moving obstacles. After which the user should input the timestamp upon the saved MO_Matrix file in the same folder it is run from, to achieve similar obstacles to last test iteration.

APPENDIX D

LEARNING OBJECTIVES

Knowledge

- have knowledge, at the highest international level of research, of at least one of the core fields of the education
- have comprehension of implications of research (research ethics)

Skills

- are able to reflect on a scientific basis on their knowledge,
- can argue for the relevance of the chosen problem to the education including specifically account for the core of the problem and the technical connections in which it appears
- can account for possible methods to solve the problem statements of the project, describe and assess the applicability of the chosen method including account for the chosen delimitation and the way these will influence on the results of the product
- can analyze and describe the chosen problem applying relevant theories, methods and experimental data
- are able to describe the relevant theories and methods in a way that highlights the characteristics and hereby document knowledge of the applied theories, methods, possibilities and delimitations within the relevant problem area
- have the ability to analyze and assess experimental data, including the effect the assessment method has on the validity of the results.

Competences

- are able to communicate scientific problems in writing and orally to specialist and non-specialist.
- are able to control situations that are complex, unpredictable and which require new solutions,

- are able to independently initiate and to perform collaboration within the discipline and interdisciplinary as well, and to take professional responsibility,
- are able to independently take responsibility for his or her own professional development and specialization.