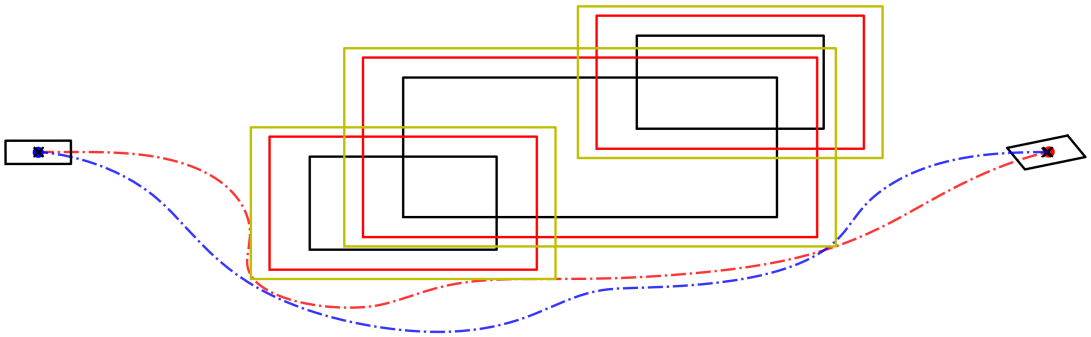


Trajectory Planning for a Fleet of Autonomous Transport Robots

Max Edin Jakobsson
Muhamed Faraj

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Trajectory Planning for a Fleet of Autonomous Transport Robots

Max Edin Jakobsson
Muhamed Faraj

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Trajectory Planning for a Fleet of Autonomous Transport Robots
MAX EDIN JAKOBSSON
MUHAMED FARAJ

Supervisor: Knut Åkesson, Co-Supervisor: Albin Dahlin
Examiner: Knut Åkesson, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Trajectories for two autonomous transport robots avoiding collisions between each other and the obstacles.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Trajectory Planning for a Fleet of Autonomous Transport Robots
MAX EDIN JAKOBSSON
MUHAMED FARAJ
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis proposes a solution for generating trajectories for a fleet of autonomous transport robots (ATRs) operating in a factory environment. The trajectories have to follow a predetermined path while avoiding both static obstacles and moving obstacles such as other robots, humans, walls, pallets etc.

To generate the trajectories, a Distributed Non-Linear Model Predictive Control (DNMPC) is used, and the solver used is a Proximal Averaged Newton-type method for Optimal Control (PANOC) solver as it is proven to be computationally faster than other solver alternatives. A DNMPC calculates the trajectories for each robot independently, but information of the other robots locations are shared. The obstacles are modelled as polygons, ellipses or circles which can be represented as inequalities so that the DNMPC can handle them as soft and hard constraints.

Additionally, a method for generating new paths around static obstacles is implemented as obstacles might be appearing after the path has been set. This is done to help the controller generate trajectories leading to the ATRs goal positions. Furthermore, an algorithm for detecting collisions based on the robots predicted states is implemented that has the ability to overrule the DNMPC as there is no guarantee that the solver will converge to a collision free trajectory.

The DNMPC, method for generating paths and collision detection are all evaluated in simulations of various scenarios. From the results it is shown that the ATRs manages to avoid obstacles and reach their goal position in most of the cases. Additionally the computation times for the solver and added algorithms are low enough and show promise for real-time implementations but since the solution wasn't tested in a real scenario there might exist issues coming from noisy data and time delays resulting in sub optimal performance.

Keywords: NMPC, PANOC, Trajectory Planning, Obstacle Avoidance.

Acknowledgements

We would like to thank our examiner and supervisor Knut Åkesson and co-supervisor Albin Dahlin, both from Chalmers. They have always been available and with their help we have been able to overcome various problems and keep the research moving forward. We would also like to thank both Per-Lage Götvall and Erik Brorsson from Volvo Group for their commitments to this project and their help over at Volvo Group.

Max Edin Jakobsson, Gothenburg, Jun 2022
Muhamed Faraj, Gothenburg, Jun 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ATR	Autonomous Transport Robot
DNMPC	Distributed Non-Linear Model Predictive Control
MPC	Model Predictive Control
NMPC	Non-Linear Model Predictive Control
PANOC	Proximal Averaged Newton-type method for Optimal Control
UAV	Unmanned Aerial Vehicles

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

k	Index for which time step
j	Index for which ATR

Sets

$\mathcal{O}$	Set of obstacles
$\mathcal{N}$	Set of positive integers

Parameters

Δt	Time discretization step (time interval)
N	MPC horizon
Q_O	Weight for colliding with static obstacles
Q_B	Weight for being outside boundaries
Q_{dyn}	Weight for colliding with dynamic obstacles
Q_{atr}	Weight for colliding with other robots
Q_{acc}	Weight on acceleration of an ATR
Q_τ	Weight on tracking error for an ATR
Q_s	Weight on deciding tracking error point

Variables

$\boldsymbol{x}_k$	State of ATR at timestep k
$\boldsymbol{u}_k$	ATR's control input at timestep k
$\boldsymbol{p}_k$	Position of ATR at timestep k in cartesian 2D coordinates
$\hat{\boldsymbol{p}}_j$	Predicted position of ATR j
x	Position of ATR along global x-axis
y	Position of ATR along global y-axis
θ	Angle of ATR w.r.t the global x-axis
$\boldsymbol{c}$	Center coordinates of an ellipse

Contents

List of Figures

List of Tables

1
Introduction

This chapter presents the thesis on a base level, the introduction contains the background, some previous work that has been done and why the chosen subject needs more research. The problem statement and research questions is also presented together with the limitations of the thesis.

1.1 Background

At Volvo GTO there is a greater need for higher flexibility along the production lines to satisfy the demand of orders with higher variance. The proposed solution to solve this problem is to use a fleet of Autonomous Transport Robots (ATRs), and consequently be able to easily change the individual components along the production line and change the specification of the trucks that are being built.

Therefore, Volvo has created the Generic Photogrammetry based Sensor System (GPSS) project, which aims to use ceiling mounted cameras to detect obstacles such as humans, robots and other obstacles that might exist on the factory floor. The positional data of these obstacles are then sent to a controller which generates the trajectories for the robots as they follow a predetermined path. This would in the long run be cheaper and bring more flexibility to the system than the current solution, which is to guide the robots using fixed magnetic strips placed along the factory floor.

1.2 Previous work

This work is mostly based on [14] and [7], which is a Master's thesis from last year on the same project and a research paper continuing on the work from the just mentioned Master's thesis.

In [14], a centralized NMPC was developed and showed that two ATRs could be coupled together and move in unison along a path. The solution shows some promising results although the controller sometimes violated constraints on obstacles, and when tested in real life the ATRs couldn't move synchronized which might be caused by time delays from the calculations and unsynchronized trajectories. The thesis [14] also mentions that when using OpEn [21], it is only possible to set absolute hard constraints on the optimization variables which lead to the ATRs breaking constraints a bit even though hard constraints were implemented.

To continue on the work, [7] developed both a distributed and a centralized NMPC formulation to examine the difference between them. When using the distributed NMPC each robot is evaluated independently and the controller only considers the objective function for one robot at the time. In a centralized controller, all the ATRs' objective functions are added together to make one big solution for all robots instead of multiple smaller solutions. From the results it is shown that the computation time for the distributed controller is both smaller and scales linearly with the number of robots, whereas the centralized controller has a higher computation time and scales exponentially with increasing amounts of robots. For more complex solutions the centralized controller does perform better in terms of the ATRs' synchronized movement since the centralized controller wants to minimize the cost for the entire system while the distributed controller only cares about minimizing the cost for each individual robot independently. The main problem that can be found in [7] is that in unfeasible solutions, the robots tend to violate the constraints put on, to avoid obstacles as in [14].

In [15], a distributed NMPC is developed for collision avoidance in a real time application intended for several unmanned aerial vehicles (UAVs). The proposed solution uses constraints on the obstacles which are enforced with the augmented Lagrangian method. Additionally, an algorithm for selecting which agents that should be included in the NMPC used in order to cut down on the computational times. The NMPC uses the PANOC solver and shows good results in regards to the computational times, however it does also have some problems with positional constraints breaking.

1.3 Problem statement

To continue research in this area, three research questions was created which aims to solve some of the problems mentioned in the previous section and to help with the project presented in the background section.

- How can trajectories leading to a collision be detected, and in which ways is it then possible to avoid or lessen the damage?
- How does the complexity of the NMPC problem scale with increasing numbers of ATRs, measured in mean and variance of calculation time?
- How can a new reference trajectory be generated in the case of obstacles causing unfeasible or bad trajectories?

1.4 Limitations

This project aims to create a NMPC formulation that solves the current problems such as unfeasible trajectories and issues caused by communication delays. As such the project will not include any design of the ATRs, in addition to this the simulation model uses the velocity and angular velocity of the ATRs as inputs and the thesis will not consider the conversion between the velocities and the corresponding

inputs for the electric motors.

The inputs to the NMPC, containing the positions of both dynamic- and static obstacles is not dealt with in this project as it is managed by other groups. As this thesis has its main focus on the trajectory planning, any path planning wont be included since it also was dealt with in [14].

2

Trajectory planning

This chapter covers some of the ways that trajectory planning can be done and aim to motivate why a NMPC controller is used for the studied problem. Trajectory planning [9] can be summarized to calculating inputs for the robot so that it follows a desired path while also taking system and state constraints into consideration. For instance one can use trajectory planning to generate inputs which follows a path, minimizes the energy used and at the same time avoids collision with obstacles. To be able to use a certain trajectory planner for this thesis it would need to be computationally fast to be able to run in real-time, take actuator constraints on signals given to the robot and be able to avoid both dynamic and static obstacles.

One of the ways that trajectory planning can be done is by using polynomial segments connecting waypoints as presented in [19]. The presented algorithm is used for generating a trajectory for a UAV and can avoid static obstacles and deal with constraints on the actuators on the UAV. The results show that the computation time is low enough to apply the algorithm for real-time applications for a small amount of agents. However, the algorithm cannot deal with dynamical obstacles and loses its numerical stability when more than five segments together with constraints are used. Additionally since it can only be implemented for a small number of agents in real-time it is a bad fit for generating trajectories to the problem at hand.

Four other concepts in which trajectory planning can be done is potential fields, cell decomposition, interdisciplinary techniques, optimal control and model predictive control. All of these are presented and compared to each other in [6]. When using potential fields to generate a trajectory, obstacles get assigned a repulsive field and the zones which are safe to be in gets assigned a attractive field. One can then combine the fields ad use an algorithm to create a path which follows the steepest gradient which in turns generate a collision free trajectory. The use of this algorithm has been experimentally proven to work for low speed maneuvers but since it cannot deal with actuator constraints on the robot it isn't suitable for use in this thesis.

The Rapidly-exploring Random Tree (RRT) [17] method is an example of a cell decomposition algorithm and works by growing a tree like structure starting with a root in the starting position and finding a way to the goal position. The expansion of the tree is done by randomly adding nodes into the defined space and if it is close enough and avoids collisions it is added to the tree. When using cell decomposition algorithms [6] it is possible to incorporate constraints on the actuators but the algorithm suffer from jerky trajectories and high computational complexity with an

increasing number of obstacles.

Inter-disciplinary techniques combines the use of steady-state equilibrium trajectories and pre-specified maneuvers to generate an obstacle free trajectory [6]. The method has been tested in low order model simulations where it has been shown that the algorithm might be able to be used in real-time applications. However it doesn't seem to have been tested in real experiments and as such it might not work.

Optimal control methods uses an objective function which is minimized to achieve both smooth and collision free trajectories while considering constraints on the robot. These methods does however often not take into account the nonlinearities brought in by the robot dynamics which can result in unfeasible solutions [6].

Model predictive control is a type of optimal control method which uses the receding horizon principle. It has the ability to deal with nonlinearities and minimizes an objective function to generate the most optimal control signals. It is mentioned that because of the nonlinearities, the computational complexity might be to high to be used in real-time but as it is shown in [7] and [15] it is possible to use. Additionally, as it is a optimal control method it can generate collision free trajectories and deal with constraints making it is a good fit for the problem at hand.

3

Non-Linear Model Predictive Control

This chapter covers the basis of how a MPC works and how a general cost function and constraints might look. Furthermore it is explained how a NMPC is implemented for this thesis, by defining the inputs, constraints and finally the objective function.

3.1 General Model Predictive Control

Model Predictive Control or MPC [18] is a control algorithm which can handle multi-input, multi-output (MIMO) systems as opposed to other controllers such as PID. The control signal from the MPC is computed by predicting the system forward a set amount of timesteps which is the horizon and a control signal is computed for each of the timesteps. The control signals are calculated so that they minimize the cost for an objective function while keeping both the system and control signals within potential constraints. The first control signal is then used on the system and the prediction process starts over. Figure 3.1 shows how a general MPC controller predicts and optimizes the control signals over a set horizon for reference tracking, where k denotes the timestep and p the prediction horizon. However, going forward the prediction horizon is denoted N, and the timestep is denoted k. As the robotic model shown in Equation 3.4 is non-linear, combined with the constraints used in

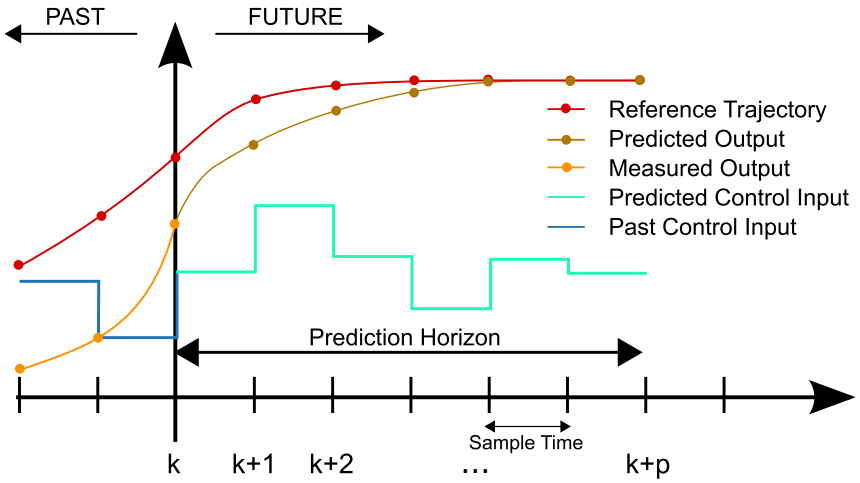


Figure 3.1: General MPC scheme for reference tracking [3].

the objective function makes it necessary to use a Non-Linear Model Predictive Controller (NMPC) [8]. For this thesis, a distributed NMPC is used which only considers one robot at a time when calculating the trajectories, compared to a centralized NMPC which considers all of the robots at the same time. A distributed NMPC is used because it's computation time is lower and doesn't scale as fast compared to a centralized NMPC [7] while returning similar results. The distributed NMPC does however still get information containing both the current states and the predicted states for the other ATRs in order to be able to avoid collisions between the robots.

3.1.1 Objective function

The objective function is used used to determine the behavior of the controller and an example from [18] can be seen in Equation 3.1. This objective function aims to minimize the cost given an initial state $\boldsymbol{x}(0)$ and the control inputs for each of the timesteps in the horizon N.

$$V_N(\boldsymbol{x}(0), \boldsymbol{u}(0:N-1)) = \boldsymbol{x}^T(N)P_f\boldsymbol{x}(N) + \sum_{k=0}^{N-1}(\boldsymbol{x}^T(i)Q\boldsymbol{x}(i) + \boldsymbol{u}^T(i)R\boldsymbol{u}(i)) \quad (3.1)$$

In this objective function there is a stage cost on both the states $\boldsymbol{x}$, and inputs $\boldsymbol{u}$ in each timestep k, decided by the weight matrices Q and R respectively. There is also an additional cost on the final state which is determined by the weight P_f

3.1.2 Constraints

When using a MPC one can set hard constraints on for example the control signals having to be inside a range such that a system can use them, it is also possible to set constraints on the states. Additionally, the prediction of the states has to follow the dynamics of the system denoted f. These constraints can be seen in Equation 3.2

$$\begin{aligned} V_N(\boldsymbol{x}(0), \boldsymbol{u}(0:N-1)) = \boldsymbol{x}^T(N)P_f\boldsymbol{x}(N) + \sum_{k=0}^{N-1}(\boldsymbol{x}^T(i)Q\boldsymbol{x}(i) + \boldsymbol{u}^T(i)R\boldsymbol{u}(i)) \\ s.t \;\; \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \\ \boldsymbol{x}_{min} \leq \boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{aligned} \quad (3.2)$$

Using hard constraints can lead to unfeasible solutions and thus they can be be relaxed into soft constraints [4]. The constraints are then instead included as a cost in the objective function as in Equation 3.3 with Q_x being the cost for breaking the soft constraints.

$$Q_x \cdot min(0, \boldsymbol{x}_k - \boldsymbol{x}_{min})^2 \quad (3.3)$$

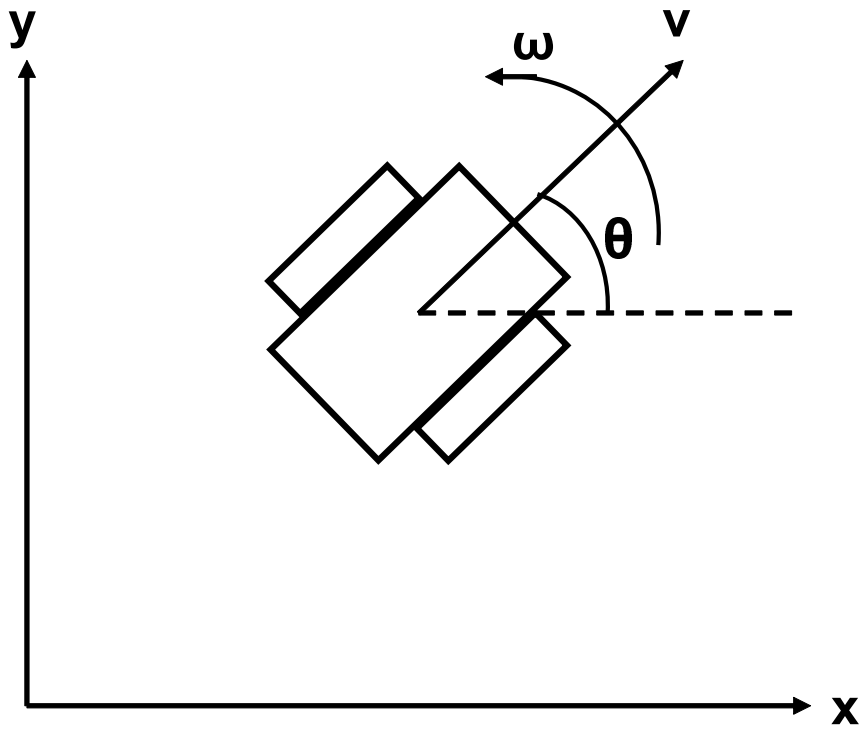


Figure 3.2: Model of an ATR together with its states and inputs.

3.2 Implemented Non-Linear Model Predictive Control

This section covers the inputs used by the NMPC, the Obstacles, Robots and boundaries. The robots are defined as points in the NMPC and such the obstacles need to be padded to compensate for this. It also covers the dynamics of the robots and how they are discretized.

3.2.1 Modelling the autonomous transport robot

The motion model for the ATRs being used was first derived in [1], and is shown in Equation 3.4.

$$\begin{aligned} \dot{x} &= v \cdot cos(\theta) \\ \dot{y} &= v \cdot sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{3.4}$$

In Figure 3.2, the ATR is shown with the coordinate system to easier show how it correlates to the world coordinates. The variable x represents the ATRs position along the x-axis, y represents the position along the y-axis and θ is the angular direction of the ATR with respect to the x-axis. These three variables are the states of the ATR whereas v and ω is the inputs to the ATR. v is the speed of the ATR, and ω is the rotational speed around its center. In some cases, the robots position p is only needed which contains the x and y position of the robot.

To be able to use the motion model within the NMPC, it is discretized using the Runge-Kutta 4 (RK4) method as it is both fast in terms of computational time and and has a good accuracy to the real model [11]. There are different versions of the Runge-Kutta methods, where the difference between them is that a higher order version is more computationally complex but provides a more accurate representation of the real model. A general scheme on how to use RK methods can be seen in 3.5 where a, b and c are constants from a Butcher tableau, f is the function for the continuous model, Δt is the length of the timestep and s is the number of stages for the selected method.

$$\begin{aligned} K_i &= f\Big(x_k + \Delta t \sum_{j=1}^{s} a_{ij} K_j, u(t_k + c_1 \Delta t)\Big) \\ x_{k+1} &= x_k + \Delta t \sum_{i=1}^{s} b_i K_i \end{aligned} \tag{3.5}$$

The Butcher tableau that is used for RK4 is presented in Table 3.1 and the complete RK4 scheme that is used can be seen in 3.6.

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table 3.1: Butcher tableau for RK4.

$$\begin{aligned} K_1 &= f(x_k, u(t_k)) \\ K_2 &= f\Big(x_k + \frac{\Delta t}{2} K_1, u\Big(t_k + \frac{\Delta t}{2}\Big)\Big) \\ K_3 &= f\Big(x_k + \frac{\Delta t}{2} K_2, u\Big(t_k + \frac{\Delta t}{2}\Big)\Big) \\ K_4 &= f\Big(x_k + \Delta t K_3, u(t_k + \Delta t)\Big) \\ x_{k+1} &= x_k + \Delta t\Big(\frac{K_1}{6} + \frac{K_2}{3} + \frac{K_3}{3} + \frac{K_4}{6}\Big) \end{aligned} \tag{3.6}$$

3.2.2 Obstacles

This subsection covers how the different obstacles used in the NMPC are defined. The obstacles that are handled are static- and dynamic obstacles, robots and the boundaries of where the ATR is allowed to operate. The obstacles are defined as in [20] where a set of obstacles $\mathcal{O} \subseteq \mathcal{R}^{n_d}$ is used, and each obstacle is described as an intersection of a finite number of nonlinear inequalities. A common notation in this subsection is $[x]_+$ which is the same as $max(0, x)$.

3.2.2.1 Static obstacles

The static obstacles that are used are modeled as polygons and constraints on being outside these polygons can be represented as in 3.7, where m is the number of

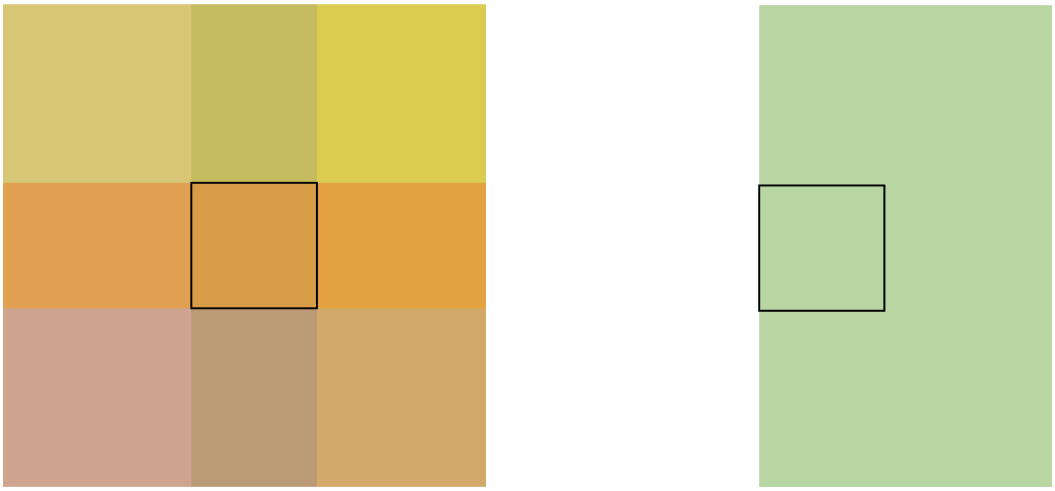
inequalities needed to represent the polygon and $b_i, a_i \in \mathbb{R}^{n_d}$.

$$\mathcal{O}_S = \{\boldsymbol{z} \in \mathbb{R}^{n_d} \ : \ b_i + a_i^T \boldsymbol{z} < 0, i \in \mathcal{N}_{[1,m]}\} \tag{3.7}$$

Equation 3.7 can be rewritten into a equality constraint instead as in 3.8.

$$\mathcal{O}_S = \{\boldsymbol{z} \in \mathbb{R}^{n_d} \ : \ \prod_{i=1}^{m} [b_i + a_i^T \boldsymbol{z}]_+^2 = 0\} \tag{3.8}$$

In the case when $n_d = 2$ as in this thesis, the inequalities represents halfspaces and when a point $\boldsymbol{z}$ is inside all of the halfspaces, it is inside the polygon. In Figure 3.3 it is shown how these halfspaces can be used to used to represent a quadratic polygon.



(a) **(b)**

Figure 3.3: In a), four halfspaces are used to represent the quadratic polygon (black square). In b) only one of the halfspaces are shown to for easier interpretation.

As the boundaries also are defined as polygons they can be represented in the same way as the static obstacles, but instead setting a constraint on being inside all of the halfspaces.This can be done by changing the product sign to a sum and reversing the inequality sign since it is now only necessary for one of the reversed inequalities to not hold for the constraint to break. The new representation can be seen in 3.9, where $[x]_-$ represents $min(0, x)$.

$$\mathcal{O}_B = \{\boldsymbol{z} \in \mathbb{R}^{n_d} \ : \ \sum_{i=1}^{m} [b_i + a_i^T \boldsymbol{z}]_-^2 = 0\} \tag{3.9}$$

When using this halfspace representation, it is required for the obstacles to be convex otherwise it will not represent the entire polygon and just parts of it. There are multiple ways to solve this problem, however this thesis doesn't consider those cases but it can be read about more in [14].

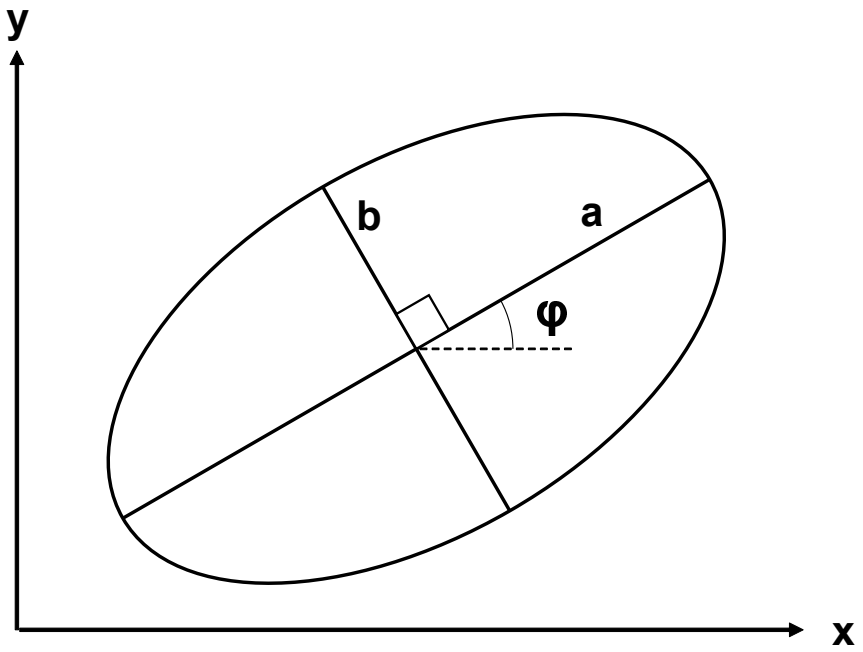


Figure 3.4: Representation of a general ellipse.

3.2.2.2 Dynamic obstacles

Dynamic obstacles are defined as ellipses with a centerpoint in $[x_c, y_c]$, and the lengths for the semimajor and semiminor axes being a and b respectively. The obstacle can also be rotated which is why φ is included and represents the angle to the x-axis, in Figure 3.4 an ellipse can be seen with its different properties. The velocity of the obstacle is also needed to be able to get the predicted states which are later used in the objective function. As with the static obstacles, the dynamic obstacles also has a padding added to the a and b values to compensate for the robots being defined as dots in the NMPC. A mathematical expression for the outline of an ellipse can be seen in 3.10.

$$\frac{(x_c \cdot cos(\varphi) + y_c \cdot sin(\varphi))^2}{a^2} + \frac{(x_c \cdot sin(\varphi) - y_c \cdot cos(\varphi))^2}{b^2} = 1 \tag{3.10}$$

To form an equality constraint on not being inside ellipses, one can use 3.11 as in [20], where $\boldsymbol{c}$ is the vector containing the center coordinates of the ellipse, $\boldsymbol{z}$ is a vector containing arbitrary points and E is the shape matrix of the ellipse.

$$\mathcal{O}_E = \{\boldsymbol{z} \in \mathcal{R}^{n_d} \ : \ [1 - (\boldsymbol{z} - \boldsymbol{c})^T E (\boldsymbol{z} - \boldsymbol{c})]_+^2 = 0\} \tag{3.11}$$

For avoiding other robots, a constraint can be formed in a similar way as in 3.11. However, since the robots are defined as points having to be a distance from each other it can be seen as a robot not being allowed to be inside a circle with the center being the other robots position. In 3.12, the constraint for not being inside a circle is shown with r being the radius of the circle.

$$\mathcal{O}_C = \{\boldsymbol{z} \in \mathcal{R}^{n_d} \ : \ [r^2 - (\boldsymbol{z} - \boldsymbol{c})^T (\boldsymbol{z} - \boldsymbol{c})]_+^2 = 0\} \tag{3.12}$$

3.2.3 Objective function

This subsection covers the individual cost functions for e.g deviating from the reference path and colliding with obstacles. The obstacle avoidance constraints are relaxed into soft constraints instead with an exception on the static obstacles as it uses both hard and soft constraints. It was decided to use the combination of soft and hard constraints because when only hard constraints are used, the controller often calculated bad trajectories as it didn't converge properly, to counteract the bad solutions, an additional smaller padding is used to lead the ATRs away from the hard constraints and bad solutions. In the end the complete objective function is presented with all the costs and the selected weight matrices.

3.2.3.1 Cost for colliding with static obstacles and the boundary

The relaxed constraint for colliding with static obstacles can be seen in 3.13 where N_s is the number of static obstacles and Q_O being the weight matrix on being inside the obstacles. The cost for being outside the boundary can similarly be formulated as in 3.14 with Q_B being the weight matrix.

$$J_O(\boldsymbol{p}) = Q_O \cdot \sum_{o=1}^{N_s} \prod_{i=1}^{m} [b_i + a_i^T \boldsymbol{p}]_+^2 \tag{3.13}$$

$$J_B(\boldsymbol{p}) = Q_B \cdot \sum_{i=1}^{m} [b_i + a_i^T \boldsymbol{p}]_-^2 \tag{3.14}$$

3.2.3.2 Cost for colliding with dynamic obstacles

When determining the cost for colliding with dynamic obstacles, the hard constraints from 3.11 can be relaxed into soft constraints as in 3.15 with the weight Q_{dyn}.

$$J_{dyn}(\boldsymbol{p}, \boldsymbol{c}) = Q_{dyn} \cdot [1 - (\boldsymbol{p} - \boldsymbol{c})^T E (\boldsymbol{p} - \boldsymbol{c})]_+^2 \tag{3.15}$$

3.2.3.3 Cost for colliding with other robots

The constraint on robots having to be a certain distance from each other is relaxed into the form shown in 3.16. It takes the position of the first robot and predicted position of the second robot at the same timestep and uses a weight Q_{atr} for when the robots are closer than the wanted distance r. The predicted position of the other robot is denoted $\hat{\boldsymbol{p}}_{\boldsymbol{j}}$.

$$J_{atr}(\boldsymbol{p}, \hat{\boldsymbol{p}}_{\boldsymbol{j}}) = Q_{atr} \cdot [r^2 - (\boldsymbol{p} - \hat{\boldsymbol{p}}_{\boldsymbol{j}})^T (\boldsymbol{p} - \hat{\boldsymbol{p}}_{\boldsymbol{j}})]_+^2 \tag{3.16}$$

3.2.3.4 Cost on acceleration

The cost on the acceleration, or jerk of the robot is used to help get a smooth trajectory with smaller difference between the control inputs in each timestep. The equation for the cost can be seen in 3.17

$$J_{acc}(\boldsymbol{u}_k, \boldsymbol{u}_{k-1}) = (\boldsymbol{u}_k - \boldsymbol{u}_{k-1})^T Q_{acc} (\boldsymbol{u}_k - \boldsymbol{u}_{k-1}) \tag{3.17}$$

The function takes the control inputs for two adjacent timesteps, and uses the weight matrix Q_{acc} to calculate a cost on the acceleration.

3.2.3.5 Cost for deviating from reference

Since it is wanted for the robots to follow a desired path, a cost on deviating from the reference path is used. The cost is calculated by squaring the distance from the robot to the reference point $\boldsymbol{p}^{ref}$ and multiplying it with the weight Q_τ which can be seen in 3.18

$$J_\tau(\boldsymbol{p}^{ref}, \boldsymbol{p}) = (\boldsymbol{p}^{ref} - \boldsymbol{p})^T Q_\tau (\boldsymbol{p}^{ref} - \boldsymbol{p}) \tag{3.18}$$

3.2.3.6 Cost for deciding tracking error point

To determine which point along the parameterized reference path the tracking error should be calculated from, an additional optimization variable s is used. The cost for this variable is calculated by multiplying the optimization variable with the negative weight Q_s as in 3.19. This is done as it is desired for the robot to move forward and since s indirectly decides the tracking error it can be seen as the variable which is driving the ATR forward along the reference path.

$$J_s(s) = s \cdot -Q_s, \ s \in [0, N] \tag{3.19}$$

3.2.3.7 Complete objective function

Combining all of the separate cost functions and the constraints, the final objective function can be seen in 3.20.

$$\begin{aligned} \min_{u,s} \ & \sum_{k=1}^{N} [J_{acc}(\boldsymbol{u}_k, \boldsymbol{u}_{k-1}) + J_O(\boldsymbol{p}_k) + J_B(\boldsymbol{p}_k) + J_{dyn}(\boldsymbol{p}_k) \\ & \sum_{j=1}^{P-1} J_{atr}(\boldsymbol{p}_k, \hat{\boldsymbol{p}}_{j,k})] + J_s(s) + J_\tau(\boldsymbol{p}_s^{ref}, \boldsymbol{p_N}) \\ s.t \ & \boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \\ & s_{min} \leq s \leq s_{max} \\ & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ & \boldsymbol{p}_k \in \mathcal{O}_S \end{aligned} \tag{3.20}$$

To get the desired behavior of the of ATRs from the NMPC, the different weight matrices were tuned as in 3.21.

$$\begin{aligned} & Q_s = 20, \ \ Q_\tau = \begin{bmatrix} 150 & 0 \\ 0 & 150 \end{bmatrix}, \ \ Q_{acc} = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix} \\ & Q_{atr} = Q_O = Q_B = 200, \ \ Q_{dyn} = 500 \end{aligned} \tag{3.21}$$

And the constraints on s and u were set as in 3.22.

$$\begin{aligned} & s_{min} = 0, \ \ s_{max} = 20 \\ & u_{min} = \begin{bmatrix} 0 \\ -2 \end{bmatrix}, \ \ u_{max} = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix}, \ \ u = \begin{bmatrix} v \\ \omega \end{bmatrix} \end{aligned} \tag{3.22}$$

3.2.4 Solver choice

The solver which is used for minimizing the objective function is a Proximal Averaged Newton-type method for Optimal Control (PANOC) [22]. The solver uses a single-shooting formulation of the problem, meaning that the states are essentially eliminated from the problem and the costs only depend on the inputs. To find the minimum of the objective function, PANOC uses line-search instead of sequential quadratic programming which helps it converge faster.

To generate the code for the PANOC solver, OpEn [21] was used. To write the code generator, CasADi [2] needs to be used which generates C code which is later compiled into Rust [16]. The generator requires a predetermined number of parameter inputs and outputs, and it can deal with constraints on the states by using either the augmented Lagrangian method or the penalty method which are described in [21]. The solver used to generate the results was built to deal with up to five static obstacles, one boundary, one dynamic obstacle and up to 20 ATRs. To deal with the positional constraints in 3.20, the augmented Lagrangian method was used.

4

Regeneration of reference online

At initialization, a complete path is generated. A part of that global path, the ATR's horizon, is used as a reference path for the NMPC. When that horizon intersects with a static obstacle, see Figure 4.2b there is a need for regenerating a part of the global path so that it instead goes around the obstacle. If not, and the obstacle is wide enough, the NMPC with its short horizon input, may not be able to find a solution where the ATR goes around the obstacle.

4.1 Regeneration and star-shaped obstacles

To generate a path around a obstacle, the work done by the Phd student Albin Dahlin is used [5]. It implements theory presented in [12], which uses "smooth motion fields around obstacles with edges" ([12], p.1). To summarize [12]; if the following dynamic holds:

$$\dot{\xi} = \mathbf{M}(\xi)\mathbf{f}(\xi) \tag{4.1}$$

where ξ is representing the position of a point mass (e.g. the ATR's), $\mathbf{M}(\xi)$ the modulation matrix and $\mathbf{f}(\xi)$ the dynamical system. Then the dynamics of the point-mass modelled ATR is collision free. The dynamical system for generating the new paths can be seen in Equation 4.2, with ξ^a being the goal point and $b \in \mathbb{R}$ being a scaling parameter set to 1.

$$\boldsymbol{f}(\xi) = -b(\xi - \xi^a) \tag{4.2}$$

The modulation matrix is constructed as in 4.3, with $\boldsymbol{E}(\xi)$ being a basis matrix containing orthonormal tangent vectors of the boundary closest to the point ξ for the an obstacle, and $\boldsymbol{D}(\xi)$ being a diagonal eigenvalue matrix. Within $\boldsymbol{D}(\xi)$ there are two variables, c^{rep} and ρ which determine how close to the obstacle the path gets generated and they are set to 1.1. More theory behind Equation 4.1 is further explained in [12].

$$\boldsymbol{M}(\xi) = \boldsymbol{E}(\xi)\boldsymbol{D}(\xi)\boldsymbol{E}(\xi)^{-1} \tag{4.3}$$

4.1.1 Path regeneration

Using Equation 4.1, the next collision free step is found. By reiterating Equation 4.1 for the entire distance, a path from any starting point to any goal point that also avoids obstacles can be generated. That is given under two circumstances:

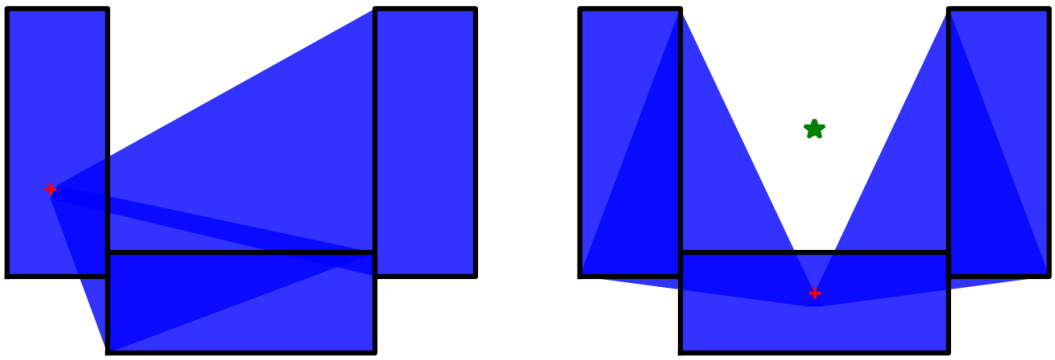
1. Firstly, no obstacles overlap and all obstacles are star-shaped.
2. Secondly, that the kernel of a obstacle is not directly in between the starting and the goal point.

The path regeneration is activated when the reference horizon, which is 2 meter long and is also the input to the NMPC, collides with a obstacle. Then the part of the global path that is occupied by the obstacle is regenerated, with the ATR's position as starting point, and the first unoccupied point on the future path as the goal point. In Figure 4.2, it is visualized when path regeneration is activated and which part of the total path it regenerates.

4.1.2 Star-shaped and overlapping obstacles

A obstacle is defined as star-shaped if there exist a point, a kernel, within the region in which all other boundary/surface points are visible [12]. By visible, it is meant that if a line is drawn from the kernel to any border/surface point of the region, that line shall not intersect with any other outer border/surface lines. If two or more obstacles overlap, they have to be converted to a single obstacle that consists of the combined area that they cover.

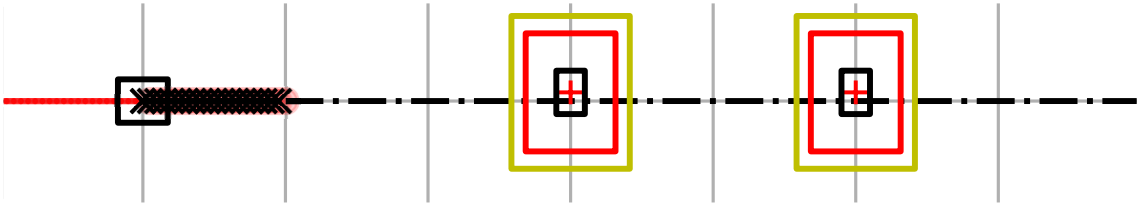
To fulfill the first condition in 4.1.1, all obstacles has to be processed to ensure that they are star-shaped. Non-star-shaped obstacles can be converted by adding hulls to obstacles. That is what is done in the solution that has been implemented in [5], and is used in this thesis. Two example of hulling is also shown in Figure 4.1. If they are not star-shaped there is a risk of Equation 4.1 converging to a local minima and not converging to the goal point.



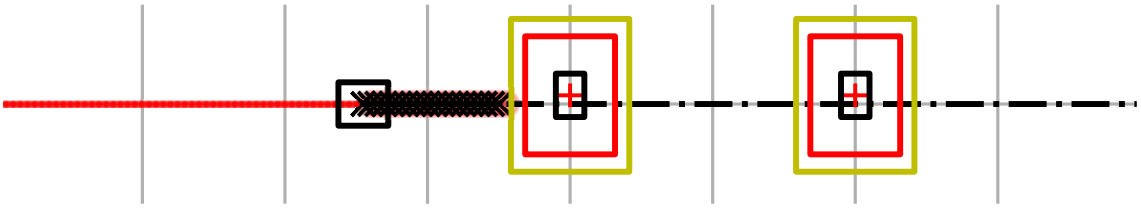
(a) (b)

Figure 4.1: The three black lined rectangles represent a cluster of obstacles that overlap. That cluster can be seen as a single obstacle that is not star-shaped, but can be converted by adding hulls. Those conversions results in a star-shaped obstacle which covers the entire blue area, and has a red cross marking its kernel. When the starting or goal point (green star) is inside the hull, a part of the hull is excluded, as seen in Figure 4.1b.

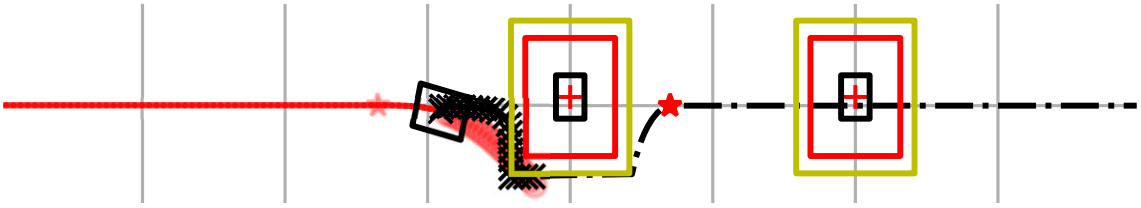
In Figure 4.2, there are two obstacles, each obstacle is described by three rectangles with different sizes and colors, black, red and yellow. The smallest and black colored rectangular represent the unpadded obstacle. The red lined rectangular represents both the padded obstacle and the hard constraints that the point-mass model is not allowed to cross. Finally, the yellow colored rectangular represent the extra-padding which is used for the soft constraints. It is the yellow lined rectangular that, in this chapter, has to be star-shaped and will be avoided in the path regeneration. It is also a collision with that rectangular which activates a reference regeneration mentioned in this chapter.



(a) A robot whose global path (black dash dotted line) is partly occupied by a two static obstacles.



(b) When the the reference horizon (black crosses) in the next timestep collides with the first obstacle, path regeneration is activated.



(c) The part of the global path that is occupied by the colliding obstacle was regenerated. As seen, the 2nd condition in subsection 4.1.1 is fulfilled, since the kernel (red plus sign) is not directly in between the starting (transparent star) och and goal point (opaque star).

Figure 4.2: The figures above demonstrate when the path generation is activated and that only the part of the total path that is occupied by the colliding static obstacle is regenerated and replaced.

Robots, or other dynamic obstacles are not taken into consideration when calculating a new reference. It is only static obstacles that the regenerated path will avoid. The classification of obstacles is done at initialization and this solution doesn't include a reclassification of obstacles. For further development, the solution could

be expanded so that a ATR or any other previously dynamic obstacle that comes to a standstill for a determined amount of time, are reclassified as a static obstacle.

5

Avoiding collisions

This chapter covers the module which is responsible for detecting collisions outside of the NMPC and how it is designed to work. The module exist of three parts, each part dealing with their own type of obstacle. Since the module is outside of the NMPC it has the ability to independently change the control signals and stop a robot if a collision is detected.

5.1 Complete module

The designed module exists of three parts each responsible for their own type of obstacle, these parts are combined into a module which has the ability to slow down the robots ahead of a possible collision to dampen the forces or completely stop the robot by overwriting the control signals from the NMPC. The module takes the robots current and predicted states as well as the positions and predicted positions of all the obstacles as input and outputs a multiplier to be used on the control inputs. There are two main reasons for developing this module, the first being that the solver used might not converge to good solutions and can in turn generate control signals which steer the robot into obstacles. The second reason is that the dynamic obstacles such as humans might not follow their predicted trajectories meaning that the controller doesn't have the correct data and might think an area is unoccupied but in reality it isn't. The algorithm for the entire module containing the sub-modules can be seen in Algorithm 1.

5.1.1 Detecting collisions with static obstacles

The algorithm for detecting collisions with static obstacles takes the robots predicted position and the positions of the static objects and outputs a Boolean variable which says if the robot needs to stop completely or not, along with a multiplier used to manipulate the control signals. The collision detection uses parts of the Shapely [10] library for calculating the distance between a polygon representation of the robot and the polygon representing the static obstacle. The multiplier being calculated depends on how far along in the predicted horizon a collision is detected and is constrained between a set minimum and one, unless the robot needs to stop immediately as the multiplier is then set to zero. The equation for calculating the multiplier is the same for all sub-modules, and the algorithm for detecting collisions can be seen in Algorithm 2.

Algorithm 1 Complete module

```
initialize list for stopped ATRs
initialize list with multipliers for each ATR
determine predicted centers for dynamic objects
for every ATR do
    determine latest timestep needed to stop
    if current ATR in list of stopped ATRs then
        set multiplier to 0
        set initial control input to 0
    else
        for every timestep in horizon do
            get predicted state of ATR
            call check_collision_static
            if detected collision then
                set predicted states as current state
                add ATR to list of stopped ATRs
                set initial control input to 0
            call check_collision_dynamic
            if detected collision then
                set predicted states as current state
                add ATR to list of stopped ATRs
                set initial control input to 0
            call check_collision_robots
            if detected collision then
                set predicted states as current state
                add ATR to list of stopped ATRs
                set initial control input to 0
        multiply initial control input with multiplier
```

Algorithm 2 *check_collision_static*

```
for every static obstacle do
    if ATR inside obstacle then
        multiplier = max(min_multiplier, k/N)
        if current timestep < critical timestep then
            set multiplier to 0
            set Detected collision to True
        return multiplier, detected collision
return multiplier, detected collision
```

5.1.2 Detecting collisions with dynamic obstacles

The inputs to the sub-module for detecting collisions with dynamic obstacles takes the predicted state of both the robot and predicted center of the dynamic obstacle as input and outputs a Boolean variable which says if the robot needs to stop completely or not, along with a multiplier used to manipulate the control signals. To calculate if the robot is inside the obstacle, 3.11 is used. In this sub-module, the padded ellipse and point representation is used in the same way as in the NMPC in contrast to when checking on static obstacles. The multiplier is then calculated, and the algorithm can be seen in Algorithm 3.

Algorithm 3 *check_collision_dynamic*

```
if ATR is inside ellipse then
    set multiplier = max(min_multiplier, k/N)
    if current timestep < critical timestep then
        set multiplier to 0
        set detected collision to True
    return multiplier, detected collision
return multiplier, detected collision
```

5.1.3 Detecting collisions with other robots

The sub-module for detecting collisions with other robots take the predicted states for all robots, and the robots current states as inputs and outputs a Boolean variable which says if the robot needs to stop completely or not, a multiplier for the control signals, the list of stopped robots, predicted states of the robots and the control signals for the robots. To determine if two robots are colliding, the shapely [10] is used as the robots are represented as polygons, and the multiplier is calculated. If a robot would need to stop, it can also mean that the other robot also would need to stop and to counteract both robot stopping this module uses a priority to see which of the robots would need to stop. Currently, this is implemented so that the robot with a higher number has a higher prioritization, however in the future it could be improved to prioritize the robot which is for instance “late” compared to where it should be. To determine which of the robots that should stop, the less prioritized robots position is set to it current state and if the prioritized robot then has a free path, it gets to go while the other robot is stopped and added to the list of stopped robots. If the path for the prioritized robot isn’t free, it is stopped and the less prioritized robot can continue moving. The pseudo code for this sub-module can be seen in Algorithm 4

Algorithm 4 *check_collision_robots*

```
1: for every other ATR do
2:     if current ATR and other ATR collides then
3:         set multiplier[current ATR] to max(min_multiplier, k/N)
4:         if current timestep < critical timestep then
5:             check if path is free when other ATR stands at original position
6:             if Path isn't free then
7:                 set multiplier[current ATR] to 0
8:                 set detected collision to True
9:                 return multiplier, detected collision, predicted states,
10:                        stopped robots, robots
11:            set predicted states for other ATR to current state
12:            add other ATR to stopped ATRs list
13:            set multiplier[other ATR] to 0
14:            set initial control input of other ATR to 0
15: return multiplier, detected collision, predicted states, stopped robots,
16:         control signals
```

6
Results

This chapter presents the hardware used for the simulations, the predefined use scenarios for evaluating both the suggested implementations and the NMPC.

6.1 Simulation hardware

All of the simulations are run on python with a 2.5GHz 8-core processor with 32 GB of RAM running at 3200MHz on a windows 10 system.

6.2 Performance metrics

To evaluate how the controller performs in the different use scenarios, some data is gathered for each of the runs. These are the computation times of the controller, the closest distance to each obstacle type and how far each ATR deviates from the reference path. For instance, the computation times are saved as they show if the proposed solution is fast enough to be implemented in a real-time system, and the distance to the obstacles are to evaluate how well obstacles are avoided and if the controller can handle more complex situations which might occur. In the data presented, all the values are either averaged for all the robots, or the minimum/maximum for all the robots. The computation times for the solver and reference generation is for one robot whereas the computation time for the collision control is for all the robots in the specific scenario. The number of robots stopped are additive for each robot that is stopped and also each timestep it is stopped. The measured distances are the distances between the polygon representations, except for the hard constraints. In the distance to static obstacle, the boundaries are also included.

6.3 Test scenarios

To evaluate the implementations and the NMPC, a set of ten predefined use scenarios are used. They are chosen to both represent normal scenarios such as an ATR joining a train of ATRs and some scenarios that is designed to challenge the controller and implementations a bit more such as ten ATRs attempting to cross each other in a small area. The ten scenarios are listed below:

1. One ATR going around a static obstacle
2. Two ATRs driving against each other

3. One ATR joining a train of ATRs
4. Two ATRs with colliding paths in a small corridor
5. Five ATRs crossing, including a dynamic obstacle
6. Ten ATRs crossing, including a dynamic obstacle
7. Ten ATRs crossing with 4 smaller static obstacles in the middle
8. Ten ATRs crossing with 5 smaller static obstacles in the middle
9. Two ATRs meeting with obstacle on the preplanned path
10. One ATR going around a tight corners

Additionaly, videos of these scenarios can be seen on YouTube [13].

6.3.1 One ATR going around a static obstacle

The first scenario consists of one robot that is following a path around a static obstacle with a relatively large distance in between them. This is considered to be a more normal scenario which can happen when implemented in a factory, A figure showing the robots trajectory can be seen in Figure 6.1, and a table showing the data gathered from the simulation can be seen in Table 6.1.

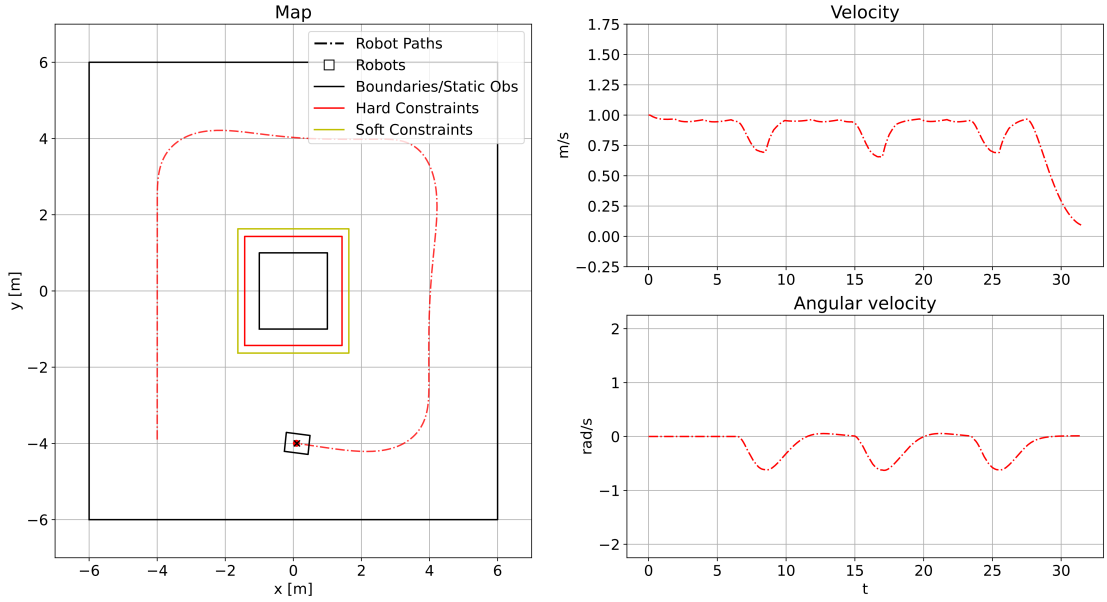


Figure 6.1: Trajectories and velocities for the ATR in scenario 6.3.1.

From the figure it can be seen that the robot keeps a smooth trajectory while staying away from the static obstacle. In the table it can be seen that the computation times for the solver, reference generation and collision control are all low enough for this to be able to run in a real-time system. The reason for the computation time of the reference generation is because even though it isn't necessary to generate a path around any obstacle, it still checks so that the reference path doesn't collide with any obstacle at each timestep. In the figure it can also be noted that in the when the robot turns, it first cuts the desired path a little bit and then overshoots it before returning, this is because only one point is used for the cost on tracking error, this

Table 6.1: Gathered data from simulation of scenario 6.3.1.

Average Computation Time [ms]	7.84
Max Computation Time [ms]	178.23
Variance Computation Time [ms]	14.58
Average Distance From Ref [m]	0.01
Max Distance From Ref [m]	0.11
Variance Distance From Ref [m]	0.02
Closest Static Obstacle [m]	1.51
Distance To Hard Constraints [m]	2.55
Closest Robot [m]	-
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	0.32
Max Time For Generating New References [ms]	0.87
Mean Time for Checking Collisions [ms]	0.62
Max Time for Checking Collisions [ms]	6.31

is also shown in the table showing that the average distance to the reference is low but the robot also deviates a little in the corners.

6.3.2 Two ATRs driving against each other

Continuing on more common situations, a scenario when two ATRs are driving horizontally against each other is presented. This is also a scenario that is considered quite normal however in the real scenarios the robots are most likely not driving on exactly the same level as in this scenario. In Figure 6.2 the robots paths can be seen and in Table 6.2 the data gathered from the simulation can be seen.

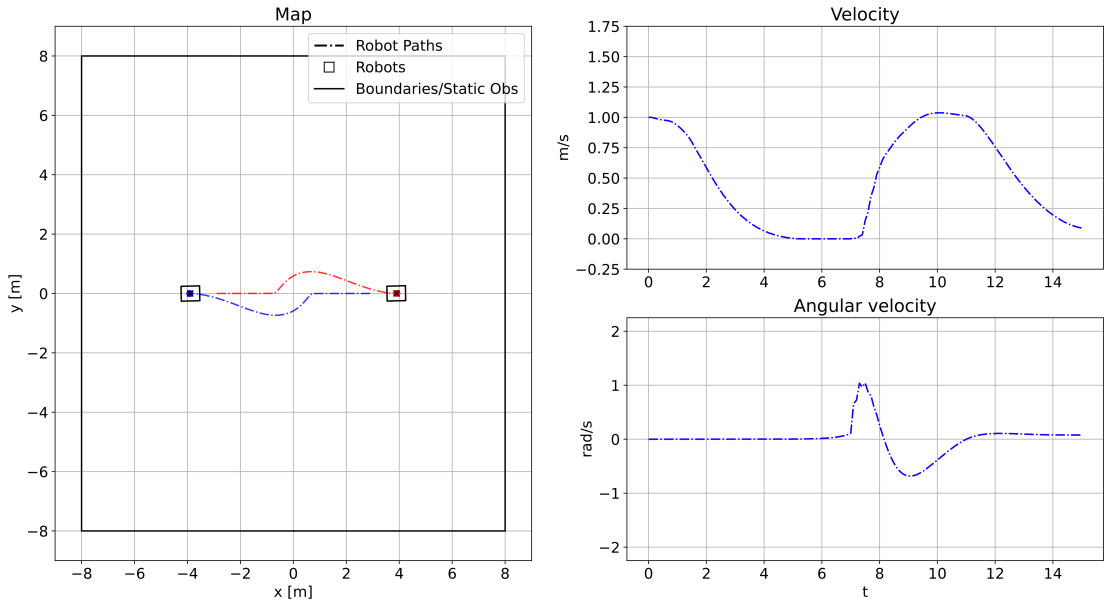


Figure 6.2: Trajectories and velocities for the ATRs in scenario 6.3.2.

Table 6.2: Gathered data from simulation of scenario 6.3.2.

Average Computation Time [ms]	14.21
Max Computation Time [ms]	362.42
Variance Computation Time [ms]	32.40
Average Distance From Ref [m]	0.07
Max Distance From Ref [m]	0.54
Variance Distance From Ref [m]	0.15
Closest Static Obstacle [m]	3.74
Distance To Hard Constraints [m]	-
Closest Robot [m]	0.46
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	0.26
Max Time For Generating New References [ms]	0.87
Mean Time for Checking Collisions [ms]	2.26
Max Time for Checking Collisions [ms]	6.11

From the figure it can be seen that the robots stop for a while when they initially meet, then turn a little and finds a way around each other and reaches the end goal. In the table it can be seen that the average computation times are low and that the robots pass each other at a minimum distance of 0.46 meters. It can also be seen that the maximum distance from the reference is a bit high but this is because the robots have to deviate from their respective paths to pass each other and reach the goal.

6.3.3 One ATR joining a train of ATRs

This scenario is another not so complex situation where four ATRs are travelling horizontally in a line, and one ATR travelling vertically tries to join the other ATRs. The trajectories for the ATRs can be seen in Figure 6.3, and the gathered data can be seen in Table 6.3.

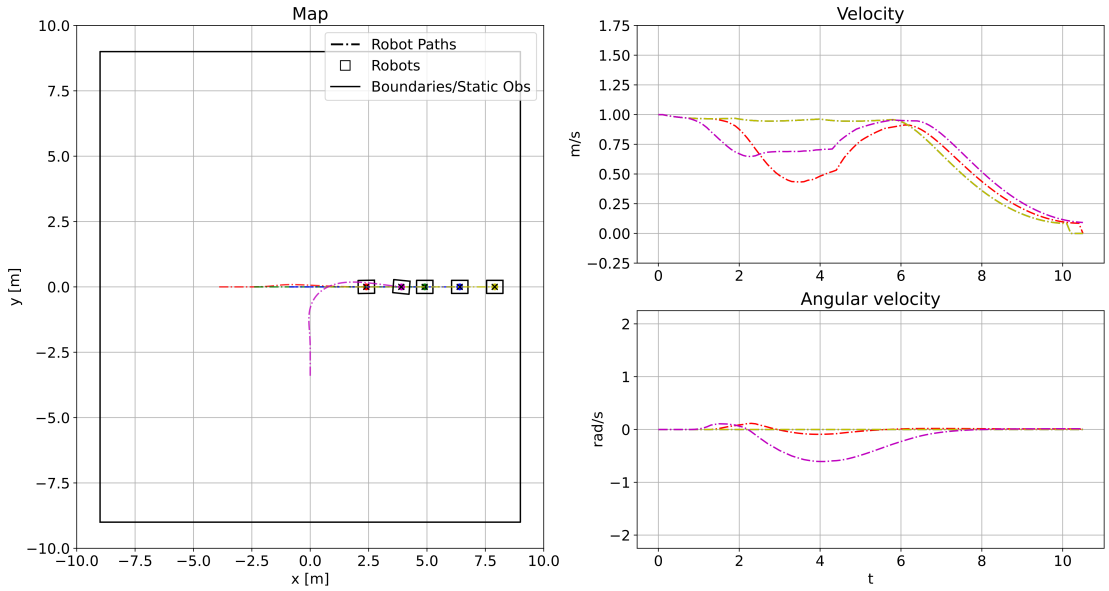


Figure 6.3: Trajectories and velocities for the ATRs in scenario 6.3.3.

Table 6.3: Gathered data from simulation of scenario 6.3.3.

Average Computation Time [ms]	7.94
Max Computation Time [ms]	141.04
Variance Computation Time [ms]	13.29
Average Distance From Ref [m]	0.00
Max Distance From Ref [m]	0.08
Variance Distance From Ref [m]	0.01
Closest Static Obstacle [m]	0.75
Distance To Hard Constraints [m]	-
Closest Robot [m]	0.28
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	0.20
Max Time For Generating New References [ms]	0.58
Mean Time for Checking Collisions [ms]	10.89
Max Time for Checking Collisions [ms]	18.65

From the figure it is shown that the single robot merges smoothly into the other because both the robot itself slows down a little and the orange robot is also slowing down and turning a little to make room. In the table it can be seen that the

computation time for controlling collisions has gone up quite a bit since five robots now need to be checked instead of one or two as in the previous two scenarios. Otherwise the computation times are sufficiently low and the robots keep to their reference paths well while avoiding collisions with each other.

6.3.4 Two ATRs with colliding paths in a small corridor

In this scenario, a situation where two ATRs are heading towards each other similar to Scenario 6.3.2, however the paths are confined inside a small corridor meaning that there is no way to pass each other and the optimal solution would be for the robots to either back out and find another way or just stop. Figure 6.4 shows the trajectories and velocities of the robots and Table 6.4 show the gathered data from the simulation.

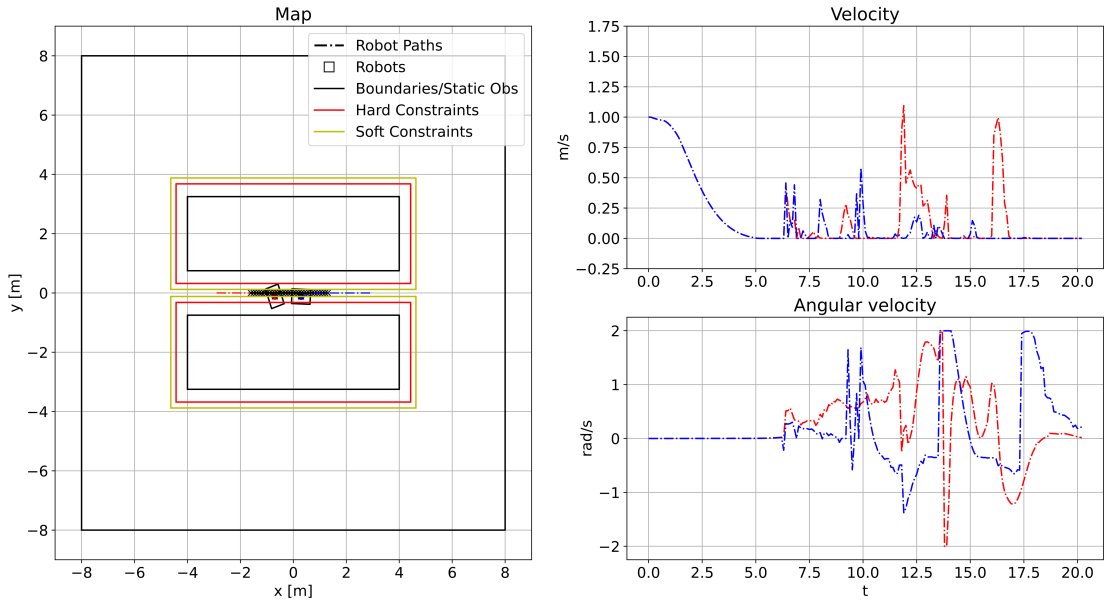


Figure 6.4: Trajectories and velocities for the ATRs in scenario 6.3.4.

Table 6.4: Gathered data from simulation of scenario 6.3.4.

Average Computation Time [ms]	6.16
Max Computation Time [ms]	66.66
Variance Computation Time [ms]	8.85
Average Distance From Ref [m]	0.03
Max Distance From Ref [m]	0.38
Variance Distance From Ref [m]	0.07
Closest Static Obstacle [m]	0.20
Distance To Hard Constraints [m]	0.20
Closest Robot [m]	0.14
Closest Dynamic Obstacle [m]	-
Robots Stopped	46
Mean Time For Generating New References [ms]	0.32
Max Time For Generating New References [ms]	0.77
Mean Time for Checking Collisions [ms]	2.15
Max Time for Checking Collisions [ms]	6.57

The figure shows that the robots in the same way as in scenario 6.3.2 stops for a while when they come close to each other, however after a small amount of time the robots starts trying to find ways around each other and the solver generates very jerky inputs signals which can be seen in the velocities. The robots does however never collide with each other or the obstacles which can be seen in the table, this is because the collision control stops the robots 46 times which also can be seen in the table. The behaviour of the ATRs from the scenario is not what is wanted as it would be preferred for the robots to stop or find another way. The reason for the behaviour is that the NMPC is tuned quite aggressively to make sure that the controller really tries to find trajectories past other robots and obstacles, the controller can be tuned down to get better results for this scenario but it would at the same time worsen the results on other scenarios.

6.3.5 Five ATRs crossing, including a dynamic obstacle

A more complex situation that is evaluated is when five ATRs is crossing an open field from different directions with an added dynamic obstacle crossing the field diagonally from the bottom left to top right of the simulated area. All of the robots start at the same time meaning that the controller has to generate trajectories away from the reference path to avoid the collisions between robots and the dynamic object. Figure 6.5 shows the resulting trajectories and velocities and the gathered data is presented in Table 6.5. Since the figure only shows the final states, the dynamic object isn't visible.

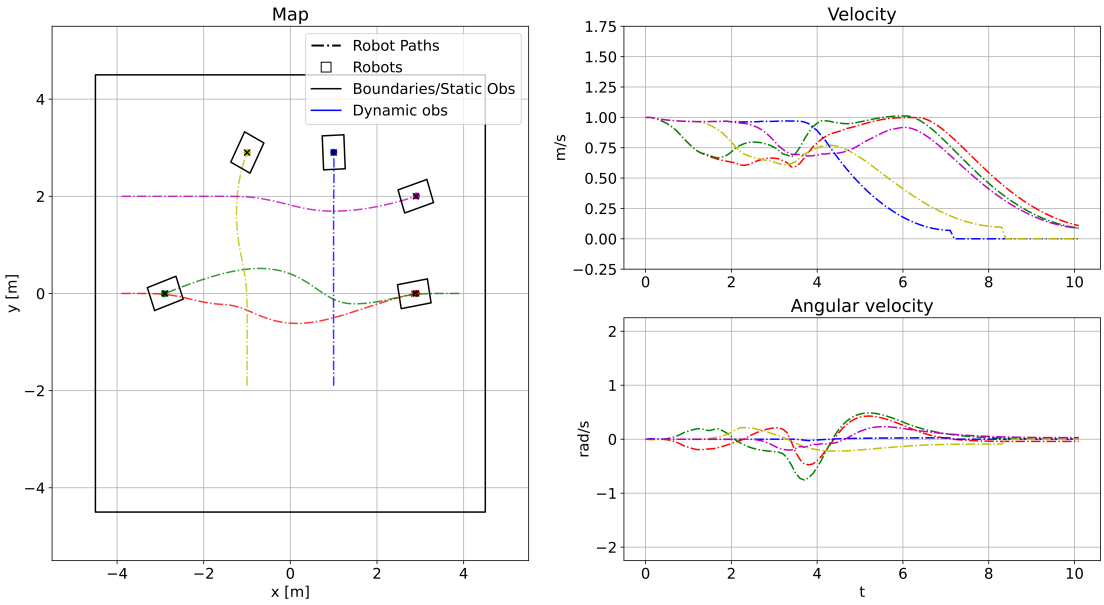


Figure 6.5: Trajectories and velocities for the ATRs in scenario 6.3.5.

Table 6.5: Gathered data from simulation of scenario 6.3.5.

Average Computation Time [ms]	11.35
Max Computation Time [ms]	236.75
Variance Computation Time [ms]	28.27
Average Distance From Ref [m]	0.03
Max Distance From Ref [m]	0.38
Variance Distance From Ref [m]	0.07
Closest Static Obstacle [m]	0.25
Distance To Hard Constraints [m]	-
Closest Robot [m]	0.22
Closest Dynamic Obstacle [m]	0.11
Robots Stopped	0
Mean Time For Generating New References [ms]	0.21
Max Time For Generating New References [ms]	0.68
Mean Time for Checking Collisions [ms]	11.86
Max Time for Checking Collisions [ms]	13.88

From the figure it can be seen that the NMPC generates smooth trajectories for all of the ATRs and they also manage to reach their goal positions. From the table it is shown that the robots avoid both each other and the dynamic object while traversing the open area. The computation time for collision control is a little bit higher than other scenarios since five robots are now used. In regards to following the trajectories the robots follow it well even though they have to avoid both each other and the dynamic obstacle.

6.3.6 Ten ATRs crossing, including a dynamic obstacle

This scenario is the same as in 6.3.5, However five more ATRs are added and a total of ten ATRs are crossing the field making it more complex and more difficult for the controller to avoid collisions between the ATRs and the dynamic object. In Figure 6.6 the trajectories and velocities for the ATRs can be seen and in Table 6.6

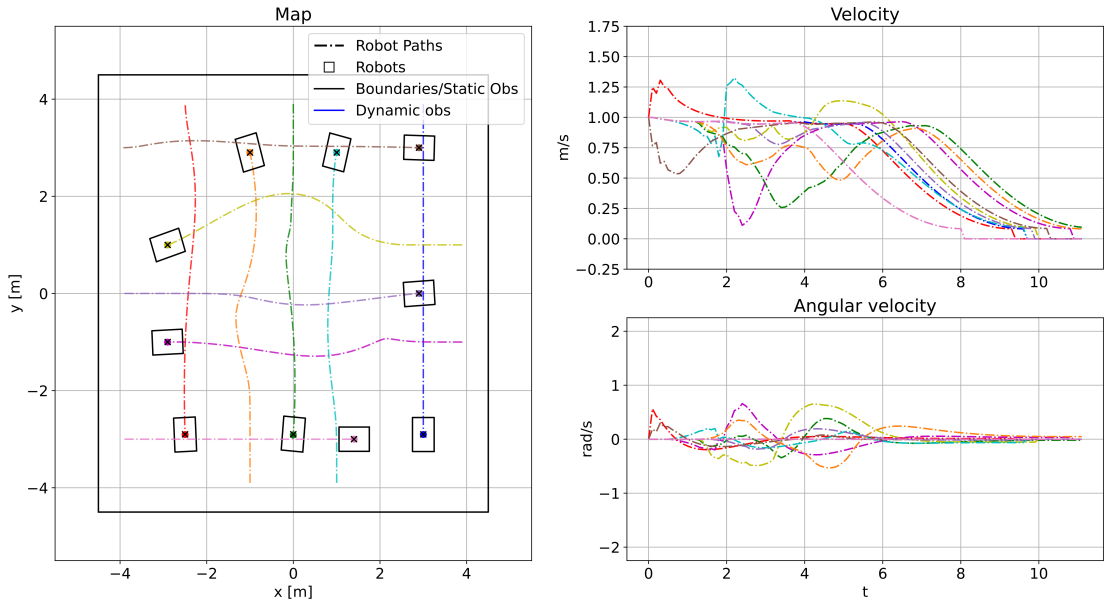


Figure 6.6: Trajectories and velocities for the ATRs in scenario 6.3.6.

Table 6.6: Gathered data from simulation of scenario 6.3.6.

Average Computation Time [ms]	9.21
Max Computation Time [ms]	230.95
Variance Computation Time [ms]	18.85
Average Distance From Ref [m]	0.03
Max Distance From Ref [m]	1.11
Variance Distance From Ref [m]	0.13
Closest Static Obstacle [m]	0.22
Distance To Hard Constraints [m]	-
Closest Robot [m]	0.23
Closest Dynamic Obstacle [m]	0.17
Robots Stopped	0
Mean Time For Generating New References [ms]	0.20
Max Time For Generating New References [ms]	0.68
Mean Time for Checking Collisions [ms]	46.03
Max Time for Checking Collisions [ms]	49.35

From the figure it can be seen that the generated trajectories are mostly smooth except for the beginning of the simulation, it is mainly the red and brown robot

having jerky inputs since the controller has a hard time deciding which direction each of them should take. This is because a distributed NMPC is used as it tries to minimize each robot by themselves which leads colliding trajectories for both robots and at the next timestep the controller takes an evasive action to avoid collision for both robots and this goes on for some timesteps until the jerky behaviour is eventually resolved. Otherwise the scenario shows good behaviour of the ATRs as they don't collide with each other or the dynamic objects and reaches their goal position which is shown in the table and figure respectively. The computation time for the NMPC solver has gone up a little but is still very low, but the collision control has gone up quite a bit since now ten ATRs and their trajectories are checked.

6.3.7 Ten ATRs crossing with four smaller obstacles in the middle

As in the previous scenario, this one also contains ten ATRs crossing a field, but without a dynamic obstacle and instead four static obstacles. This results in a situation where some of the ATRs can travel in between the obstacles and some travel outside the cluster of obstacles. A few of the ATRs original paths does however go through the obstacles and need to have their paths recalculated as presented in Chapter 4.

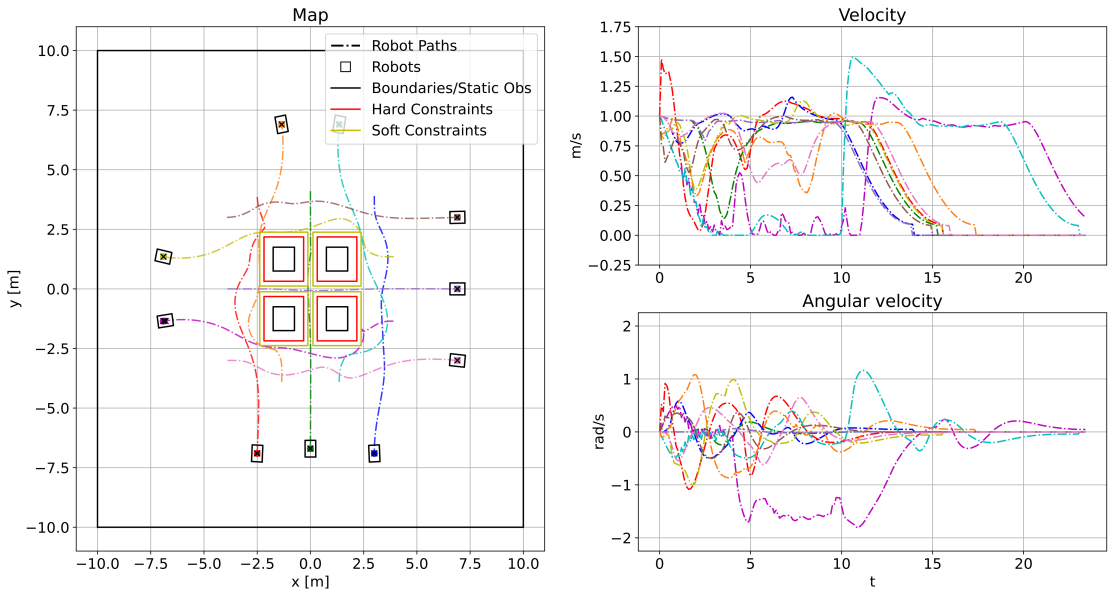


Figure 6.7: Trajectories and velocities for the ATRs in scenario 6.3.7.

Table 6.7: Gathered data from simulation of scenario 6.3.7.

Average Computation Time [ms]	10.11
Max Computation Time [ms]	257.32
Variance Computation Time [ms]	23.28
Average Distance From Ref [m]	0.06
Max Distance From Ref [m]	1.39
Variance Distance From Ref [m]	0.15
Closest Static Obstacle [m]	0.20
Distance To Hard Constraints [m]	0.20
Closest Robot [m]	0.21
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	3.03
Max Time For Generating New References [ms]	1034.02
Mean Time for Checking Collisions [ms]	47.07
Max Time for Checking Collisions [ms]	53.68

From the velocities shown in the figure it is shown that the generated trajectories are fairly smooth throughout the entire simulation, it can also be seen that after approximately five seconds the magenta colored robot stops to wait on passing robots and it also starts spinning around its center. This is probably because there is no cost on the angular velocity in the objective function but there is a cost on the angular acceleration so the optimal solution according to the objective function would be to keep the angular velocity instead of slowing down to a complete stop. Otherwise except for the spinning, the controller performs well and avoids collisions between the robots and obstacles. The reference generation works well and creates new reference paths going around the obstacles so that the controller easier can create trajectories which avoids the obstacles and leads the robots to their goal positions.

As it can be seen in the table, the computation time for the solver is still low, but the time it takes for the reference generation method to create a new reference is really high, over one second. If the system would be used as it is in the simulations it wouldn't really work in real-time as the robots would have to wait a whole second when a new reference is generated. This could be solved by having a separate computation thread that is checking further ahead to be able to generate the new paths before they are needed as inputs to the NMPC.

6.3.8 Ten ATRs crossing with five smaller obstacles in the middle

A similar case to scenario 6.3.7 is also used to evaluate the controller and implementations but an additional obstacle is added to the middle. This leads to most of the ATRs having to recalculate their paths and a lot of colliding trajectories around the cluster of obstacles. The plot containing the trajectories and velocities of the ATRs can be seen in Figure 6.8 and the gathered data is shown in Table 6.8.

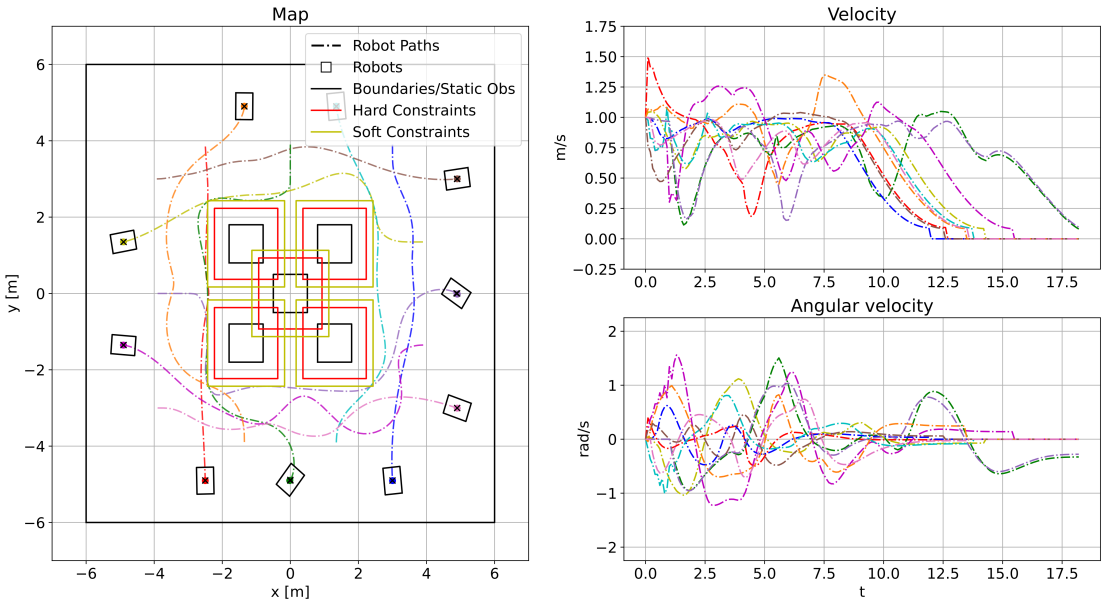


Figure 6.8: Trajectories and velocities for the ATRs in scenario 6.3.8.

Table 6.8: Gathered data from simulation of scenario 6.3.8.

Average Computation Time [ms]	11.15
Max Computation Time [ms]	235.17
Variance Computation Time [ms]	24.19
Average Distance From Ref [m]	0.08
Max Distance From Ref [m]	0.77
Variance Distance From Ref [m]	0.15
Closest Static Obstacle [m]	0.36
Distance To Hard Constraints [m]	0.20
Closest Robot [m]	0.23
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	4.20
Max Time For Generating New References [ms]	837.50
Mean Time for Checking Collisions [ms]	48.76
Max Time for Checking Collisions [ms]	78.14

The resulting trajectories are fairly smooth as in scenario 6.3.7 but they are slightly better as there are no robots that stops and spins. It can also be seen that throughout the simulations none of the robots completely stops. Furthermore there are no collisions between the robots and they all reach their goal positions. The average computation times are kept low except for the reference generation even though it is a complex scenario with many robots included.

6.3.9 Two ATRs meeting with obstacle on the preplanned path

The presented scenario is fairly simple and mostly aims to evaluate the path generation from 4. The two ATRs travel in opposite directions with a cluster of obstacles in the middle, meaning that a new reference trajectory has to be calculated and the ATRs need to avoid each other on the edge of the obstacles.

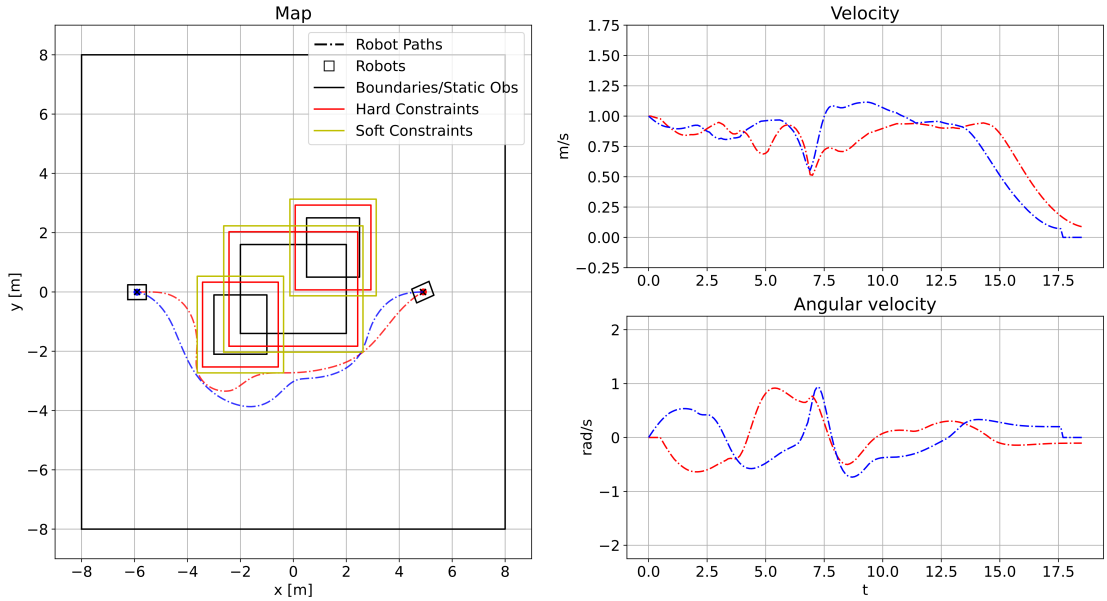


Figure 6.9: Trajectories and velocities for the ATRs in scenario 6.3.9.

Table 6.9: Gathered data from simulation of scenario 6.3.9.

Average Computation Time [ms]	13.62
Max Computation Time [ms]	229.80
Variance Computation Time [ms]	35.12
Average Distance From Ref [m]	0.07
Max Distance From Ref [m]	0.95
Variance Distance From Ref [m]	0.18
Closest Static Obstacle [m]	0.36
Distance To Hard Constraints [m]	0.20
Closest Robot [m]	0.47
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	6.99
Max Time For Generating New References [ms]	1084.53
Mean Time for Checking Collisions [ms]	2.24
Max Time for Checking Collisions [ms]	4.12

The figure shows that the robots have smooth trajectories and from the table one can

see that it manages to avoid collisions both between each other and the obstacles. The time it takes to generate a new path is very high as in other scenarios where it is used and not suitable for real-time applications as it is implemented now.

6.3.10 One ATR going around tight corners

This case aims to evaluate a how well a robot is able to follow a reference path while traversing between tight obstacles and sharp corners. The lower most obstacle also lies on the initial reference path, so a new one has to be generated around it. In Figure 6.10 the trajectory and velocity of the ATR is shown and Table 6.10 contains the data gathered from the simulation.

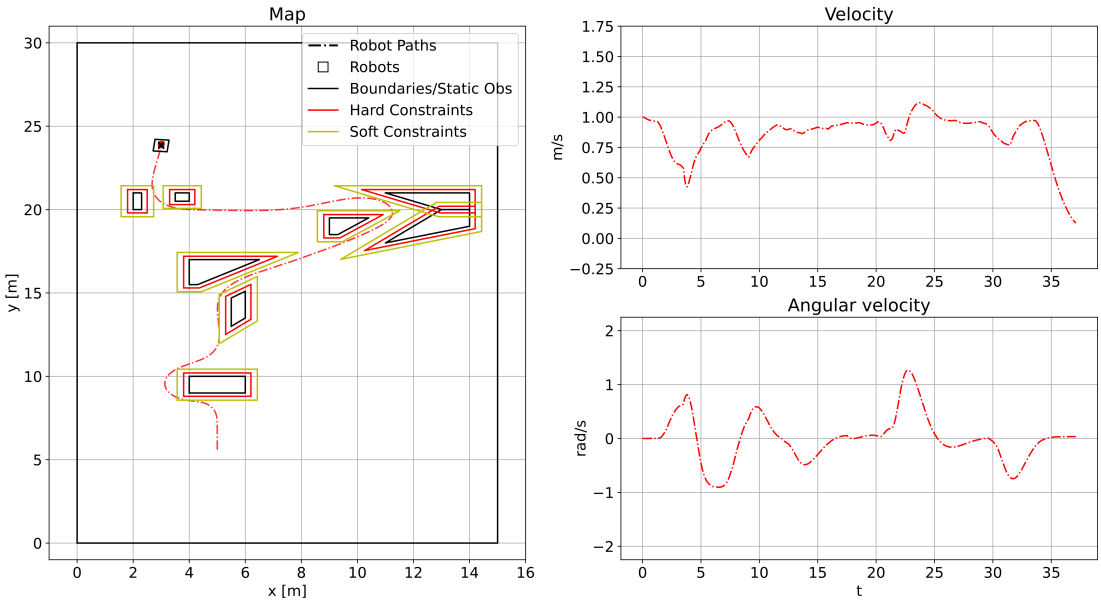


Figure 6.10: Trajectories and velocities for the ATR in scenario 6.3.10.

Table 6.10: Gathered data from simulation of scenario 6.3.10.

Average Computation Time [ms]	11.42
Max Computation Time [ms]	245.25
Variance Computation Time [ms]	25.29
Average Distance From Ref [m]	0.04
Max Distance From Ref [m]	0.45
Variance Distance From Ref [m]	0.08
Closest Static Obstacle [m]	0.11
Distance To Hard Constraints [m]	0.09
Closest Robot [m]	-
Closest Dynamic Obstacle [m]	-
Robots Stopped	0
Mean Time For Generating New References [ms]	8.81
Max Time For Generating New References [ms]	1038.31
Mean Time for Checking Collisions [ms]	0.94
Max Time for Checking Collisions [ms]	2.48

From the figure it is shown that the robot manages to avoid the static obstacles and reach the goal position, it is also seen that in the sharp corner the robot overshoots the reference quite a lot and it also cuts the soft constraints a lot on one of the final obstacles. In the table one can see that the reference is followed well most of the time but it deviates 0.95 m in the sharp corner which isn't really necessary. One can also see that even though the soft constraints are cut by a lot in one of the final obstacles, the robot never enters the hard constraint of the obstacle. The average computation time is relatively high even though only one ATR is used and this might be because it has to navigate through tight spaces making it harder for the NMPC solver to find a trajectory that avoids the obstacles.

7
Discussion

This chapter discusses the different implementations made in the thesis and how the controller performs in the different use cases. The research questions from the Introduction are answered and finally some recommended improvements/questions that needs more work/research is presented.

7.1 Non-Linear Model Predictive Control

From the results it can be seen that the robots quite often cut the corners of the soft constraints on the static obstacles when turning close to it. This is due to the fact that using the current obstacle representation, the cost for being inside the corners is very small and thus it is the optimized solution considering the objective function. At first, the NMPC only considered soft constraints, however as it was desired to make sure that the robot never collided with the obstacles hard constraints were used instead. The hard constraints worked and made sure that the robot didn't cut any corners but they instead resulted in very bad trajectories as the controller had a hard time finding good solutions when operating near the constraints. Because of that, it was decided to use two paddings on the static obstacles, one for the hard constraints to make sure the robots doesn't collide with the obstacles and one for the soft constraints to make the problem easier to solve for the controller and steer the robot away before it can come close to the hard constraints. This dual padding is only used on the static obstacles currently but should probably be included on the other types of obstacles as well to make the solution safer from collisions. When using this double padding, the obstacles does however take up more space which might lead to paths being blocked if they are tight.

When using only one tracking error point, the reference tracking isn't optimal which can be seen from various of the presented scenarios, especially 6.3.10. It does however make it so that it is easier for the controller to find ways around other obstacles and make sure that the robot reaches its end goal instead of stopping in front of other robots. To get better reference tracking, it is possible to increase the number of tracking points along the reference path which is used as input to the NMPC. The downsides of doing this is that the computation time for the solver does increase quite a bit and it becomes harder for the controller to find trajectories around obstacles. As a compromise one can use adaptive weights that uses a cost both on the tracking error along the reference path and a cost on the final point on the reference path. One can then lower the cost on many of the points and keep the cost on the

final point if a possible colliding trajectory is detected. This was tested shortly and showed some promise but no good solution was found which is why it isn't presented.

As it is with the current weights the controller is tuned more aggressively to make sure that the robots reaches their goal positions. This is the reason why scenario 6.3.4 performs very bad. This could be solved by tuning the controller to be more cautious, mainly by decreasing the cost on s (Setting a value closer to zero). By doing this the controller instead selects a closer point on which the tracking error is calculated which leads to the solver not focusing so much on driving the robots forward. This would be good for 6.3.4, but the opposite would happen in scenario 6.3.2 where the robots would stop in front of each other instead of finding a way past each other and towards the goal. There are also some other scenarios in which this change would lead to "worse" solutions, for instance it would also lead to robots stopping when they are meeting around a corner instead of going past each other.

Overall the NMPC performs well in all of the presented scenarios, with the main exception being in scenario 6.3.4, and the fact that the robot doesn't follow the reference to well around corners. The computation times for the solver are all low enough to be implemented in a real-time scenario, however there does exist situations where the solver time goes up quite a lot which can be seen in all the tables. This can be reduced as one can set a time limit for the PANOC solver but it is a bit unclear what the results would be, but since these occurrences are very rare the results would probably be similar. The controller does also generate collision free trajectories for most cases which could be improved to all cases by tuning it but that would also create some scenarios where the robots wouldn't be able to reach their goal positions.

7.2 Generating references

The method for generating new reference paths around static obstacles works well and helps the controller create trajectories leading to the goal position for the robots. In the results it is implemented such that an entire path around the obstacle is created at the same time which leads to the very high computation times shown. One can also implement it in a way that it only calculates the path 20 timesteps ahead which is then directly used as input to the NMPC which lowers the computation time for one timestep. Using the later mentioned method does however still take some time and when looking overall, the total time is lower when calculating the entire new path once instead of smaller segments multiple times. To use the current implementation in real-time it would be necessary to look further ahead than just 20 timesteps when checking if the original reference path collides with a static obstacle and then calculate the new reference path separately on another thread. If this wouldn't be done the implementation wouldn't be feasible as the controller then would have to wait for approximately one second before it can be called on again.

In the current implementation, the variables c^{rep} and ρ are set to 1.1 as it is desired to not generate trajectories which is directly along the edge of the obstacles. This

makes it easier for robots along the edges of obstacles to pass each other as it gives a bit more room for the robots to veer off towards the obstacle. This can also come with some problems, if the variables are set too high or obstacles that aren't touching each other are to close the resulting generated path converges badly and doesn't give satisfactory results.

One possible improvement that was attempted is to also generate new paths around other robots and dynamic objects but since they are moving it is a lot harder and no satisfactory solution to it was found. If this were to be done, the controller could possible be tuned to only need to follow the reference path well as the path generation would make sure that the path is collision free. There is no indication if this would be possible to do in real-time as the computation time is quite high but it is lower when paths are calculated around smaller obstacles such as robots.

7.3 Check collision

The implementation of the check collision algorithm uses the polygons as limits for when a robot needs to be stopped or not because when the paddings of the obstacles was used, the algorithm stopped the robots too often even if it wasn't really necessary. This was because the solver often generates trajectories which cuts the corners or edges of the paddings. A better solution might be to instead use multiple paddings as the static obstacles where one is used in the NMPC and one is used as a limit for how close the robot can be and used in the check collision algorithm.

From the results it is shown that the check collision algorithm isn't really needed most of the times except for scenario 6.3.4 but the controller can also be tuned so that it isn't necessary in that scenario either. The algorithm was first implemented when another NMPC formulation was used but since the s variable was introduced, the algorithm has become a bit redundant. With that being said it might still be a good idea to use it as the computation times are low and and there might exist situations where the solver doesn't converge to collision free trajectories.

7.4 Research questions

7.4.1 How can trajectories leading to a collision be detected, and in which ways is it then possible to avoid or lessen the damage?

By using the check collision algorithm it is possible to detect collisions with the help of the predicted states of the robots and the algorithm has a low enough computation time to be applicable in real-time scenarios. The algorithm can be configured in a way that it first slows the robots down with a multiplier depending on how many timesteps away a collision is detected and then completely stops the robot if the distance is lower than a selected threshold.

Furthermore, the incorporation of the optimization variable s helps because when it is used, the tracking error isn't always calculated at the furthest point on the reference. This helps because then the objective function doesn't get as high cost on the tracking error and doesn't force the robot through obstacles to get the lowest cost. An example of the check collision algorithm working can be seen in 6.3.4.

7.4.2 How does the complexity of the NMPC problem scale with increasing numbers of ATRs, measured in mean and variance of calculation time?

When comparing the computation times for the solver in the different scenarios presented, the average computation time for each ATR stays fairly similar and only differs by approximately 6.4 milliseconds between the fastest and slowest scenario. There doesn't seem to be any correlation between more ATRs and higher computation times and instead the computation times depends more on the specific scenario the robots are in. In regards to the variance of the computation times the same conclusion can be drawn that it depends more on the specific scenario than the number of robots. To conclude on the computation times for the NMPC solver, one can say that the total time for all ATRs scale linearly with respect to the number of robots that are used in the simulation.

The computation times for the check collision algorithm does however scale very quickly with regards to the number of robots used ranging from a maximum of 6.31 milliseconds when one robot is simulated to 78.14 milliseconds when ten robots are simulated. To counteract this it would be good to develop some filter to determine which robots and obstacles that should be included in the calculations. This filter would probably be good for the NMPC as well if many more robots are supposed to be controlled.

7.4.3 How can a new reference trajectory be generated in the case of obstacles causing unfeasible or bad trajectories

When using the proposed method presented in Chapter 4 the resulting generated new reference paths follows the outline of the obstacle and can handle a variety of cases. The issue with it is how it should be implemented in real-time applications since the computation time is fairly long. The proposed solution for using the method in real-time is to use a separate thread for these calculations and check the initial reference path further ahead than 20 timesteps so that potential obstacles are detected earlier and the new reference path is generated before it is needed as input to the NMPC.

7.5 Future work

This section covers the areas that we think would be good to further investigate.

7.5.1 Include better predictions for dynamical obstacles

Currently the dynamic obstacles are defined to have a constant velocity and direction. However, in the real-world this is more than often not the be the case. It would therefore be good to find some prediction algorithm to more accurately predict the movement of dynamic obstacles.

7.5.2 Evaluate in real ATRs

As this thesis only evaluated the algorithm in simulations, there is no effect from either noise or time delays on the results. As it is mentioned in [14] the tests on real ATRs came with some problems which is thought to be from the noise and time delays. It would therefore be good to implement in ROS and examine this further. It should be noted also that when doing some small testing with simulated noise, the NMPC wasn't as affected when using fewer tracking points. A possible solution for filtering the noise would be to implement some type of Kalman filter that is able to deal with non-linear systems such as an extended Kalman filter. Regarding the time delays coming from the computation time, there is no proposed solution.

7.5.3 Filtering ATRs

As this algorithm might be used for up to hundreds of robots simultaneously, it is necessary to filter which robots should be included as inputs to the NMPC depending on if they might interfere with the robot for which a trajectory is calculated. This would be done to reduce the complexity and in turn the computational time to be able to run the system in real-time. A simple way to do this would be to only include robots withing a certain radius. This could be expanded further to for instance check if a robot is on another side of a wall and unreachable or if the path for two robots are close but never crossing each other.

7.5.4 Defining robots and dynamic obstacles as static obstacles

In some scenarios such as in 6.3.3 it would be good to redefine one of the robots to a static obstacle, by doing this a new reference path would be generated out and around the corridor. The problem lies with which robot should be defined as static and to define under which conditions a robot should be redefined.

8

Conclusion

In this thesis a NMPC controller, path generator and collision avoidance algorithm was developed. The NMPC controller is of a distributed type and generates the trajectories for each robot individually. The controller manages to avoid collisions in most cases and has a low computation time suitable for real-time application with a linearly scaling computation time with respect to the number of robots being used. There are some downsides being that it doesn't follow the reference path too well and it hasn't been tested in real scenarios with noisy data. The path generator works well and creates new reference paths around obstacles that might appear after an initial path has been calculated. The proposed path generator can deal with single or multiple intersecting obstacles. However, it has a long computation time and some changes are necessary for it to be suitable in real-time applications. To guarantee that the robots never collide, a collision avoidance algorithm is implemented which stops the robots if necessary, the algorithm works well but the computation times scale up fast with the number of robots used and for larger operations a filter for which robots should be included is needed.

Bibliography

[1] Klara Pålsson an Emma Svärling. Nonlinear model predictive control for constant distance between autonomous transport robots. Master's thesis, 2020.

[2] 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.

[3] Martin Behrendt. Mpc scheme basic. `https://upload.wikimedia.org/wikipedia/commons/1/11/MPC_scheme_basic.svg`, 2009. This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License. To view a copy of this license, visit `https://creativecommons.org/licenses/by-sa/3.0/deed.en`.

[4] James B.Rawlings, David Q.Mayne, and Moritz M.Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[5] Albin Dahlin and Yiannis Karayiannidis. Creating star worlds – modelling concave obstacles for reactive motion planning, 2022.

[6] Shilp Dixit, Saber Fallah, Umberto Montanaro, Mehrdad Dianati, Alan Stevens, Francis Mccullough, and Alexandros Mouzakitis. Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects. *Annual Reviews in Control*, 45:76–86, 2018.

[7] F.Bertilsson, M.Gordon, J.Hansson, D.Möller, and D.Söderberg. Robot fleet collision avoidance performance comparison between distributed and centralized non-linear model predictive control strategies. Technical report, 2021.

[8] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. In *21st Benelux meeting on systems and control*, volume 11, pages 119–141. Citeseer, 2002.

[9] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. *Path Planning and Trajectory Planning Algorithms: A General Overview*, pages 3–27. Springer International Publishing, Cham, 2015.

[10] Sean Gillies et al. Shapely: manipulation and analysis of geometric objects, 2007–.

[11] Sebastien Gros and Bo Egardt. Modelling and simulation lecture notes, Jul 2020.

[12] Lukas Huber, Jean-Jacques Slotine, and Aude Billard. Avoiding dense and dynamic obstacles in enclosed spaces: Application to moving in crowds. *IEEE Transactions on Robotics*, 2022.

[13] Max Edin Jakobsson and Muhamed Faraj. Trajectory generation for atrs. `https://youtube.com/playlist?list=PLUSGWbNnUTi012TTVJxYlMTY6O5dkJAKl`, 2022.

[14] Valdemar Krona and Noa Lindén. Trajectory generation for multiple autonomous transport robots using nonlinear model predictive control. Master's thesis, 2021.

[15] Björn Lindqvist, Pantelis Sopasakis, and George Nikolakopoulos. A scalable distributed collision avoidance scheme for multi-agent uav systems. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9212–9218, 2021.

[16] Nicholas D Matsakis and Felix S Klock II. The rust language. In *ACM SIGAda Ada Letters*, volume 34, pages 103–104. ACM, 2014.

[17] Cebisile Mthabela, Daniel Withey, and Chioniso Kuchwa-Dube. Rrt based path planning for mobile robots on a 3d surface mesh. In *2021 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pages 1–6, 2021.

[18] Nikolce Murgovski. Model predictive control lecture notes, Jan 2021.

[19] Charles Richter, Adam Bry, and Nicholas Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, Cham, 2016.

[20] Ajay Sathya, Pantelis Sopasakis, Ruben Van Parys, Andreas Themelis, Goele Pipeleers, and Panagiotis Patrinos. Embedded nonlinear model predictive control for obstacle avoidance using panoc. In *2018 European Control Conference (ECC)*, pages 1523–1528, 2018.

[21] P. Sopasakis, E. Fresk, and P. Patrinos. OpEn: Code generation for embedded nonconvex optimization. In *IFAC World Congress*, Berlin, Germany, 2020.

[22] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos. A simple and efficient algorithm for nonlinear model predictive control. In *IEEE Conference on Decision and Control (CDC)*, pages 1939–1944, Dec 2017.

DEPARTMENT ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY