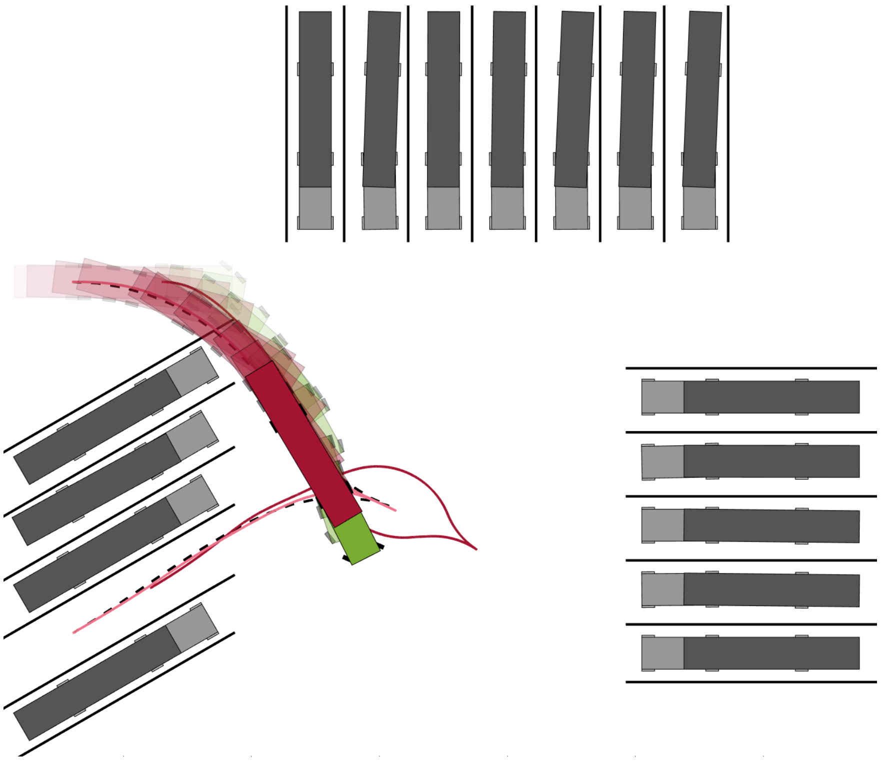
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimised reverse parking of a semi-trailer truck

Path and trajectory planning for parking in reverse with a semi-trailer truck using graph search and model predictive control

Master's thesis in Systems, Control and Mechatronics

JENS REHN
MARTIN THANDER

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Optimised reverse parking of a semi-trailer truck

Path and trajectory planning for parking in reverse with a semi-trailer truck using graph search and model predictive control

Jens Rehn
Martin Thander

Department of Electrical Engineering
Division of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2020

Optimised reverse parking of a semi-trailer truck
Path and trajectory planning for parking in reverse with a semi-trailer truck using graph search and model predictive control
Jens Rehn
Martin Thander

Supervisor: Karin Brötjefors, Aptiv Contract Services Sweden AB
Examiner: Nikolce Murgovski, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Division of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Timelapse of simulation of the trajectory planner.

Optimised reverse parking of a semi-trailer truck
Path and trajectory planning for parking in reverse with a semi-trailer truck using graph search and model predictive control
Jens Rehn
Martin Thander
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The field of autonomous vehicles is rapidly growing and is currently being extensively researched. One part of driving autonomously is to plan paths and trajectories. Paths contain the position and possibly orientation of a body, while the trajectory contains more detailed knowledge over time. This can include, but is not limited to, acceleration, velocity and steering angle. Both the path and trajectory should avoid any kind of collision.

The aim of this project is to develop an algorithm to successfully park a semi-trailer truck in reverse. Two types of path planning algorithms have been investigated, the first is graph search based, while the latter utilises model predictive control. These path planners provide the trajectory planner with a reference curve, which it is to follow. The trajectory planner is also implemented using model predictive control.

All implementations are able to solve the designed scenario. Both path planners are able to find a point of which the reversing should begin at. The trajectory planner is then able to use this reference curve to reach the provided end destination, while avoiding collision and obeying the constraints given. The trajectory is optimised with regard to time consumption and comfort, by minimising jerk and steering rate, while deviating as little as possible to the desired target pose.

Keywords: Trajectory planning, Path planning, Model predictive control, Graph search, A*, Theta*, Articulate vehicles, Non-holonomic vehicles.

Acknowledgements

We want to show our gratitude towards our examiner Nikolce Murgovski and supervisor Karin Brötjefors. They have guided us through our thesis by providing continuous support and assistance. We would also like to thank Aptiv Contract Services Sweden AB for giving us this opportunity and great hospitality during the entire project.

Jens Rehn & Martin Thander
Gothenburg, May 2020

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Aim . . . 2
- 1.3 Scope and limitations . . . 3
- 1.4 Specification of issue under investigation . . . 4
- 1.5 Thesis outline . . . 5

2 Theory **7**
- 2.1 Path planning . . . 7
- 2.2 Trajectory planning . . . 7
- 2.3 Graph search . . . 8
  - 2.3.1 Graph search as a path planner . . . 8
    - 2.3.1.1 Heuristics . . . 10
    - 2.3.1.2 A* algorithm . . . 11
    - 2.3.1.3 Theta* algorithm . . . 11
- 2.4 Model predictive control . . . 12
  - 2.4.1 A problem formulation . . . 13
  - 2.4.2 Discretisation of optimisation problem . . . 14
  - 2.4.3 Minimum time control . . . 15
- 2.5 Interior point optimiser . . . 16
- 2.6 Dynamic model . . . 18
  - 2.6.1 Detailed model . . . 18
  - 2.6.2 Simplified model . . . 20

3 Graph search path planner **25**
- 3.1 Graph design . . . 25
- 3.2 Parent tracking . . . 27
  - 3.2.1 Angular bins . . . 27
- 3.3 Node prioritisation . . . 28
  - 3.3.1 Cost function . . . 28
  - 3.3.2 Heuristics . . . 28
- 3.4 Finding neighbours . . . 29

3.4.1 Arc of sight 29
3.4.2 Neighbours in proximity of the goal node 32
3.5 Inheriting parenthood 33

4 Model predictive control 35
4.1 Interpretation of obstacles 35
4.2 Path planner 37
4.2.1 Obstacle avoidance 37
4.2.2 Final state 38
4.2.3 Optimisation problem 38
4.2.4 Post processing 40
4.3 Trajectory planner 40
4.3.1 Obstacle avoidance 40
4.3.2 Reference path 41
4.3.3 Final state 41
4.3.4 Optimisation problem 43

5 Results 45
5.1 Path planners 45
5.1.1 Graph search path planner 46
5.1.2 MPC path planner 48
5.2 Trajectory planner 52

6 Analysis 59
6.1 Path planners 59
6.1.1 Graph search path planner 59
6.1.2 MPC path planner 60
6.2 Trajectory planner 61
6.3 Conclusion 62

7 Discussion 65
7.1 Future work 66

A Alternative scenario A I

B Alternative scenario B V

List of Figures

1.1 Simplified flow of information for an autonomous vehicle. 3
1.2 Simplified flow of information, where the vehicle behaviour is being simulated. 4
1.3 Illustration of the start and end position in the given environment and an example solution for truck and trailer in red and pink respectively. 5

2.1 Illustration of an 8 point grid based graph. 9
2.2 Comparison between A* and Theta* 12
2.3 Illustration of a model of an articulated vehicle. 18
2.4 Illustration of a truck with marked measurements and position of states in the vehicle frame. 19
2.5 Illustration of a simplified model for an articulated vehicle. 21

3.1 Illustration of how points are interpolated to the closest grid point . . 26
3.2 Target heading is set to 0, and the number of bins is 4. 27
3.3 Illustration of arc calculation 30
3.4 Neighbours found by the graph search algorithm, where turning angles are limited to $\frac{\pi}{4}$. 31
3.5 Illustration of how neighbours are found when the distance to the goal node is less than one turning radius 33
3.6 Visualisation of how AoS is determined. Here, the parent node does not have a valid path, while the current node does. 34

4.1 Definition of an object when interpreted as a rotated rectangle. 36
4.2 The grey area is the safety margin of width m, the black area is the obstacle. The black line symbolises the side of the semi trailer truck that cuts through the safety margin. The red dots are the point from which the distance to the object is measured from. 37
4.3 Illustration of how the target position is soft constrained and takes the intended movement direction into account. The circles describe the target set of the horizon and the line connecting them is tangent to the reference path at $\mathbf{r}(\kappa + k_H|\kappa)$. 43

5.1 Two alternative scenarios to evaluate the path planning algorithms (a) Scenario A requires a rotation of π before parking (b) In scenario B the final orientation should be the same as the starting one. 46

5.2 Results generated by the graph search algorithm for the main scenario 46
5.3 Results generated by the graph search algorithm for alternative scenario A 47
5.4 Results generated by the graph search algorithm for alternative scenario B. 48
5.5 This path is the result of an A* search. 50
5.6 The solution to the optimisation problem (4.9) when using every 4$^{\text{th}}$ sample of an A* search as an initial guess. 51
5.7 The solution to the optimisation problem (4.9) when using an A* search as initial guess but keeping all samples as initial guess. 51
5.8 In these two plots the MPC path planner is tested on the alternative scenario A (a) MPC path planner with all samples from the A* search. (b) MPC path planner with every 4$^{\text{th}}$ sample from the A* search. . . 52
5.9 These trajectories were the result from the trajectory planner when the reference path, $(r_x(t|\tau), r_y(t|\tau))$, was generated using a) Graph search b) MPC. 55
5.10 Predictions of the horizon τ_k. The truck plans to let $(x_1(\tau_k{+}t_H|\tau), y_1(\tau_k|\tau_k{+}t_h))$ be as close to $(r_x(\tau_k{+}t_H|\tau), r_y(\tau_k|\tau_k{+}t_h))$ as possible and $(x_2(\tau_k{+}t_H|\tau), y_2(\tau_k|\tau_k{+}t_h))$ to be in a direction tangent to the reverence path. The reference path was generated using a) Graph search b) MPC. . . 56
5.11 These plots show the speed and control inputs along the parking situation. The reference path was generated using a-c) Graph search d-f) MPC. 57
5.12 Each time step is adjusted by optimising the variable $z(k|\kappa)$. These plots show the implication of the time consumption after optimising these variables. This is the result of the MPC trajectory planner using the path generated from a) Graph search b) MPC as a reference. 58

A.1 Generated trajectory for scenario A with reference from a) Graph search b) MPC I
A.2 Predicted states for scenario A with reference from a) Graph search b) MPC II
A.3 Final control signals for scenario A with reference from a-c) Graph search d-f) MPC III
A.4 Time control for scenario A with reference from a) Graph search b) MPC IV

B.1 Generated trajectory for scenario B with reference from a) Graph search b) MPC V
B.2 Predicted states for scenario B with reference from a) Graph search b) MPC VI
B.3 Final control signals for scenario B with reference from a-c) Graph search d-f) MPC VII
B.4 Time control for scenario B with reference from a) Graph search b) MPC VIII

List of Tables

I List of weightings in the objective function in (4.19) 49
II List of parameters used in the optimisation problem 49
III List of weightings in the objective function in (4.19) 53
IV List of parameters used in the optimisation problem 54

1 Introduction

Productivity and efficiency are two key components for successful companies. In order to improve these factors, automation is frequently used. It removes the human error and can in some cases use optimised strategies that are hard for humans to perceive. In the case of trucks, it can be efficient driving on roads, which is automated by cruise controllers, or it can also be in the form of parking. Today there exists automated parking assistance for cars, that either instructs the driver or has control over the steering of the car. The next step, automated valet parking is under development, which means the driver is able to leave the vehicle during the parking process. The same concept does, to the best of the authors' knowledge, not exist for reversing semi-trailer trucks.

In order to develop such features, one would need sufficient knowledge of the surrounding environment. This knowledge is usually gained by adding sensors to the truck and the trailer, but there are also possible solutions involving GPS or being provided the information by the area itself. Two of the most common sensors for this purpose on consumer vehicles are cameras and radars. In some cases, more advanced technology is used, such as Li-DAR, which grants more accurate knowledge, at the expense of an increased financial cost. So far, the amount of sensors installed on trailers is sparse.

1.1 Background

In order for autonomous vehicles to become a reality, there need to exist reliant solutions for navigation. This includes both planning of trajectories and execution of them. The amount of features available for commercial vehicles in this field is sparse. A type of commercial vehicle which usually lacks these features, is the semi-trailer truck. One reason could be the fact that it usually consists of multiple parts which are then connected, namely the truck and trailer. This means that both parts need to be compatible if sensors were to be installed on them. One truck might be used for several trailers and it would have to be consistent with all of them.

Semi-trailer trucks are mainly used to transport goods over long distances. By making this process autonomous, a driver would no longer be necessary. When arriving at a destination, the parking situation is normally done in reverse in order to connect the trailer with the loading dock. While reversing, the vision of the target destination is limited. The driver has to plan ahead and make sure that nothing enters an area, which the individual has no vision of, since it is obstructed by the vehicle and the connected trailer. This introduces a safety risk, as the driver has to assume that the situation remains the same.

In order for the autonomous parking to be performed safely, collisions need to be avoided and a certain distance to obstacles need to be kept to give a sense of security. The solution has to be reliable and trustworthy in order to be a worthy replacement of drivers with years of experience. By having knowledge of the vehicle dynamics and the surrounding, one is able to first plan a path to the goal destination, refine it into a trajectory and then begin the execution of the process.

Path planning is a widely spread method in robotics and navigation. By assuming simplified dynamics one achieves a collision free path containing information regarding position and possibly orientation from a start position to a target position. The amount of information contained in the computed path depends on the implementation of the algorithm. This varies with the type of problem but is usually tied to requirements and some optimisation criteria [1].

There are ways to include simplified dynamics in the search process, but it usually needs to be refined in someway to yield a more realistic trajectory, where a trajectory contains more extensive information than a path, such as velocity and acceleration. In some cases, as in [2], a path planner in the form of a graph search is used to obtain a reference curve, which is then used in a model predictive control (MPC) framework to compute a trajectory. MPC is commonly used to solve constrained optimisation problems. It utilises a dynamic model to describe the system, which is to be controlled under the set of constraints given [3]. These properties make MPC suitable for path and trajectory planning, where constraints will be the vehicle state and dynamics, control inputs, and the boundaries set by the environment.

Trajectory planning is a more extensive description of how the movement should be executed. It contains further information about the motion, such as velocity, acceleration and jerk. The advantage of trajectory planning is that the resulting trajectory usually resembles reality more than the output of a path planner does [4]. When planning trajectories for autonomous vehicles there are several parameters to keep in mind. Avoiding collision with objects can only actively be done if their motion can be predicted with enough confidence. This includes both stationary and moving obstacles.

1.2 Aim

The task at hand is a reverse parking scenario for an articulated vehicle, which in this case is a truck with an attached trailer. The aim is to solve the problem by first finding a path between two given points in a known environment. Once the path has been planned, a trajectory should be found using the path as a reference curve. The trajectory should take a known motion model in regard and follow the dynamics of said model. The reference curve should contain information regarding positions and moving direction, where the direction is either forward or backwards, in order to indicate reversing. The resulting path will be subjectively compared to what seems like reasonable behaviour for the given scenario and how well the trajectory planner follows it. The trajectory planner will mainly be evaluated based on how well it avoids obstacles and its ability to reach the target state.

1.3 Scope and limitations

This project assumes that vital steps before trajectory planning have been done, yielding a map over the surrounding area without ambiguity, noise or moving obstacles. It is also assumed that the map perfectly describes occupied and unoccupied space together with a target position and desired orientation. Obstacles are limited to static rectangular shapes, but can have any orientation in the environment. In Fig. 1.1-1.2 a simplified overview of the navigation process for an autonomous vehicle and the scope of the project is given.

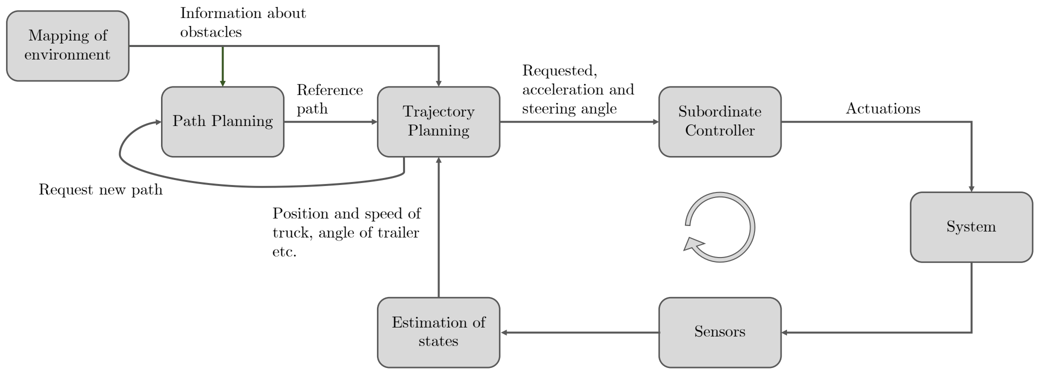


Fig. 1.1. Simplified flow of information for an autonomous vehicle.

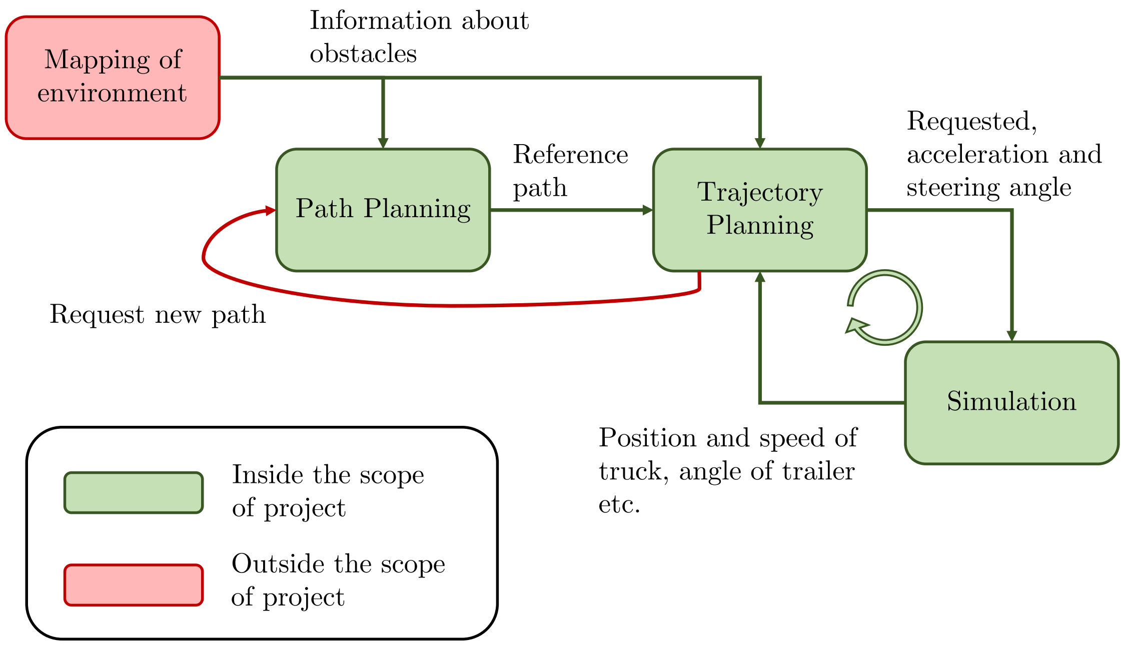

Mapping of environment
Information about obstacles
Path Planning
Reference path
Trajectory Planning
Requested, acceleration and steering angle
Request new path
Simulation
Position and speed of truck, angle of trailer etc.
Inside the scope of project
Outside the scope of project

Fig. 1.2. Simplified flow of information, where the vehicle behaviour is being simulated.

The path planning part will assume simplified vehicle dimensions, and the trajectory planning will not include setting control signals of the vehicle but rather references on states which is later passed on to a low level controller. The results of the planners will only be verified by simulation, no physical testing will be done. If the trajectory planner fails to follow the reference given by the path planner, the solution fails. A request for a new reference could be made, but it is outside the scope of this project.

All code implementations are done in MATLAB™ and CasADi, with no limitation on available toolboxes.

1.4 Specification of issue under investigation

The task which is to be solved is designed as a parking scenario for an articulated vehicle. An articulated vehicle is a vehicle with one or more pivot joints. It starts by facing forward and needs to round a corner before reversing into the designated parking space. The reversing will be through a narrow corridor, which resembles a parking spot for a trailer. The reasoning behind this design is to evaluate several challenges which are present in reversing scenarios. Obstacles may have any rotation and need to be handled accordingly, the parking spot may have a limited width, requiring high precision in order to avoid collision. With these aspects in mind, the following situation is setup, seen in Fig. 1.3.

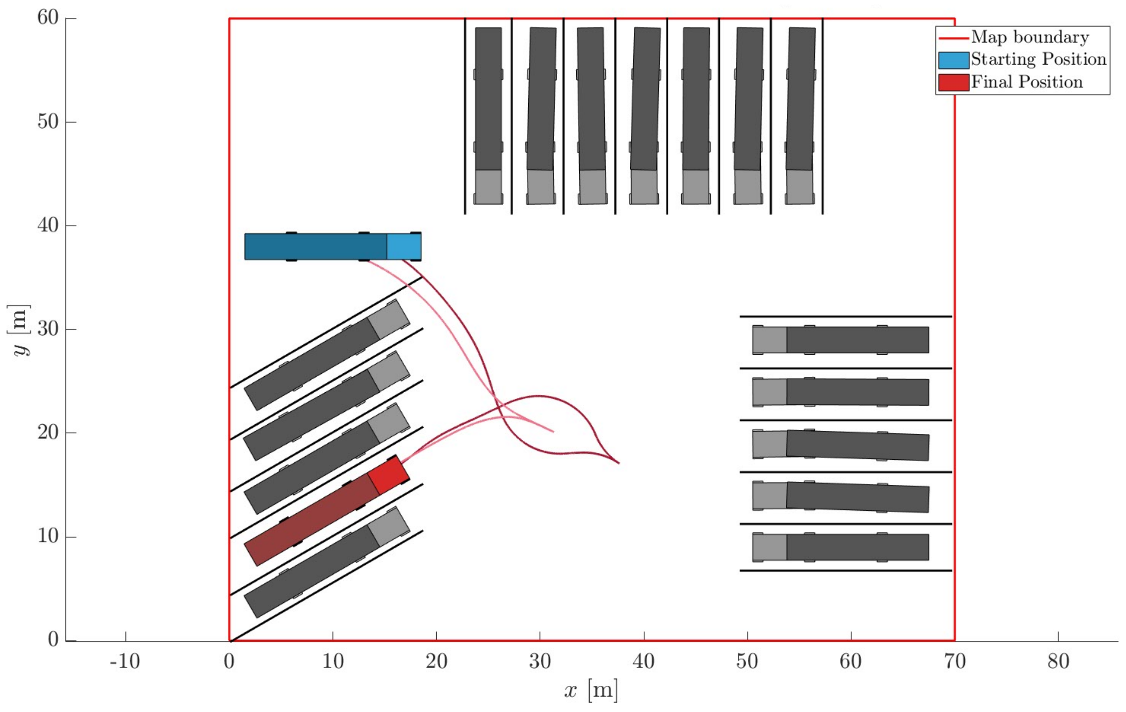


Fig. 1.3. Illustration of the start and end position in the given environment and an example solution for truck and trailer in red and pink respectively.

All obstacles are static and the positions of them are known at all times. The possible positions for the vehicle will be bounded by the extreme values of the map, resulting in a rectangular box. In addition to the environment, the initial and final positions are given as input to the path planner. If a specific orientation is wanted in the end state, this will be added as an additional input. The task is to first find a reference curve to the final position using a path planning algorithm, and then plan a trajectory from the start position to the end position. The trajectory should contain information such as, pose, steering angle, acceleration, and velocity for each computed stage.

1.5 Thesis outline

Following the introduction is a summarised description of the different subjects covered in this report. It starts out by defining the differences between path planning and trajectory planning and continues on describing graph search, model predictive control and more. All this is in Section 2. Thereafter the different approaches and chosen methods follow in Section 3-4. The results, analysis, and conclusions drawn are covered in Section 5-6. Lastly, a discussion about the entire project, the choice of methods, and its limitations, is found in Section 7.

2 Theory

This chapter introduces theory needed for implementation and provides understanding of the design process of the algorithms.

2.1 Path planning

Path planning is the process of finding the way between two points, with respect to positioning and orientation of a body [1]. No regard is taken to the velocity of said body [5]. The goal of a path planning algorithm is to find an optimal path based on some condition, usually the shortest path [6]. It could also include keeping a safe distance to objects, as in [7], to avoid collision. One of the downsides to path planning algorithms is potential incompatibility with dynamics and vehicle kinematics [8]. In some cases, these are taken in regard when planning the path [1], this comes at a cost of more complex problems and higher computational effort. For ground based vehicles, a restriction might simply be that moving laterally might be impossible without also moving longitudinally. These kind of vehicles are usually categorised as non-holonomic.

2.2 Trajectory planning

Trajectory planning is generally more extensive than path planning, since all information in a path is contained in a trajectory. In addition to the pose of the vehicle, the trajectory also contains the state of the body over a given period of time. This can include velocity, acceleration and other relevant parameters for guiding a vehicle [9]. In addition to containing all this information, the trajectory can be optimised in regard to chosen parameters, such as travel time, comfort or keeping a minimum distance to certain objects. These objects can be both stationary or moving, depending on the application [10]. In this project, the planner will have full knowledge of the dynamic model and the environment. It will be given a reference curve from the path planner and should find an optimal trajectory based on these parameters.

Trajectory planning can also include the creation of reference outputs to a lower level motion controller. These reference outputs can be requested set-points or states, such as, velocity, acceleration, or steering angle etc. [10].

2.3 Graph search

Starting from the initial node in a graph, neighbouring nodes are gradually added to a queue of nodes to expand, which is hereafter referred to as the open list. Once a node has been fully expanded, it will be added to the closed list, since there is no information to be added. The closed list is used to keep track of nodes which are no longer necessary to visit [11].

The criteria of what considers a neighbouring node varies from different search algorithms and graph structures. The connection between two nodes is usually called an edge. When a neighbour is connected to the current node, the current node will be the parent of that neighbour. By iteratively checking parenthood from the goal node, one will find the computed path starting from the initial node to the final one.

Once the goal destination has been found, the cheapest path to reach it, in regard to some measurement, is returned. The cost of reaching a node n from another node m is calculated as follows

$$\text{cost}(n) = \text{cost}(m) + C(n, m), \tag{2.1}$$

where C is the cost function, and the cost of each node is the accumulated cost from the start node.

2.3.1 Graph search as a path planner

One common method to plan paths is to apply a search algorithm on a grid based graph to find a valid, and often the shortest, path. The graph will contain information of position and occupancy, where occupancy describes objects. An occupied grid point is represented by the value 1 and a free point will have the value 0. The node connection will depend on if there is a valid traversable path between them in the environment described by the graph. This enables the applied search algorithm to take objects in regard and actively avoid them since an occupied grid point is untraversable [12]. Doing so would lead to illegal paths and would be a poor estimation of a reference curve to the trajectory planner. In addition to avoiding collision the chosen graph search algorithm might need to take dynamics of the vehicle in regard [13]. Too sharp turns will be impossible for some, since long vehicles will in general have a larger turning radius, potentially resulting in corners being cut or the estimated path being too narrow.

The downside to grid based graphs is the angular limitations created by the environment. An 8 point grid based graph has angular connections of $\frac{\pi}{4}$ multiples, where the 8 indicates how many nodes are considered to be adjacent to the current node. This means that any path generated by the search algorithm is restricted by this movement, unless neighbours are found in another way or the path is post processed to achieve smoother paths [14]. An example of how an 8 point grid can be structured is shown in Fig 2.1.

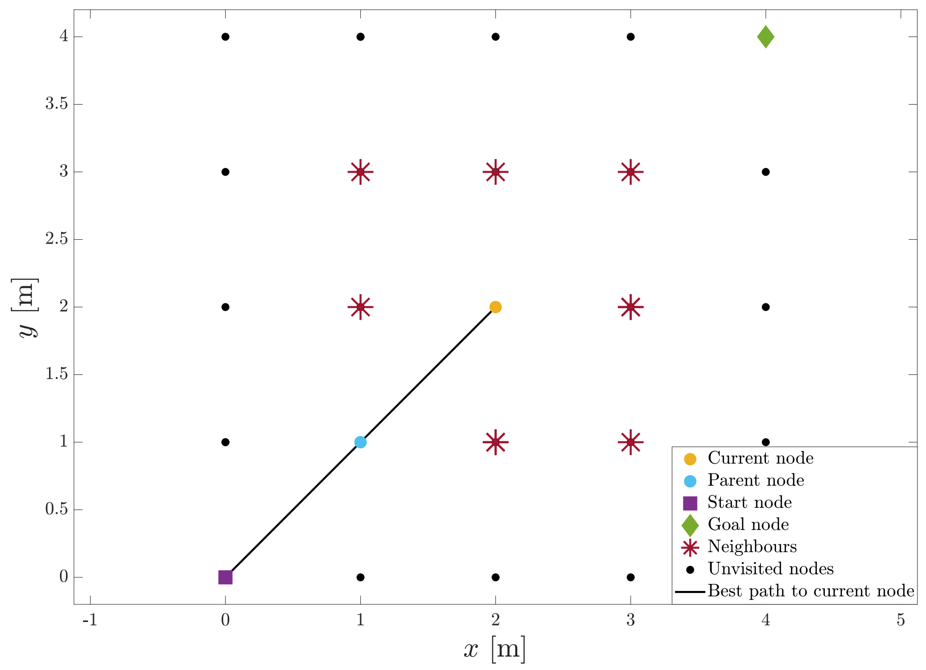


Fig. 2.1. Illustration of an 8 point grid based graph.

Since most graph search algorithms use a point mass to represent the vehicle while performing the search, generation of paths close to obstacles is a common issue. One simple way to take on this problem is to pad all obstacles, making them larger in the search process than they actually are [1]. The resulting path could still have sharp turns, but now corners of the path can be cut without risking collision, assuming each object was padded sufficiently. Another common approach is to apply configuration space, which is when the objects are padded by half the width of the robot. This is however less suitable for oblong or asymmetrical vehicles, as the width varies [15]. In the case where one might need to pad more than half the width of the vehicle in order to achieve wide enough turns, it could result in removing narrow paths that would otherwise lead to shorter paths, or being unable to find the goal node, since it may have been padded shut.

This can be handled by allowing the graph to contain information about possible orientations in the corresponding positions [16]. Doing so would result in a three dimensional graph, where different rotations in each position affect how the objects are interpreted.

There already exists several search algorithms for graphs, such as A* and its derivatives. Theta* is one of the algorithms that are based on A* and has the advantage of creating even shorter paths. It does so by not restricting the path to the edges of the graph and can instead have any angle between two nodes [17]. Both A* and Theta* algorithm are not guaranteed to find the optimal trajectory, as they tend to get stuck in a local optima. However, they will find the shortest path given consistent heuristics in a grid based graph. This path is usually not suitable for non-holonomic vehicles, as their dynamics are limited. This results is in need of post processing or modification before a generated trajectory resembles the motion of a vehicle. By including turning radius of the vehicle in the graph search, one might avoid finding infeasible trajectories due to the kinematics. This will reduce the amount of post processing needed in order to obtain a feasible trajectory [18].

Depending on how many parameters are taken in regard when planning the path, the computation time for a graph search can become high [17]. In situations where the environment might change as time passes, due to moving objects or new information being obtained this trait might hinder performance substantially, as the path would need recalculation continuously.

Previous approaches to finding paths better suitable for non-holonomic include setting a restriction on the angular rate as in [19], which affects how far the planned path need to be from objects. Since the resulting path will be in the shape of an arc, it would intersect with the corner of the object if the vehicle is too close to the edge when beginning to turn. This can be counteracted with the configuration space approach described in the third paragraph.

2.3.1.1 Heuristics

In graph search theory, a heuristic is an estimate of the distance from the current node to the goal node. This information is used in order to enable prioritisation of nodes that are more likely to be part of the optimal path, which means the search is directed towards the goal. This approach is known as an informed search. In order to achieve an informed search the heuristic has to underestimate or perfectly estimate the distance to the goal node. If the heuristics overestimate the cost, it will mislead the search and might increase the total search time compared to randomly expanding in all directions. A heuristic which never overestimates the step cost between two nodes is denoted as consistent [11]. A consistent heuristic $h(n_i)$ is defined as

$$h(n_i) \leq d(n_i, n_j) + h(n_j) \qquad \forall (n_i, n_j), \tag{2.2}$$

where d is the length of the edge between the two nodes n_i and n_j. This results in nodes being expanded in a prioritised order based on the current estimate on the remaining distance to the goal node. By performing an informed search with a consistent heuristic, one also achieves the optimal solution. If the heuristic only underestimates the distance left to the goal node, it is instead referred to as admissible. A consistent heuristic is always admissible, but the opposite is not necessarily true. If the heuristic is admissible but not consistent, finding the optimal solution can only be ensured if nodes are allowed to be visited multiple times. This means that when a node is rediscovered with a lower cost than was previously estimated, it is removed from the closed list and reevaluated with the new information [11].

2.3.1.2 A* algorithm

A* is a graph search algorithm commonly used in path planning; it is a best first search using a graph with known weights [20], which means it prioritises nodes with the highest potential by using heuristics. In a grid based graph, the heuristics are usually described by the remaining distance to the goal node. This is usually the euclidean, diagonal or manhattan distance, depending on the graph structure [21]. The cost of each node is based on the travelled distance it took to get there from the initial start position [22]. This cost is used in combination with the heuristics to determine in which direction to expand the search [7]. Given a consistent heuristic, A* is guaranteed to find the optimal path in a graph without evaluating each node more than once [23]. When applying A* to an 8 point graph its search direction is limited to multiples of $\frac{\pi}{4}$, as connections are only made to the 8 adjacent points, hence the name. This results in A* finding the shortest path in the grid, but this is not necessarily the shortest path available, as there may be shorter connections that are not multiples of $\frac{\pi}{4}$ [24]. These sharp turn angles are not suitable for non-holonomic vehicles, as their dynamics are restricted.

There are versions of A* which handle vehicle dynamics, such as Hybrid A* used in [6]. Here, the heading, in addition to position is used to project possible motion primitives in order to find new nodes. The motion primitives are based on the vehicle dynamics and the starting heading is given as input. This means that for each node, the heading is tracked in order to obtain a curved path instead of the previous limit of $\frac{\pi}{4}$ turns.

2.3.1.3 Theta* algorithm

Theta* is an expanded version of A*, where inheriting parenthood is allowed. When visibility between the current node and a neighbour is evaluated, visibility between the neighbour and the parent of the current node is also determined. If the current node is visible from the parent, the parenthood is assigned to the parent node instead of the current node. This changes the restricted movement in A* where it only checks adjacent neighbours resulting in movement of $\frac{\pi}{4}$ multiples. This makes Theta* an any-angle search algorithm as long as no constraints are set on changes of heading in an 8 point grid based graph. It also enables finding shorter paths, since nodes can be skipped if the cost is deemed to be smaller as seen in Fig. 2.2 [17][25].

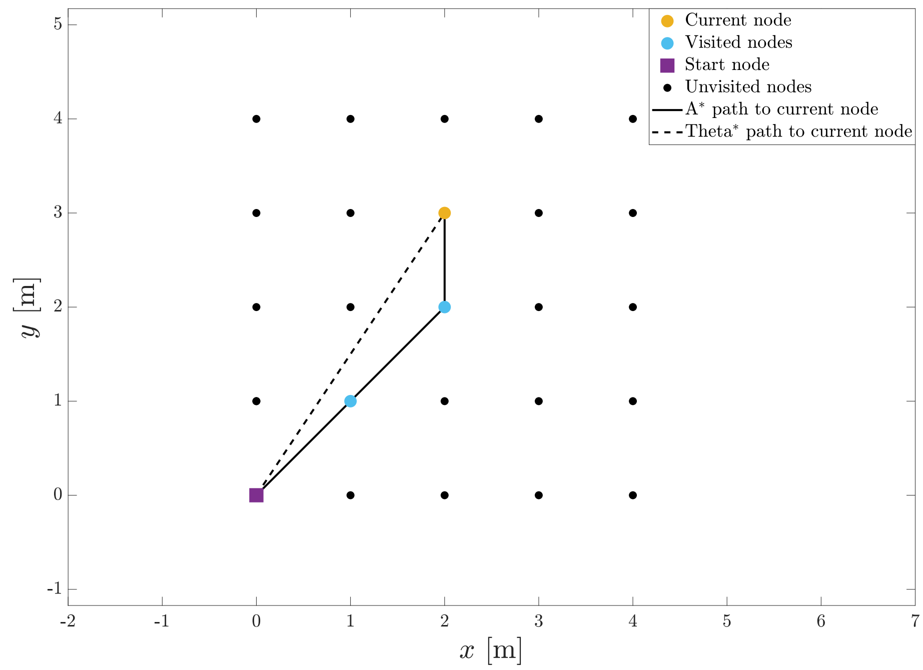


Fig. 2.2. Comparison between A* and Theta*

One algorithm to determine visibility between two points in a grid based graph with obstructions present is Bresenham's line algorithm [26]. It is commonly used in computer graphics to draw approximations of straight lines between two points. By usage of integer addition, subtraction and bit shifting it is computationally efficient, and suitable for this purpose, since the graph information is contained in the index of each point. Estimating a straight line between two points will return all the occupancy information needed to determine line of sight.

2.4 Model predictive control

Model predictive control is an optimal control strategy, similar to Linear Quadratic (LQ) control, but instead has a finite horizon. It can also handle non-linear time invariant systems as well as constraints on states and control signals as opposed to an LQ controller. It does so by solving an optimisation problem at each time step and outputting the optimal control signals for the specific objective [27]. A generic optimisation problem will be presented in Section 2.4.1-2.4.3.

Although, before proceeding, the notation of states and variables will be established. The sampling variable is denoted t and can take values between 0 and T. Each prediction horizon (as far in the future the controller predicts the states at each time step) is denoted τ and is of length t_H. The individual horizons τ can take discrete values between $[0, T - t_H]$, i.e. $\tau \in \{0, \Delta t, 2\Delta t, ..., T - t_H\}$.

With these variables, the predicted state $\mathbf{x}$ at some time t, and the actual state at the same time t, can be distinguished. A state $\mathbf{x}$, that is predicted in the horizon

τ, and represents the state at time $t \in [\tau, \tau + t_H]$, is denoted $x(t|\tau)$. When the problem is discretised, the same state at the same point in time is denoted $x(k|\kappa)$ where $k \in \{\kappa, \kappa+1, ..., \kappa+k_H\}$ and $\kappa \in \{0, 1, ..., K - k_H\}$.

2.4.1 A problem formulation

Given a state $\mathbf{x}_f \in \mathcal{X}_f$ as a target final state, the purpose of this controller is to keep the states $\mathbf{x}(t)$ as close as possible to some reference $\mathbf{x}_r(t)$. It should do so while minimising usage of control inputs and time control, i.e the time it takes to complete the scenario. Let $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$ be the model of the system to be controlled, where $\mathbf{x}(t)$ is the states, $\mathbf{u}(t)$ control signals and t time. The acceptable states are those in the set $\mathcal{X}$, and valid control signals are in $\mathcal{U}$. The optimisation problem is to minimise the integral of some objective function $J(\mathbf{x}, \mathbf{u})$ under the constraints on states and control signals defined by the sets $\mathcal{X}$,$\mathcal{U}$. To relax the problem, a set of acceptable final states $\mathcal{X}_f$ can be defined in which the final state $\mathbf{x}(T|\tau_f)$ has to be a member of, where τ_f indicates the final horizon. This final state can be associated with its own cost $J_f(\mathbf{x})$ that penalises the difference between the state $\mathbf{x}(T|\tau_f)$ and the target final state $\mathbf{x}_f(T|\tau_f)$. The objective function is sometimes referred to as the cost function and is used to penalise certain states and/or control signals.

The time control can be minimised in two ways. Either as a running cost, minimising $\int t\ dt$, or as a final cost, minimising $t_H(\tau)$. In this project the time control will be minimised by considering it as a running cost. How this is implemented is described in 2.4.3. Since the goal is also to minimise the use of control signals while keeping the states as close to the target states as possible, the objective function can be stated as

$$\begin{aligned}
&\int_{\tau}^{\tau+t_H} J(\mathbf{x}, \mathbf{u}, t)dt + J_f\Big(\mathbf{x}(T|\tau_f)\Big) \\
= &\int_{\tau}^{\tau+t_H} \|\mathbf{u}(t|\tau)\|_Q^2 + \|\mathbf{x}(t|\tau) - \mathbf{x}_r(t|\tau)\|_R^2 + R_t t\ dt + \|\mathbf{x}(T|\tau_f) - \mathbf{x}_f\|_{R_f}^2,
\end{aligned} \tag{2.3}$$

with some weightings Q, R, R_t. The notation $\|\mathbf{x}\|_R$ is short for

$$\|\mathbf{x}\|_R^2 = \mathbf{x}^T R \mathbf{x}. \tag{2.4}$$

The complete optimisation problem which is to be solved at each time step is formulated as

$$\begin{aligned}
\min_{x,u} \quad & \int_{\tau}^{\tau+t_H} J\Big(\mathbf{x}(t|\tau), \mathbf{u}(t|\tau), t\Big)dt + J_f\Big(\mathbf{x}(t|\tau)\Big) && \text{(2.5a)} \\
\text{s.t.} \quad & \dot{\mathbf{x}}(t|\tau) = f(\mathbf{x}(t|\tau), \mathbf{u}(t|\tau), t) && \text{(2.5b)} \\
& \mathbf{g}(\mathbf{x}(t|\tau), \mathbf{u}(t|\tau), t) \leq 0 && \text{(2.5c)} \\
& \mathbf{x}(t|\tau) \in \mathcal{X}(\tau) && \text{(2.5d)}
\end{aligned}$$

$$\mathbf{x}(\tau + t_H|\tau) \in \mathcal{X}_f(\tau) \tag{2.5e}$$
$$\mathbf{x}(\tau|\tau) = \mathbf{x}_0(\tau) \tag{2.5f}$$
$$\mathbf{u}(t|\tau) \in \mathcal{U}(\tau). \tag{2.5g}$$

The constraints are imposed for all $t \in [\tau, \tau + t_H]$ and $\tau \in [0, \tau_f]$.

Assuming that the constraints and vehicle dynamics reflect the system accurately, in addition to the weightings of the objective function being tuned accordingly, the solution to this problem will have the desired behaviour.

2.4.2 Discretisation of optimisation problem

The optimisation problem is often translated from continuous to a discrete. I.e. instead of a differential equation $\dot{\mathbf{x}} = f(\mathbf{x}(t|\tau), \mathbf{u}(t|\tau), t)$, a difference equation $\mathbf{x}(k+1) = \tilde{f}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), k)$ is constructed by approximating the integral of the differential equation between two time steps. For the approximation to be reliable, it has to capture the dynamics of the system. The more non-linear the system is and the faster the dynamics are, the more emphasis is put on the integration method. One simple integration method is the euler forward integration which looks like

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \Delta t \cdot f(\mathbf{x}(t_k), \mathbf{u}(t_k), t_k), \tag{2.6}$$

where $t_k = t(k)$.

For this approximation to capture the dynamics of the system used in this project, a short sample time needs to be used. This in turn makes the optimisation problem more computationally complex, since the number of constraints scales with the number of time steps of each horizon. In order to lower the number of time steps needed, a more accurate integration method can be used. The euler forward approximation is also called the first order explicit Runge-Kutta method. With a higher order, the accuracy increases and larger time steps can be used while still capturing the dynamics. The disadvantage is that the computational complexity increases with an increased order of integration. The fourth order Runge-Kutta method (RK4) is, in this project, regarded as a good trade off between accuracy and the computational effort it comes with. It is defined as

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \Delta t \sum_{i=1}^{4} \mu_i \lambda_i, \tag{2.7}$$

where

$$\begin{aligned}\lambda_1 &= f(\mathbf{x}(t_k), \mathbf{u}(t_k), t_k)\\ \lambda_2 &= f(\lambda_1\frac{\Delta t}{2} + \mathbf{x}(t_k), \mathbf{u}(t_k + \frac{\Delta t}{2}), t_k)\\ \lambda_3 &= f(\lambda_2\frac{\Delta t}{2} + \mathbf{x}(t_k), \mathbf{u}(t_k + \frac{\Delta t}{2}), t_k)\\ \lambda_4 &= f(\lambda_3\Delta t + \mathbf{x}(t_k), \mathbf{u}(t_k + \Delta t), t_k)\end{aligned} \tag{2.8}$$

and

$$\mu_1 = \frac{1}{6},\ \mu_2 = \frac{1}{3},\ \mu_3 = \frac{1}{3},\ \mu_4 = \frac{1}{6}. \tag{2.9}$$

If constant control signals from t_k up to, but not including, $t_k + \Delta t$, are assumed, the coefficients λ_{2-4} are instead

$$\begin{aligned}\lambda_2 &= f(\lambda_1\frac{\Delta t}{2} + \mathbf{x}(t_k), \mathbf{u}(t_k), t_k)\\ \lambda_3 &= f(\lambda_2\frac{\Delta t}{2} + \mathbf{x}(t_k), \mathbf{u}(t_k), t_k)\\ \lambda_4 &= f(\lambda_3\Delta t + \mathbf{x}(t_k), \mathbf{u}(t_k), t_k)\end{aligned} \tag{2.10}$$

The objective function $J(\cdot)$ in (2.3) is discretised in the same way into $\tilde{J}(\cdot)$

2.4.3 Minimum time control

In order to minimise the time control, the problem can no longer be expressed as a function of time. When it is discretised, the sample time will be fixed and, therefore, also the time it takes to complete the scenario. In [28, 29], a state $z(t)$ is introduced, referred to as *lethargy*, which is effectively the inverse speed. In [28] they also make a variable shift from a temporal dependency to a spatial one. In this project, it has the effect of sampling in discrete space intervals along some virtual path. The new spatial variable is denoted $\zeta(t)$ and is defined from the chain rule as

$$\dot{\mathbf{x}} = \frac{d}{dt}\mathbf{x}(t) = \frac{d}{d\zeta}\frac{d\zeta}{dt}\mathbf{x}\big(\zeta(t)\big) = \underbrace{\frac{d}{d\zeta}\mathbf{x}\big(\zeta(t)\big)}_{\mathbf{x}'}\underbrace{\frac{d\zeta}{dt}}_{1/z(\zeta(t))} = f\Big(\mathbf{x}\big(\zeta(t)\big), \mathbf{u}\big(\zeta(t)\big), \zeta(t)\Big), \tag{2.11}$$

where $\mathbf{x}'$ is the differentiated state $\mathbf{x}(\zeta(t))$ with respect to ζ. The variable $z(\zeta)$ is the inverse speed and will together with the sample length $\Delta\zeta$ control the time between two samples. From now on, the dependency of t will be omitted. By solving for $\mathbf{x}'$, the expression for the differentiated states is

$$\mathbf{x}' = z(\zeta)f(\mathbf{x}(\zeta), \mathbf{u}(\zeta), \zeta). \tag{2.12}$$

The implication of this variable shift is easiest seen when discretising the function as in (2.7)-(2.9). By defining λ_i for $i \in [1,4]$ as before the discretisation becomes

$$\mathbf{x}(k+1) = \tilde{f}(\mathbf{x}(k), \mathbf{u}(k), z(k), k) = \mathbf{x}(k) + \Delta\zeta \sum_{i=1}^{4} \mu_i z(k) \lambda_i, \tag{2.13}$$

where

$$\begin{aligned} \lambda_1 &= f(\mathbf{x}(\zeta_k), \mathbf{u}(\zeta_k), \zeta_k) \\ \lambda_2 &= f(z(k)\lambda_1 \frac{\Delta\zeta}{2} + \mathbf{x}(\zeta_k), \mathbf{u}(\zeta_k), \zeta_k) \\ \lambda_3 &= f(z(k)\lambda_2 \frac{\Delta\zeta}{2} + \mathbf{x}(\zeta_k), \mathbf{u}(\zeta_k), \zeta_k) \\ \lambda_4 &= f(z(k)\lambda_3 \Delta\zeta + \mathbf{x}(\zeta_k), \mathbf{u}(\zeta_k), \zeta_k) \end{aligned} \tag{2.14}$$

and

$$\mu_1 = \frac{1}{6},\ \mu_2 = \frac{1}{3},\ \mu_3 = \frac{1}{3},\ \mu_4 = \frac{1}{6}. \tag{2.15}$$

The product $z(k)\Delta\zeta$ has the unit seconds and therefore controls the sample time. By including $z(k)$ in the objective function of the problem, the problem also includes minimisation of the lethargy and hence the time it takes to complete each horizon. The variable $z(k)$ can also be bounded between a lower and upper value to limit the time between two samples.

The complete discretised problem is formulated as

$$\begin{aligned} \min_{x,u,z} \quad & \sum_{\kappa}^{\kappa+k_H} \tilde{J}(\mathbf{x}(k), \mathbf{u}(k), z(k), k|\kappa) \\ \text{s.t.} \quad & \mathbf{x}(k+1|\kappa) = \tilde{f}(\mathbf{x}(k), \mathbf{u}(k), z(k), k|\kappa) \\ & \mathbf{x}(k|\kappa) \in \mathcal{X} \\ & \mathbf{x}(\kappa + k_H|\kappa) \in \mathcal{X}_f \\ & \mathbf{u}(k|\kappa) \in \mathcal{U} \\ & z(k|\kappa) \in \mathcal{Z} \end{aligned} \tag{2.16}$$

where $\mathcal{Z}$ is the set of acceptable values of $z(k|\kappa)$.

The optimisation problem is then passed on to a numerical solver, one such solver is the Interior point optimiser (Ipopt).

2.5 Interior point optimiser

When solving large optimisation problems, analytic solutions are intractable. Instead a numeric solver can be used. The open-source Ipopt is a solver for non-linear and non-convex optimisation problems and suitable for this project. It solves problems in the form of

$$\mathbf{w} = [\mathbf{x}_{\kappa}^T, \mathbf{u}_{\kappa}^T, \mathbf{x}_{\kappa+1}^T, \ldots, \mathbf{x}_{\kappa+k_H-1}^T, \mathbf{u}_{\kappa+k_H-1}^T, \mathbf{x}_{\kappa+k_H}^T, \mathbf{u}_{\kappa+k_H}^T, \mathbf{x}_{\kappa+k_H+1}^T]^T \tag{2.17}$$

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{\kappa}^{\kappa+k_H} \tilde{J}(\mathbf{w}) \\ \text{s.t.} \quad & g_L \leq g(\mathbf{w}) \leq g_U \\ & \mathbf{w}_L \leq \mathbf{w} \leq \mathbf{w}_U, \end{aligned} \tag{2.18}$$

where $\mathbf{w} \in \mathbb{R}^{n_x k_H + n_u k_H}$ is the optimisation variables, $g(\mathbf{w})$ the general constraints, g_L, g_U the corresponding upper and lower bound, $\mathbf{w}_L, \mathbf{w}_U$ the box constraints for the optimisation variables and $\tilde{J}(\mathbf{w})$ the objective function. κ is the index of the first variable of the current horizon and k_H is the length of each horizon.

General constraints are those that can not be contained by upper and lower bounds. They are instead expressed as a function $g(\mathbf{w})$. The function is in turn bounded by predefined limits. E.g. the euclidean distance between a state (x, y) and a point (p_x, p_y), can not be constrained by two bounds. The constraints are formulated as

$$\begin{aligned} d_{min} &\leq \left\| \begin{bmatrix} x_\kappa \\ y_\kappa \end{bmatrix} - \begin{bmatrix} p_x \\ p_y \end{bmatrix} \right\| \leq \infty \\ d_{min} &\leq \left\| \begin{bmatrix} x_{\kappa+1} \\ y_{\kappa+1} \end{bmatrix} - \begin{bmatrix} p_x \\ p_y \end{bmatrix} \right\| \leq \infty \\ &\vdots \\ \underbrace{d_{min}}_{g_L} &\leq \underbrace{\left\| \begin{bmatrix} x_{\kappa+k_H} \\ y_{\kappa+k_H} \end{bmatrix} - \begin{bmatrix} p_x \\ p_y \end{bmatrix} \right\|}_{g(\mathbf{w})} \leq \underbrace{\infty}_{g_U} \end{aligned} \tag{2.19}$$

The IPOPT finds a solution to the problem in (2.18) by executing a line search algorithm from a user-provided initial starting point $\mathbf{w}_0 \in \mathrm{R}^{n_x}$. If one is not provided or an infeasible starting point is given, it is set to zero or moved to satisfy all constraints, depending on if zero is within the constraints or not.

If the problem is non-convex, the solution and its feasibility depend on the starting point. E.g. if the shortest path between two points is to be found, and there is an alternative to pass an obstacle either on the left or right. The initial guess of the optimal path, which is either to take left or right, limits the solution to that choice, even if that path is longer. For that reason, the initial guess needs to be as good as possible to ensure an acceptable solution. Also, the IPOPT does not guarantee a global optimal solution for a non-convex problem formulation, only a local optimum, at best, can be guaranteed [30].

2.6 Dynamic model

In this project, two different dynamic models will be investigated for the two different MPC planners. One detailed model and one simplified. They are all expressed in the time domain and are transformed as described by the sections 2.4.2 and 2.4.3.

2.6.1 Detailed model

A defining difference between a truck and a semi-trailer truck is that the latter has a pivot point. Therefore, the model needs to reflect those dynamics. The model used is a single track model with an articulate point, and based on [31]. An illustration of the model and its components can be seen in Fig. 2.3.

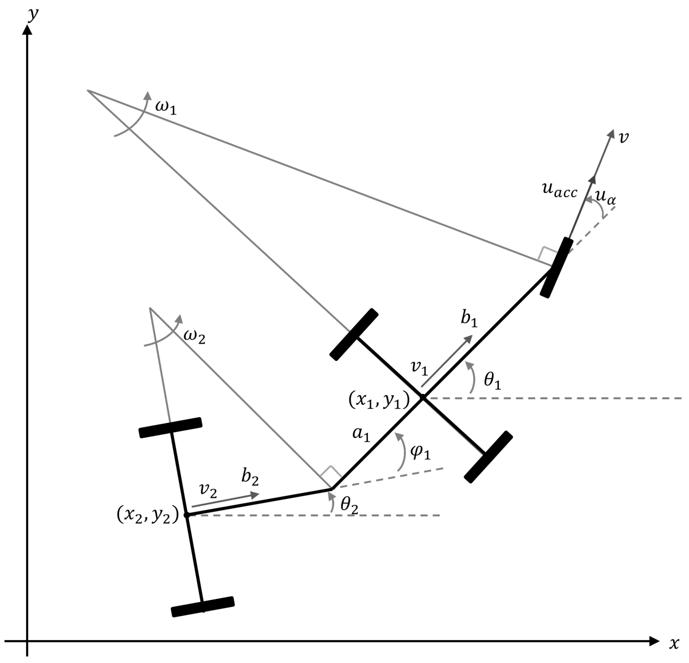


Fig. 2.3. Illustration of a model of an articulated vehicle.

The states of the model are the position of the center of the rear axle of the truck and trailer, (x_1, y_1) and (x_2, y_2), including the heading of the front part of the vehicle θ_1 and the relative heading φ_1. An illustration of a truck and the location of the states corresponding to the positions are shown in Fig. 2.4. The distance a_1 is defined as negative since the articulation point is in front of the rear axle of the truck.

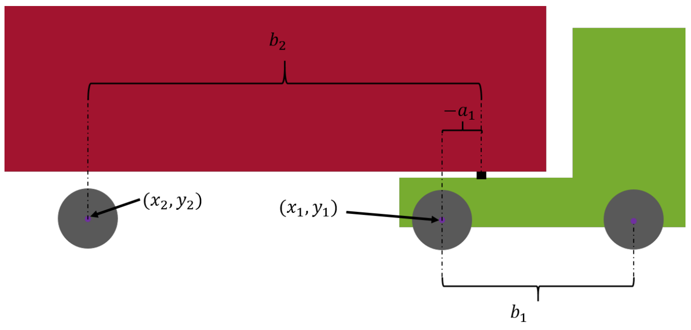


Fig. 2.4. Illustration of a truck with marked measurements and position of states in the vehicle frame.

In [31], the velocity of the front wheels is considered as a control input, which results in a jerky driving behaviour. By setting the velocity v as a state, and instead control the acceleration, the velocity is smoothly increased over time, giving a more realistic driving behavior. The other control signal is kept as the steering angle u_α. The model is also completed with two states in order to penalise steering rate and jerk, namely $\vartheta_{\dot{\alpha}} = \dot{u}_\alpha$, $\vartheta_{\text{jerk}} = \dot{u}_{\text{acc}}$. The model parameters are the distances a_1, b_1, b_2 and are designed in order to describe a common truck-trailer configuration. The model equations are

$$\dot{\mathbf{x}}(t) = f\Big(\mathbf{x}(t), \mathbf{u}(t)\Big) = \begin{bmatrix} v_1(t)\cos\theta_1(t) \\ v_1(t)\sin\theta_1(t) \\ v_2(t)\cos\Big(\theta_1(t) - \varphi_1(t)\Big) \\ v_2(t)\sin\Big(\theta_1(t) - \varphi_1(t)\Big) \\ \frac{\sin\varphi_1(t)}{b_2 + a_1\cos\varphi_1(t)} v_1(t) + \frac{b_2}{b_2 + a_1\cos\varphi_1(t)}\dot{\varphi}_1(t) \\ \omega_1(t) - \omega_2(t) \\ u_{\text{acc}}(t) \\ \ddot{u}_\alpha(t) \\ \ddot{u}_{\text{acc}}(t) \end{bmatrix} \tag{2.20a}$$

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ y_1(t) \\ x_2(t) \\ y_2(t) \\ \theta_1(t) \\ \varphi_1(t) \\ v(t) \\ \vartheta_{\dot{\alpha}}(t) \\ \vartheta_{\text{jerk}}(t) \end{bmatrix} \tag{2.20b}$$

$$\mathbf{u}(t) = \begin{bmatrix} u_\alpha(t) \\ u_{\text{acc}} \end{bmatrix}, \tag{2.20c}$$

where

$$\begin{bmatrix} v_1(t) \\ \omega_1(t) \\ v_2(t) \\ \omega_2(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos u_\alpha(t) \\ \frac{1}{b_1} v(t) \sin u_\alpha(t) \\ \Big(\cos\varphi_1(t)\cos u_\alpha(t) + \frac{a_1}{b_1}\sin\varphi_1(t)\sin u_\alpha(t)\Big)v(t) \\ \frac{1}{b_2}\Big(\sin\varphi_1(t)\cos u_\alpha(t) - \frac{a_1}{b_1}\cos\varphi_1(t)\sin u_\alpha(t)\Big)v(t) \end{bmatrix}. \tag{2.21}$$

The position of the rear axle (x_2, y_2) is fully defined by the angles φ_1, θ_1 and does not need a state of its own. This means that the model is over defined. By removing this state one obtains the following model:

$$f\Big(\mathbf{x}(t), \mathbf{u}(t)\Big) = \begin{bmatrix} v_1(t)\cos\theta_1(t) \\ v_1(t)\sin\theta_1(t) \\ \frac{\sin\varphi_1}{b_2 + a_1\cos\varphi_1(t)} v_1(t) + \frac{b_2}{b_2 + a_1\cos\varphi_1(t)}\dot{\varphi}_1(t) \\ \omega_1(t) - \omega_2(t) \\ u_{acc}(t) \\ \vartheta_{\dot{\alpha}}(t) \\ \vartheta_{\text{jerk}}(t) \end{bmatrix}, \tag{2.22}$$

where

$$\begin{bmatrix} v_1(t) \\ \omega_1(t) \\ \omega_2(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos u_\alpha(t) \\ \frac{1}{b_1} v(t) \sin u_\alpha(t) \\ \frac{1}{b_2}\Big(\sin\varphi_1(t)\cos u_\alpha(t) - \frac{a_1}{b_1}\cos\varphi_1(t)\sin u_\alpha(t)\Big)v(t) \end{bmatrix}. \tag{2.23}$$

The position of the rear axle is calculated as

$$\begin{aligned} x_2(t) &= x_1(t) - a_1\cos\Big(\theta_1(t)\Big) - b_2\cos\Big(\theta_1(t) - \varphi_1(t)\Big) \\ y_2(t) &= y_1(t) - a_1\sin\Big(\theta_1(t)\Big) - b_2\sin\Big(\theta_1(t) - \varphi_1(t)\Big) \end{aligned} \tag{2.24}$$

The constraints on steering angle, acceleration and trailer angle are

$$\begin{aligned} u_{\alpha,\min} &\leq u_\alpha(t) \leq u_{\alpha,\max} \\ u_{\text{acc},\min} &\leq u_v(t) \leq u_{\text{acc},\max} \\ \varphi_{\min} &\leq \varphi(t) \leq \varphi_{\max}. \end{aligned} \tag{2.25}$$

2.6.2 Simplified model

The simplified model is based on the detailed one but does not include an articulation point and instead of acceleration, the speed is chosen as the input. It also includes an extra state $\vartheta_v(t) = u_v(t)\cos u_\alpha(t)$ and its derivative $\vartheta_{\text{acc}}(t) = \frac{d}{dt}\Big(\vartheta_v(t)\Big)$ in order to be able to penalise longitudinal acceleration. The model then becomes

$$\dot{\mathbf{x}}(t) = f\Big(\mathbf{x}(t), \mathbf{u}(t)\Big) = \begin{bmatrix} u_v(t)\cos u_\alpha(t)\cos\theta_1(t) \\ u_v(t)\cos u_\alpha(t)\sin\theta_1(t) \\ \frac{1}{b_1}u_v(t)\sin u_\alpha(t) \\ \frac{d}{dt}\Big(u_v(t)\cos u_\alpha(t)\Big) \\ \dot{\vartheta}_v(t) \end{bmatrix} \tag{2.26a}$$

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ y_1(t) \\ \theta_1(t) \\ \vartheta_v(t) \\ \vartheta_{\text{acc}} \end{bmatrix} \tag{2.26b}$$

$$\mathbf{u}(t) = \begin{bmatrix} u_v(t) \\ u_\alpha(t) \end{bmatrix}. \tag{2.26c}$$

However, by substituting the control signals as in Fig. 2.5, the model can be further simplified to speed up calculations. The new control signals $\mathbf{u}(t)$ becomes

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} u_v\cos u_\alpha(t) \\ u_v\sin u_\alpha(t) \end{bmatrix}. \tag{2.27}$$

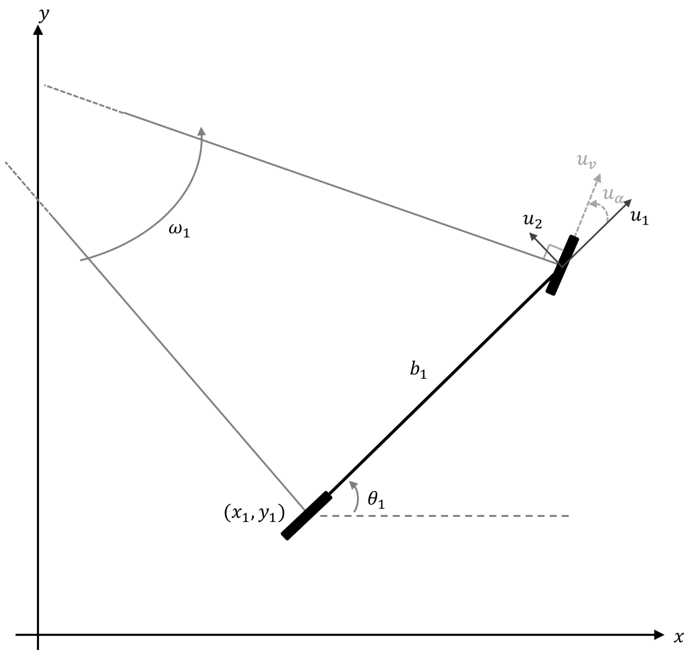


Fig. 2.5. Illustration of a simplified model for an articulated vehicle.

The model is then

$$f(\mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} u_1(t)\cos\theta_1(t) \\ u_1(t)\sin\theta_1(t) \\ \frac{1}{b_1}u_2(t) \\ \dot{u}_1(t) \\ \dot{\vartheta}_v(t) \end{bmatrix} \tag{2.28}$$

with the states as in (2.26b) and control signals in (2.27). The steering angle and speed can be retrieved as

$$\begin{aligned} \alpha(t) &= \arctan\left(\frac{u_2(t)}{u_1(t)}\right) \\ v(t) &= \frac{u_1(t)}{\cos\alpha(t)} \end{aligned} \tag{2.29}$$

Since the the steering angle and velocity have been substituted, the constraints for these become more elaborate. One way of implying a constraint on the steering angle, is to constrain the ratio between u_2 and u_1. This can be seen by first dividing u_2 with u_1 which will result in the expression

$$\frac{u_2(t)}{u_1(t)} = \frac{\sin\Big(u_\alpha(t)\Big)}{\cos\Big(u_\alpha(t)\Big)} = \tan\Big(u_\alpha(t)\Big). \tag{2.30}$$

If the ratio is constrained by the tangent value of $u_{\alpha,\min}$ and $u_{\alpha,\max}$, the expression becomes

$$\tan(u_{\alpha_{\min}}) \leq \frac{u_2(t)}{u_1(t)} \leq \tan(u_{\alpha_{\max}}). \tag{2.31}$$

The constraint is now non-convex and is advantageously simplified to decrease computational effort. It can be solved by multiplying both sides with u_1. On the other hand, since u_1 can take negative values it causes the constraint to become conditionally dependent on the sign of u_1. However, since $|u_{\alpha,min}| = |u_{\alpha,max}|$ this problem can be avoided by taking the absolute value of u_1. This forms the steering constraint as

$$|u_1(t)|\tan(u_{\alpha_{\min}}) \leq u_2(t) \leq |u_1(t)|\tan(u_{\alpha_{\max}}). \tag{2.32}$$

The maximum values of u_1 is constrained as

$$u_{1,\min} \leq u_1(t) \leq u_{1,\max}, \tag{2.33}$$

where $|u_{1,\min}| < |u_{1,\max}|$. In that way, together with minimising the time consumption, driving forward is promoted, yielding a more natural path with shorter reversing distance. The maximum and minimum value of the control signal u_2 is implicitly constrained with the combination of the equations (2.32) and (2.33).

3 Graph search path planner

During this project, a graph search path planning algorithm was developed. Since the turning radius of a semi-trailer truck is restricted, it is important that the reference curve keeps these parameters in regard in order to avoid collision. The aim is to develop an algorithm which computes a path that allows sneaking of curves and resembles the curvature of a turning truck. The developed algorithm is mainly based on Theta* described in 2.3.1.3, with additional logic to handle vehicle dynamics. The main difference here is how neighbours are found in the graph by the algorithm, with restrictions based on the dynamics of a truck in order to find paths which are closer to reality than what Theta* normally returns. This means that not all paths are guaranteed to be the shortest possible, since there might be constraints on how fast the heading can change, or how sharp turns are allowed to be.

The designed algorithm utilises a third dimension in order to keep track of possible headings in each node. The third dimension is divided into angular bins, which are then split into equally large parts, giving a range of possible angles for each bin. When a node is reached with a certain heading, it is assigned to the corresponding bin for that angle. A condition is set on the goal node in order to achieve a solution which fulfills the target heading. This means that the algorithm can find a solution with a mix of forward and backward movements.

3.1 Graph design

In this project, the graph was designed as a grid based graph. The indices of the graph describes the position in the world frame, where a resolution is set to determine the scale. The value of each index describes the occupancy status, as described in Section 2.3. The corresponding position $[x,\ y]^T$ is calculated from the given index $[i_x,\ i_y]^T$ as follows

$$\begin{bmatrix} x \\ y \end{bmatrix} = g_{\text{res}} \begin{bmatrix} i_x \\ i_y \end{bmatrix} \tag{3.1}$$

where g_{res} is the metric distance between two contiguous points. This means the size of the graph $(N_x,\ N_y)$ will be computed by

$$
\begin{aligned}
N_x &= \frac{x_{\text{max}} - x_{\text{min}}}{g_{\text{res}}} + 1 \\
N_y &= \frac{y_{\text{max}} - y_{\text{min}}}{g_{\text{res}}} + 1
\end{aligned}
\tag{3.2}
$$

where the max and min values describe the graph boundaries in x and y directions. Once the initial graph has been constructed, obstacles are added. The parameters of each obstacle are described by

$$
\begin{bmatrix} x_j & y_j & w_j & h_j & \alpha_j \end{bmatrix}^T \tag{3.3}
$$

where x_j and y_j is the position of the center of the object j, w_j and h_j are the width and height respectively and α_j is the rotation of the object. In addition to these parameters, a padding can be set to further increase the size of objects, this forces the graph search to take wider turns since the objects appear larger. The padded area is not necessarily restricted in the trajectory planning, and can be tuned separately if needed. Any point of the graph inside the boundary described by each obstacle and the additional padding is set to 1. This representation allows for fast computation and easy access to information. An example of how positions are evaluated is seen in Fig. 3.1. Each position is interpolated to the closest grid point and the occupancy status is checked to determine if this point is in an occupied space or not.

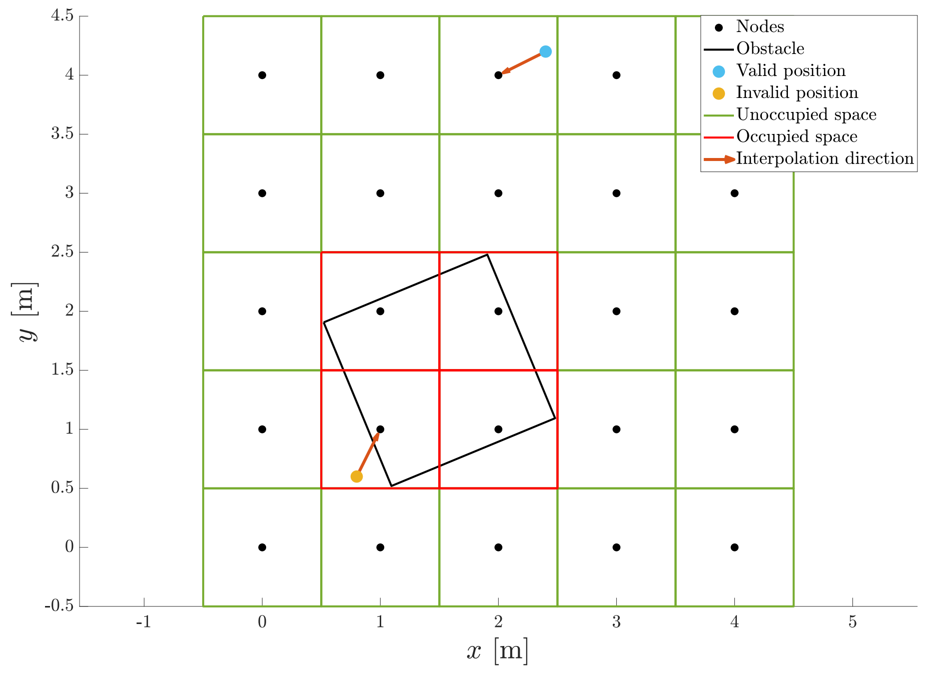


Fig. 3.1. Illustration of how points are interpolated to the closest grid point

Since the graph is grid based and the resolution is not infinitely high, there might be cases where a collision check fails to see obstacles. A common occurrence of this is when the path traverses a corner of an obstacle, which is in between points in the graph. The interpolation will fail to find this corner and thus consider this to be a valid path. This is another benefit of the padding, since the path will overlap with the padding, but not the obstacle itself, assuming the padding is large enough. Making obstacles appear larger is especially important when they can assume any rotation, since the edges are no longer lined up with the grid. Increasing the resolution enough to make this negligible would increase the computational effort immensely. Instead, a higher amount of padding could be used to handle the resolution problems that come with rotated obstacles.

3.2 Parent tracking

With the addition of the third dimension mention in Section 3, the parenthood of each node must also contain which bin of the parent it originates from. This results in each bin of each node being able to have a unique parent with a corresponding bin of that parent. This allows each node to be evaluated separately for each bin. This is valuable as the cost differs greatly between different bins. The result is an algorithm which tries multiple approaches in order to reach the target node. The computational complexity is increased, but the gain in flexibility allows for more complex scenarios.

3.2.1 Angular bins

The bins are designed to be around the target heading, where each bin has a uniform size based on the set number of bins. The goal node has a target bin, which is determined by the target heading. This enforces a specific range of allowed angles based on the size of each bin. An illustration of how the bins can be divided is found in Fig. 3.2.

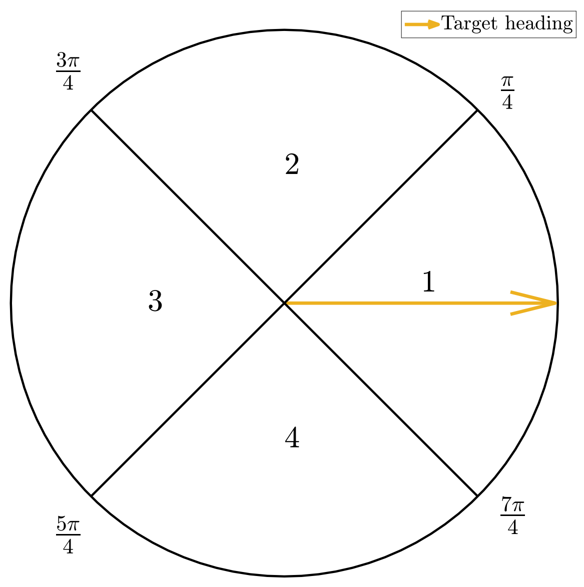


Fig. 3.2. Target heading is set to 0, and the number of bins is 4.

Calculation of which bin to assign is done as follows

$$\text{bin}_{\text{size}} = \frac{2\pi}{n_{\text{bins}}} \tag{3.4}$$

$$\text{bin}(\theta_{\text{current}}) = (\lfloor \frac{2\pi + \theta_{\text{current}} - \theta_{\text{goal}} + \frac{\text{bin}_{\text{size}}}{2}}{\text{bin}_{\text{size}}} \rfloor \ \% \ n_{\text{bins}}) \tag{3.5}$$

$$\theta_{\text{current}} \in [-\pi,\ \pi], \qquad \theta_{\text{goal}} \in [-\pi,\ \pi] \tag{3.6}$$

The addition of 2π is there to enable handling of negative angles, and the modulo operator is there to restrict the bin number from overflowing. The first bin is centered around the target heading, and then incremented counter clockwise. The size of each bin is uniform and based on the desired amount of bins described by the parameter n_{bins}. This parameter decides the resolution of the third dimension, indicating how many angles are possible in each position. Each bin will contain the lowest cost for the angular range of that slot. The number of possible bins decides the accuracy of the found solution, with a larger number giving a higher accuracy, due to the target bin being smaller with an increasing number of bins. More bins does, however, increase the computational complexity. Increasing the number of bins does not ensure a higher possibility of finding the target node, since there is a limit on the precision based on the available turn angles in the algorithm.

3.3 Node prioritisation

The neighbours found in each iteration are put in a priority queue based on the cost of reaching that node and an estimation of the remaining distance given by the heuristics. The aim is to prioritise the node which is most likely to reach the goal node in the desired bin.

3.3.1 Cost function

In order to prioritise the paths that are most likely to end up with the desired heading, the cost function includes a cost for headings differing from the target heading in addition to the euclidean distance $d(n_i, y_j)$ between the two nodes n_i, m_j, where i and j denote which bin are currently being evaluated for each node. These are not necessarily the same, but can be if the turning angle is small. The cost of traversing node n_i to m_j is then calculated as

$$C(n_i, m_j) = |\theta_{n_i} - \theta_{\text{goal}}| + d(n_i, m_j) \tag{3.7}$$

The cost array also includes a third dimension, since each node has a different cost for every bin, due to the cost depending on the difference in heading.

3.3.2 Heuristics

The chosen heuristics is the euclidean distance, this to assure that the cost of reaching the goal node is always underestimated. This is always a consistent heuristic, since the true cost can never be lower than the euclidean distance between two nodes. By using a consistent heuristic, the computed path will be optimal, but not necessarily the global optima. Given the restrictions of the algorithm, the optima might not be found in the search made by the algorithm, as some options are never found in the way the search is performed. Since the main focus of this project is to find a feasible path, this is considered to be a reasonable trade off.

3.4 Finding neighbours

To enable finding paths including both driving forward and reversing, neighbours are sought after in both directions for each node. The aim is to find a combination of these two directions in order to find the shortest path which fulfills the requirements set on the target heading. This results in up to twice as many neighbours in each search, as the search space is equally large in both directions. Since each neighbour is evaluated, this greatly increases the time for the algorithm to find a path to the goal. The back draft of increased computational complexity comes with the benefit of enabling conditions to be set on the final orientation.

In order to find paths which resemble the dynamics of the vehicle, neighbours are found using turning arcs with a predetermined turning radius. The visibility is determined by evaluating the resulting arc between two points and is hereafter denoted as arc of sight (AoS).

3.4.1 Arc of sight

The AoS is evaluated by projecting an arc based on a given turning angle α. A visualisation of the arc is illustrated in Fig. 3.3.

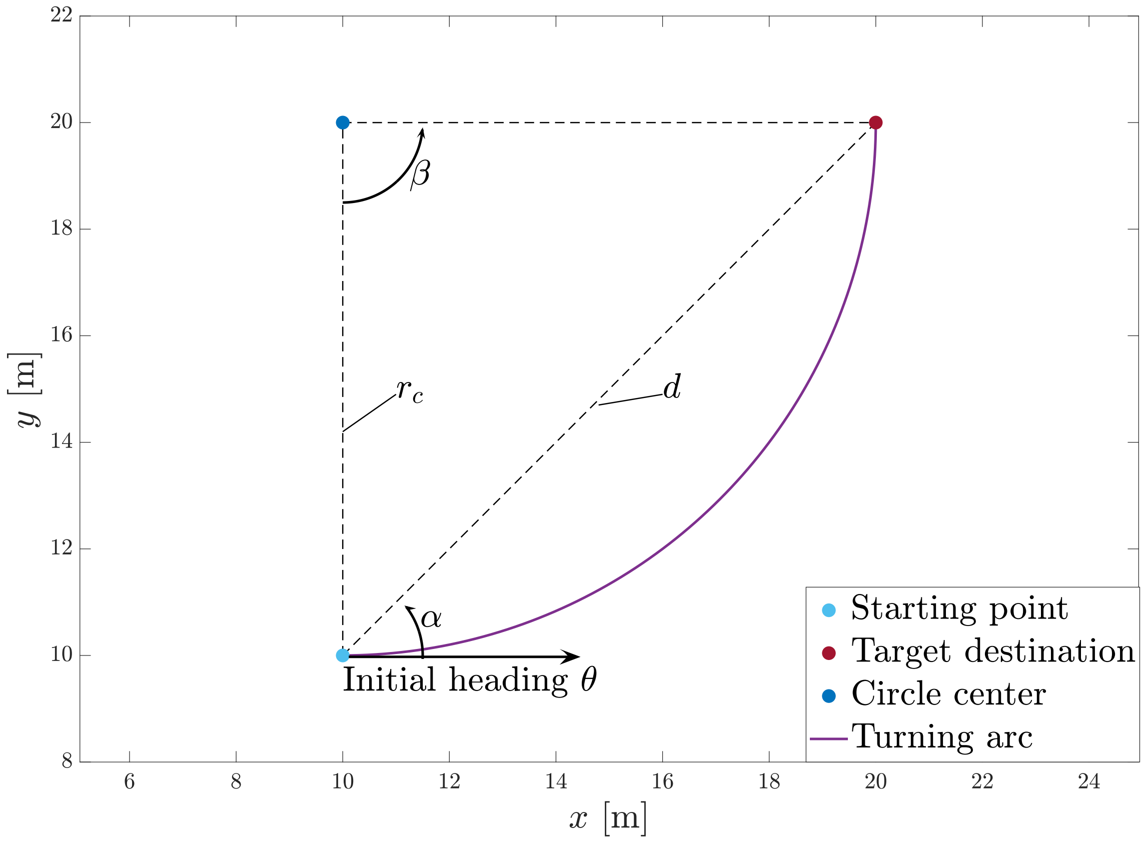


Fig. 3.3. Illustration of arc calculation

Here α is the turning angle, β is the angle between the two points seen from the center of the circle and θ is the initial heading. The turning radius is denoted by r_c and d is the euclidean distance between the two points. The turning radius is calculated by assuming a constant turning angle, yielding a circular motion. The center of rotation is by definition perpendicular to the current position $[x_0 \;\; y_0]^T$. By using the distance d between two positions, and knowing that the angle between them is the steering angle α, the distance to the center of rotation can be calculated using the following equations.

$$\begin{aligned} \beta &= 2\alpha \\ r_c &= \frac{d}{2\sin\alpha} \end{aligned} \tag{3.8}$$

Since the movement is perpendicular to the center of the circle, the change in heading is equal to β. The new position $[x_1 \;\; y_1]^T$ and new heading θ_1 is calculated by.

$$\begin{aligned} x_1 &= x_0 + d\cos(\theta + \alpha) \\ y_1 &= y_0 + d\sin(\theta + \alpha) \\ \theta_1 &= \theta + \beta \end{aligned} \tag{3.9}$$

By lowering the resolution and limiting the amount of turning angles available, the process is sped up significantly. By default, only full turns of $\frac{\pi}{4}$ are allowed. If the algorithm fails to find the goal node, the available turns are scaled up to handle more complex situations. The full turn results in a heading change of $\frac{\pi}{2}$, but since the algorithm allows inheriting parenthood from the parent of the parent node, more angles can be achieved. This proved to be useful in order to handle scenarios of rotated obstacles, since the required heading might not be a multiple of $\frac{\pi}{2}$. Due to the available turning angles being restricted, more nodes are sought after in the current heading, as well as in the opposite direction up to a distance equal to the turning radius multiplied by $\sqrt{2}$. The position of these nodes are found using Bresenham's line algorithm. This is illustrated below in Fig. 3.4.

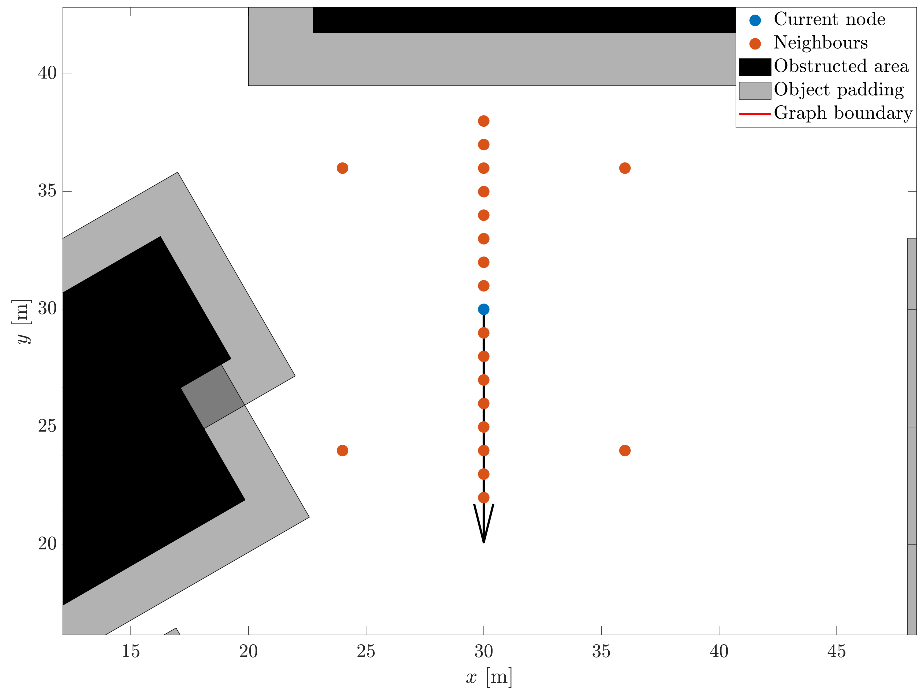


Fig. 3.4. Neighbours found by the graph search algorithm, where turning angles are limited to $\frac{\pi}{4}$.

To see if the selected target node is in sight of the parent of the current node the angle α and euclidean distance d is used to calculate the arc parameters as seen in (3.8). The points p on the arc are obtained by creating a spectrum of angles and computing the points as follows

$$
\begin{aligned}
\begin{bmatrix} c_{rx} \\ c_{ry} \end{bmatrix} &= \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} r_c \sin(\theta - \frac{\pi}{2}) \\ r_c \cos(\theta - \frac{\pi}{2}) \end{bmatrix} \\
\begin{bmatrix} p_{rx} \\ p_{ry} \end{bmatrix} &= \begin{bmatrix} c_{rx} + r_c \cos(\theta + \frac{\pi}{2} - \gamma) \\ c_{ry} + r_c \sin(\theta + \frac{\pi}{2} - \gamma) \end{bmatrix} \qquad \forall \gamma \in [0, \beta] \\
\begin{bmatrix} c_{lx} \\ c_{ly} \end{bmatrix} &= \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} r_c \sin(\theta + \frac{\pi}{2}) \\ r_c \cos(\theta + \frac{\pi}{2}) \end{bmatrix} \\
\begin{bmatrix} p_{lx} \\ p_{ly} \end{bmatrix} &= \begin{bmatrix} c_{lx} + r_c \cos(\theta - \frac{\pi}{2} + \gamma) \\ c_{ly} + r_c \sin(\theta - \frac{\pi}{2} + \gamma) \end{bmatrix} \qquad \forall \gamma \in [0, \beta]
\end{aligned}
\tag{3.10}
$$

for a right turn and left turn respectively. Here l indicates left turn and r a right turn. The location of the respective circle centers are described by c and γ is a vector of angles with a resolution high enough to ensure that no collisions are missed. Each point along the arc is compared to the obstacles in the environment. If any point on the arc has a value of 1 in the graph, the neighbour is out of sight.

3.4.2 Neighbours in proximity of the goal node

When the distance to the goal node is lower than the predetermined turning radius, full steps can no longer be taken while assuring the goal node is found. Instead of just searching the full turns and straight ahead, all neighbours within the boundary described by these are identified. This is done by creating a cone defined by the possible maximum turning angles of the vehicle. This angle is set to $\pm\frac{\pi}{4}$, giving a triangular grid of reachable points, which are then interpolated to nodes in the graph, resulting in a set of neighbours. The triangular grid is calculated by the points between the three corners consisting of current position $[x_0 \;\; y_0]^T$ and the two points $[x_n \;\; y_n]^T$ that can be reached by making a turn to the left or right. The two points are calculated as follows

$$
\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \sqrt{2} T_r \cos(\theta \pm \frac{\pi}{4}) \\ \sqrt{2} T_r \sin(\theta \pm \frac{\pi}{4}) \end{bmatrix}, \qquad n \in 1, 2 \tag{3.11}
$$

where T_r is the turning radius and θ is the current heading, which determine the size of the cone. The grid is then filtered by removing the neighbours requiring turning angles higher than the maximum. This is done by checking the distance to the circle center described in Section 3.4.1, if this distance is lower than the circle radius, the point is within the circle. If a point is within the circle, it means that it is not possible to reach it from the current node. The resulting neighbours resemble a bell shape and can be seen in Fig 3.5

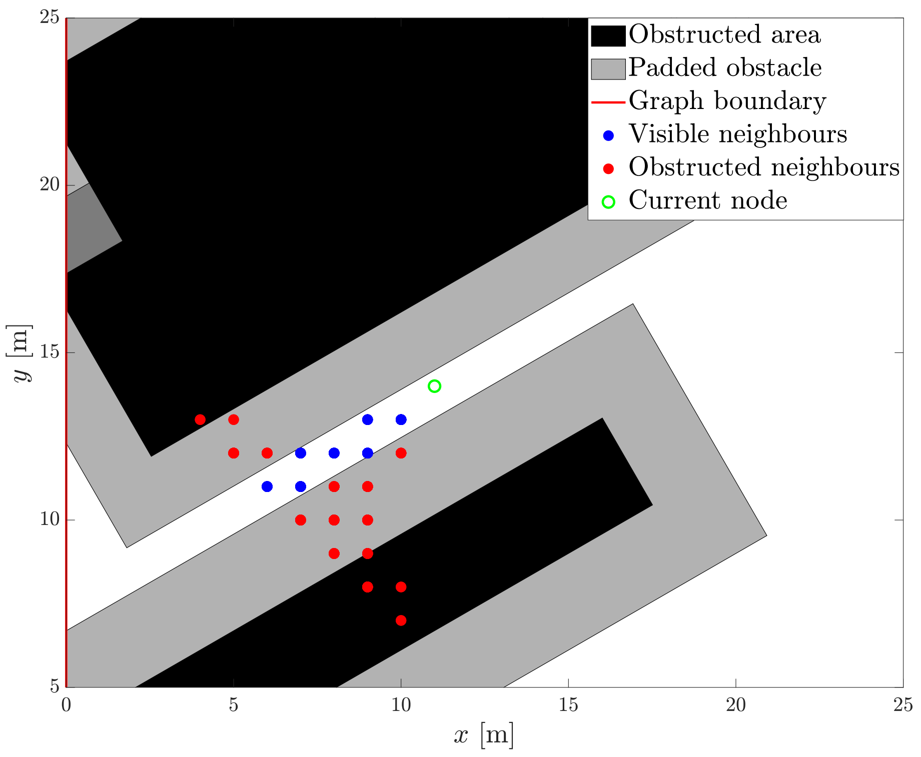


Fig. 3.5. Illustration of how neighbours are found when the distance to the goal node is less than one turning radius

Here, some neighbours are not considered to be visible as they are inside the obstacles or padding, this is determined by checking the occupancy value of each node. Any neighbour which is unreachable based on the vehicle dynamics has been removed before line of sight is determined.

3.5 Inheriting parenthood

Once the neighbours are identified, they are put in the open list and the AoS is evaluated from the current node and the parent of the current node. In this algorithm, the parenthood can not be inherited if the turn is too sharp. If the distance between the parent and the neighbour is below or equal to the turning radius, the lateral movement will be limited by the set dynamics. The lateral distance will at maximum be equal to the covered longitudinal distance in the vehicle coordinate system, assuming a $\frac{\pi}{4}$ turn. An illustration of the AoS can be seen in Fig. 3.6 where the visibility check is shown.

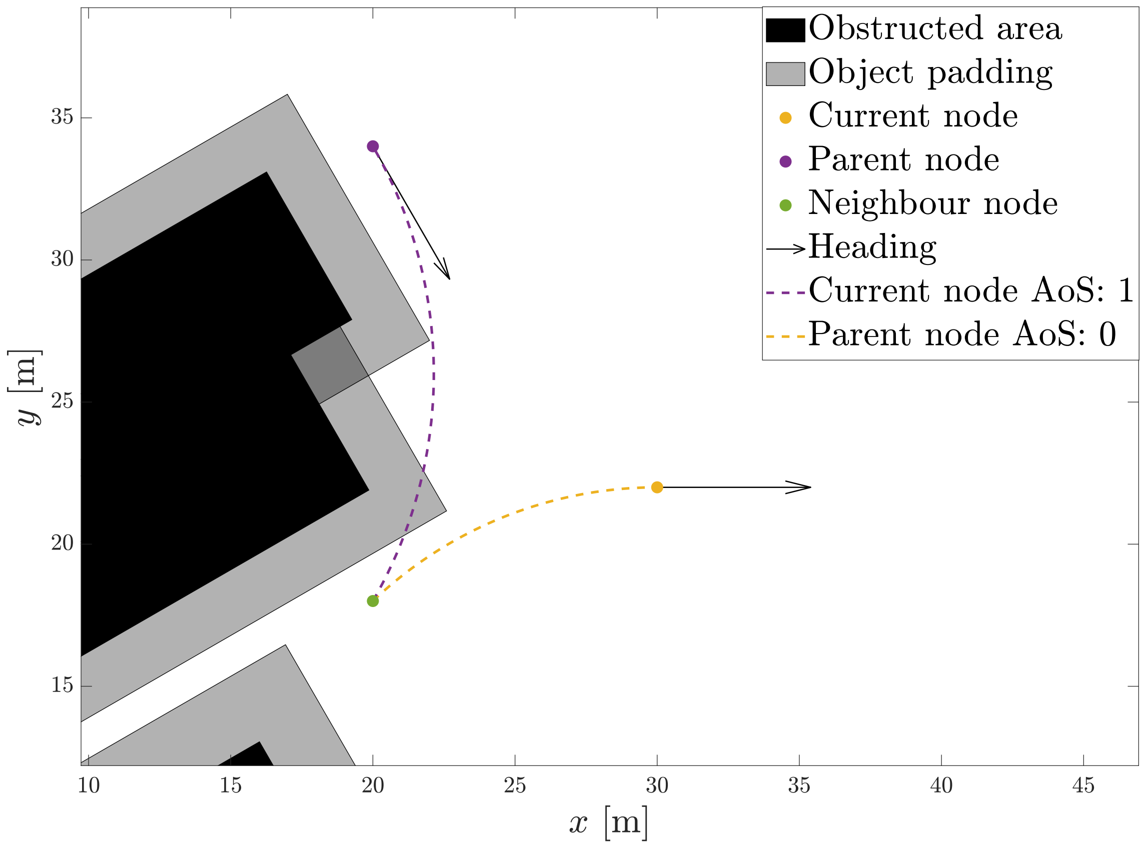


Fig. 3.6. Visualisation of how AoS is determined. Here, the parent node does not have a valid path, while the current node does.

The search direction from the current node is backwards, and the parent is searching forwards. Since the vision of the parent is obstructed by the obstacle the parenthood can not be inherited by the neighbour. Instead the parenthood is assigned to the current node as there is a valid path to it. The search direction is determined by the heading of the parent node and current node. The only valid search direction for the current node is backwards, as making a π turn is unwanted and unnecessary. The heading at the neighbour after assignment of parent will depend on the type of movement made to reach it. If it was reached by reversing, the heading will be opposite of the movement direction to represent the direction of the front of the truck. This means that in order to continue in this direction from the node, it has to keep reversing. Since the search method in both directions is equal, this has no effect on the neighbours found, but it will affect how the MPC trajectory planner follows the resulting reference path as the driving direction is saved in each position. The driving direction is represented by a boolean along the points on the curve, where true indicates reversing.

4 Model predictive control

MPC is a framework used to optimise controlling of systems, where a traditional LQ controller is not general enough. With MPC, there is a great freedom in designing the objective function and constraints in order to accurately describe the system. The challenge lies in the design of these.

When using MPC to plan a trajectory, metrics such as acceleration and jerk is often penalised by including it in the objective function. Objects, however, are handled differently in various implementations. In this project, obstacles are considered as constraints as opposed to maximising the distance to them.

MPC will be used in two different contexts. Both to create a reference path and also to follow a reference path. The first will be a single horizon MPC, while the other has a receding horizon.

In order to decrease the computational effort in the trajectory planning phase, the horizon needs to be as short as possible. Too short, and the problem might become infeasible, too long and the solver will be too slow. However, time complexity will not be investigated in this project, but a reasonable horizon will nevertheless be used.

In this section the approach of using MPC as a path planner will be discussed in Section 4.2 and how it is later passed on and used in Section 4.3.

4.1 Interpretation of obstacles

Objects can be interpreted in many different ways but will in this project be approximated as one or more rectangles. Each with a center, width, height and rotation. For them to be included in the MPC formulation, they need a mathematical definition. The shortest distance $d_{ij}(k|\kappa)$, between a point p_i on the truck, and a rotated rectangle with index j is expressed as

$$
\begin{aligned}
d_{ij}(k|\kappa) &= \left\| \max\left(\mathrm{abs}\left(\mathbf{R}^T \begin{bmatrix} x_i(k|\kappa) - x_j \\ y_i(k|\kappa) - y_j \end{bmatrix} \right) - \begin{bmatrix} w_j/2 \\ h_j/2 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \right\|, \\
\text{where,} \quad \mathbf{R} &= \begin{bmatrix} \cos(\alpha_j) & -\sin(\alpha_j) \\ \sin(\alpha_j) & \cos(\alpha_j) \end{bmatrix}.
\end{aligned}
\tag{4.1}
$$

The angle α is a counter clock-wise rotation measured from the x-axis and w, h is the width and height of the rotated rectangle. The reason for the abs$(\cdot)$ and max$(\cdot)$ operators is due to the fact that only the absolute distances to the objects are of interest.

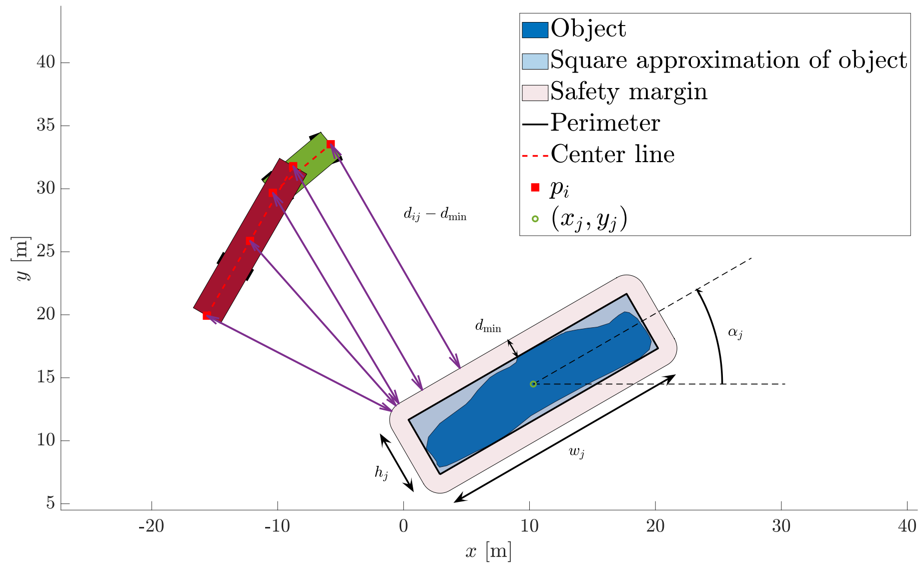


Fig. 4.1. Definition of an object when interpreted as a rotated rectangle.

The distance $d_{ij} - d_{\min}$ between points along the center line of the semi-trailer truck and an obstacle is depicted in Fig. 4.1. The safety margin of size $d_{\min}$ is used since the points along the truck only approximates its shape. If the safety margin is too narrow, some parts of the truck may be inside the obstacle even though the points are not.

The problem arises when two points are on opposite sides of a corner. In Fig. 4.2 this situation is illustrated, where the points are red and denoted p_i. The black line symbolises one of the sides of the semi-trailer truck and the red line is going through the center of the vehicle. The line is at a $\frac{\pi}{4}$ angle which is the worst case scenario and needs the largest margin. From this, the relation between the minimum margin m and distance between points Δp can be established as

$$m = \frac{1}{2}w_{\text{veh}}\cos\left(\frac{\pi}{4}\right) + \frac{1}{2}\Delta p\cos\left(\frac{\pi}{4}\right) \tag{4.2}$$

$$\Rightarrow m = \frac{1}{2\sqrt{2}}(w_{\text{veh}} + \Delta p) \tag{4.3}$$

This calculation assumes that the edge of the safety margin surrounding the obstacle is square. That means for small enough Δp, the needed margin m is actually larger than what (4.3) suggests. This has to be taken into consideration when designing the appropriate safety margin.

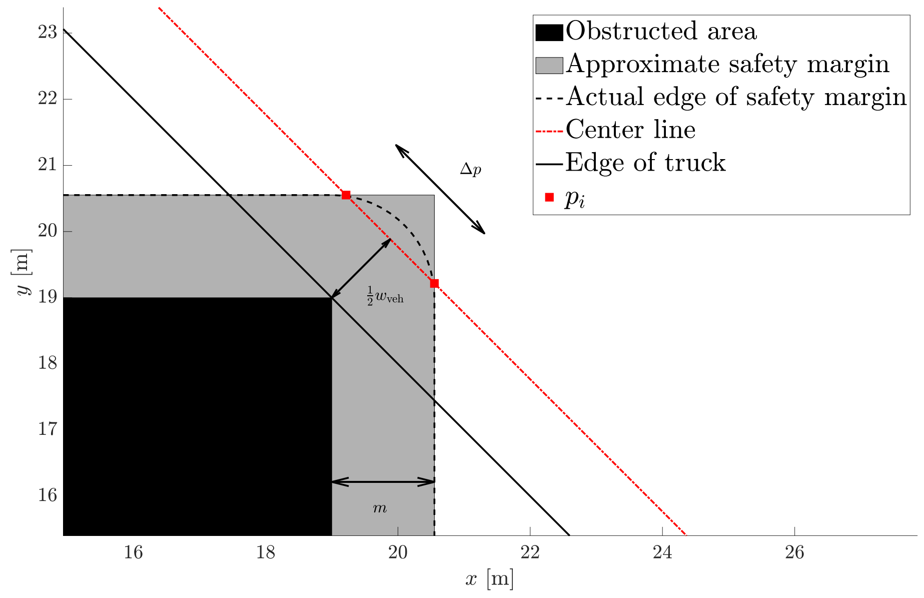


Fig. 4.2. The grey area is the safety margin of width m, the black area is the obstacle. The black line symbolises the side of the semi trailer truck that cuts through the safety margin. The red dots are the point from which the distance to the object is measured from.

4.2 Path planner

The advantage of solving an optimisation problem in an MPC fashion when creating a reference path, is that it is possible to force it to obey specific dynamics. In this case, it will be the simplified model described in Section 2.6.2 with one variable $z(\kappa)$ controlling the lethargy over the whole horizon.

4.2.1 Obstacle avoidance

Since the reference path needs to avoid obstacles, points along the center line of the simplified model are constrained to always have a minimum distance to the obstacles. The distance between object j and a point i on the vehicle, denoted $d_{ij}(k|\kappa)$, are calculated as in Section 4.1 and constrained as

$$d_{ij}(k|\kappa) \geq d_{\min}, \tag{4.4}$$

where $d_{\min}$ is calculated using (4.3).

4.2.2 Final state

To make sure that the vehicle ends up in the correct position and orientation, the last state is constrained. However, the orientation θ_1 is not invariant to multiples of 2π. That is, if the final orientation is set to an incorrect angle, e.g. $\theta_f + 2\pi$ rad instead of θ_f, the path will make a 2π turn before ending up in the end position. The correct angle is dependant on the path it chooses to take. If the vehicle has to turn counter clock wise on its way to goal, the state θ_1 is decreasing and the target orientation must be matched, e.g. $-3\pi/4$ rad instead of $5\pi/4$ rad. To solve this problem, the orientation is instead constrained by considering the position of the vehicle (x_1, y_1) together with the position of the rear axle of the vehicle (x_2, y_2). This constraint is formulated as

$$\begin{aligned} x_1(\kappa + k_H|\kappa) &= x_{1f} \\ y_1(\kappa + k_H|\kappa) &= y_{1f} \\ x_2(\kappa + k_H|\kappa) &= x_{1f} - b_1 \cdot \cos\theta_f \\ y_2(\kappa + k_H|\kappa) &= y_{1f} - b_1 \cdot \sin\theta_f, \end{aligned} \tag{4.5}$$

where θ_f is the target orientation and x_{1f}, y_{1f} is the target position. The position (x_2, y_2) is in turn calculated as

$$\begin{bmatrix} x_2(\kappa + k_H|\kappa) \\ y_2(\kappa + k_H|\kappa) \end{bmatrix} = \begin{bmatrix} x_1(\kappa + k_H|\kappa) \\ y_1(\kappa + k_H|\kappa) \end{bmatrix} - b_1 \cdot \begin{bmatrix} \cos\Big(\theta(\kappa + k_H|\kappa)\Big) \\ \sin\Big(\theta(\kappa + k_H|\kappa)\Big) \end{bmatrix}. \tag{4.6}$$

These two equations together form the complete constraint as

$$\begin{aligned} x_1(\kappa + k_H|\kappa) &= x_{1f} \\ y_1(\kappa + k_H|\kappa) &= y_{1f} \\ x_1(\kappa + k_H|\kappa) - b_1 \cos\Big(\theta(\kappa + k_H|\kappa)\Big) &= x_{1f} - b_1 \cos\theta_f \\ y_1(\kappa + k_H|\kappa) - b_1 \sin\Big(\theta(\kappa + k_H|\kappa)\Big) &= y_{1f} - b_1 \sin\theta_f, \end{aligned} \tag{4.7}$$

4.2.3 Optimisation problem

The objective function will be designed such that the solution to the optimisation problem follows the intended behaviour. In this case, acceleration is considered unwanted together with time consumption. By reducing these, the path will include as few changes of driving direction as possible as well as the fastest path. It will still be possible to find paths of three- or more point turns and paths of mainly reversing, they will just not be favored. $\tilde{J}(\cdot)$ is therefore defined as

$$\tilde{J}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa)) = w_1\Big(\vartheta_{\text{acc}}(k|\kappa)\Big)^2 + w_2\Big(z(\kappa)\Big)^2 \\ \forall\ k \in [\kappa, \kappa + k_H] \tag{4.8}$$

The complete optimisation problem with the constraints (2.32)-(2.33), (4.4), (4.7) and the objective function (4.8) is then formulated as

$$\begin{aligned}
\min_{x,u} \quad & \sum_{\tau}^{\tau+t_H} \tilde{J}\Big(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), z(\kappa)\Big)dt && \text{(4.9a)}\\
\text{s.t.} \quad & \mathbf{x}(k+1|\kappa) = \tilde{f}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), z(\kappa), k) && \text{(4.9b)}\\
& d_{ij}(k|\kappa) \geq d_{\min} && \text{(4.9c)}\\
& x_1(\kappa + k_H|\kappa) = x_{1f} && \text{(4.9d)}\\
& y_1(\kappa + k_H|\kappa) = y_{1f} && \text{(4.9e)}\\
& x_1(\kappa + k_H|\kappa) - \cos\Big(\theta(\kappa + k_H|\kappa)\Big) = x_{1f} - \cos\theta_f && \text{(4.9f)}\\
& y_1(\kappa + k_H|\kappa) - \sin\Big(\theta(\kappa + k_H|\kappa)\Big) = y_{1f} - \sin\theta_f && \text{(4.9g)}\\
& |u_1(k|\kappa)|\tan(u_{\alpha_{\min}}) \leq u_2(k|\kappa) \leq |u_1(k|\kappa)|\tan(u_{\alpha_{\max}}) && \text{(4.9h)}\\
& u_{1,\min} \leq u_1(k|\kappa) \leq u_{1,\max}. && \text{(4.9i)}
\end{aligned}$$

This optimisation problem is passed on to the numerical solver Ipopt. As explained previously in Section 2.5, the solver needs a feasible initial guess of variables. The most important variables are the positions. A random guess, or even a straight line between the start and end, is deemed insufficient. By measuring the euclidean distance, no regard is taken to objects. This means, that there is still no estimate of what the true distance might be and no conclusion of the amount of needed samples can be drawn.

Therefore, the result of an A* graph search from start to end is used. With enough padding it is hypothesised that it will provide a feasible starting point for the positions. By performing an initial search, an estimate of the distance is made. This estimate can be used in order to determine the amount of sample points needed to solve the task. The A* algorithm will also find the shortest path, when there are several options.

There are however other variables to initialise, e.g. θ_1. It is set to a gradient transition from θ_0 to θ_f across the horizon, either clock-wise or counter clock-wise, whichever is closest. The control signals u_1, u_2 are initialised to zero.

4.2.4 Post processing

The sampling interval of the MPC path planner does not necessarily have to be the same as the one used in the trajectory planner. As the trajectory planner needs higher precision when calculating the distance to objects, it requires a denser sampling interval. To ensure complete freedom in the choice of sampling interval for the trajectory planner, the path created by this path planner has to be upsampled.

It is possible to use a simple interpolation algorithm to post process, such as linear or spline. However, these do not take the motion model into account. In order to interpolate with the dynamics in regard, the states are simulated with the control signals calculated by the path planner. By providing the trajectory planner with a densely sampled reference curve, it is given complete freedom of choosing its own sampling interval.

4.3 Trajectory planner

The trajectory planner uses the path from one of the two previous generated paths as reference. The reference can be considered in two different ways. Both minimising the distance to it but also using it as a reference on how to position the truck at the end of the horizon. The latter is needed to force reversing even though there is a possibility to reach the target state by driving forward.

However, the reference curve that comes from the two path planners is densely sampled and needs to be split into parts of equal length $\Delta\zeta$. This parameter is tuneable and, together with the horizon length ζ_H, adjust how many samples each horizon contains. Denser sampling leads to better obstacle avoidance but also results in longer computation time. The task of the trajectory planner is to continuously set control signals based on the current state and predictions. In order for it to work well, the model needs to accurately describe the motion of the vehicle and for an implementation on a real semi-trailer truck, the model presented in Section 2.6.1, is not accurate enough. However, for the purpose of this project, i.e. to plan a trajectory that includes reversing, it is sufficient if the model produces reasonable simulations. If it was to be deployed in a real truck, the model can be extended to include slipping, motor specification etc.

4.3.1 Obstacle avoidance

The obstacles are considered in the same way as in the MPC path planner, namely

$$d_{ij}(k|\kappa) \geq d_{\min}. \tag{4.10}$$

where d_{ij} is the distance between object i and point j on the vehicle.

4.3.2 Reference path

For the MPC to make use of the provided reference curve, the distance between a point on the vehicle and the reference curve can be penalised. Since neither of the algorithms that create the reference path take a trailer into account, using the point on the rear axle of the truck (x_1, y_1) would make the trailer cut turns. Instead, the axle of the trailer, (x_2, y_2), is considered best suited to be as close to the reference path as possible. This distance is expressed as

$$d(k|\kappa) = \left\| \begin{bmatrix} x_2(k|\kappa) \\ y_2(k|\kappa) \end{bmatrix} - \mathbf{r}(k|\kappa) \right\| = \sqrt{\Big(x_2(k|\kappa) - r_x(k|\kappa)\Big)^2 + \Big(y_2(k|\kappa) - r_y(k|\kappa)\Big)^2}, \tag{4.11}$$

where

$$\mathbf{r}(k|\kappa) = \begin{bmatrix} r_x(k|\kappa) \\ r_y(k|\kappa) \end{bmatrix}. \tag{4.12}$$

Since the solver needs to differentiate the expression $d(k|\kappa)$, there can not be a square root expression of a variable. That will make the derivative undefined where the distance between the position $\Big(x_2(k|\kappa), y_2(k|\kappa)\Big)$ and the reference path $\Big(r_x(k|\kappa), r_y(k|\kappa)\Big)$ equals zero. Therefore, the square distance is added to the cost function as

$$d(k|\kappa)^2 = \left\| \begin{bmatrix} x_2(k|\kappa) \\ y_2(k|\kappa) \end{bmatrix} - \mathbf{r}(k|\kappa) \right\|^2 = \Big(x_2(k|\kappa) - r_x(k|\kappa)\Big)^2 + \Big(y_2(k|\kappa) - r_y(k|\kappa)\Big)^2. \tag{4.13}$$

4.3.3 Final state

To make sure that the vehicle ends up where it should, the position at the end of the horizon needs to be constrained to the reference curve. If not, the solution that minimises the objective function could be to not move at all. However, the reference curve might be too close to an obstructed area which would make for an infeasible solution. This suggests that flexibility is needed, which is solved by introducing soft constraints. For the same reason as before, the square distance is constrained as

$$\left\| \begin{bmatrix} x_2(\kappa + k_H|\kappa) \\ y_2(\kappa + k_H|\kappa) \end{bmatrix} - \mathbf{r}(\kappa + k_H|\kappa) \right\|^2 \leq \delta_r(\kappa)^2 \tag{4.14}$$

where $\delta_r(\kappa)$ is the soft constraint variable and is only a function of κ since there is only one per horizon. The variable itself will also be constrained as

$$0 \leq \delta_r(\kappa) \leq \delta_{r,\max} \tag{4.15}$$

to guarantee a minimum distance to the target position. It will later be added to the objective function to minimises the deviation. To further guide the vehicle in each horizon, the angle of the truck and trailer is soft constrained as

$$\begin{aligned} \left\| \begin{bmatrix} x_1(\kappa + k_H|\kappa) \\ y_1(\kappa + k_H|\kappa) \end{bmatrix} - \begin{bmatrix} p_x(\kappa + k_H|\kappa) \\ p_y(\kappa + k_H|\kappa) \end{bmatrix} \right\|^2 \leq&\ \delta_f(\kappa)^2 \\ 0 \leq&\ \delta_f(\kappa) \leq \delta_{f,\max}. \end{aligned} \tag{4.16}$$

This makes it cheaper to align with the reference curve at the end of each horizon but leaves the possibility of having different trailer angles. The reason why the position of the front is constrained instead of the angle of the truck and trailer is for the same reason as for the MPC path planner, to get rid of the ambiguity of angles of multiples of 2π. The position $\big(p_x(\kappa + k_H|\kappa), p_y(\kappa + k_H|\kappa)\big)$ is the calculated position of the front given that the trailer angle $\varphi = 0$ and truck tangent to the reference curve. They are calculated as

$$\begin{aligned} p_x(\kappa + k_H|\kappa) &= r_x(\kappa + k_H|\kappa) \pm \frac{a_1 + b_2}{\Delta\zeta}\Big(r_x(\kappa + k_H|\kappa) - r_x(\kappa + k_H - 1|\kappa)\Big) \\ p_y(\kappa + k_H|\kappa) &= r_y(\kappa + k_H|\kappa) \pm \frac{a_1 + b_2}{\Delta\zeta}\Big(r_y(\kappa + k_H|\kappa) - r_y(\kappa + k_H - 1|\kappa)\Big) \end{aligned} \tag{4.17}$$

where $\pm$ depends on if the vehicle is to reverse or not. The information of reversing is encoded in the reference curve. This is visualised in Fig. 4.3. In the right plot, the truck is instructed to reverse at the end of horizon and therefore flips the target heading π by changing the sign to negative in Equation (4.17). At the last horizon, the target state $\mathbf{x}_f$ that includes position and orientation is given by the problem as opposed to the reference path which is true for all other horizons.

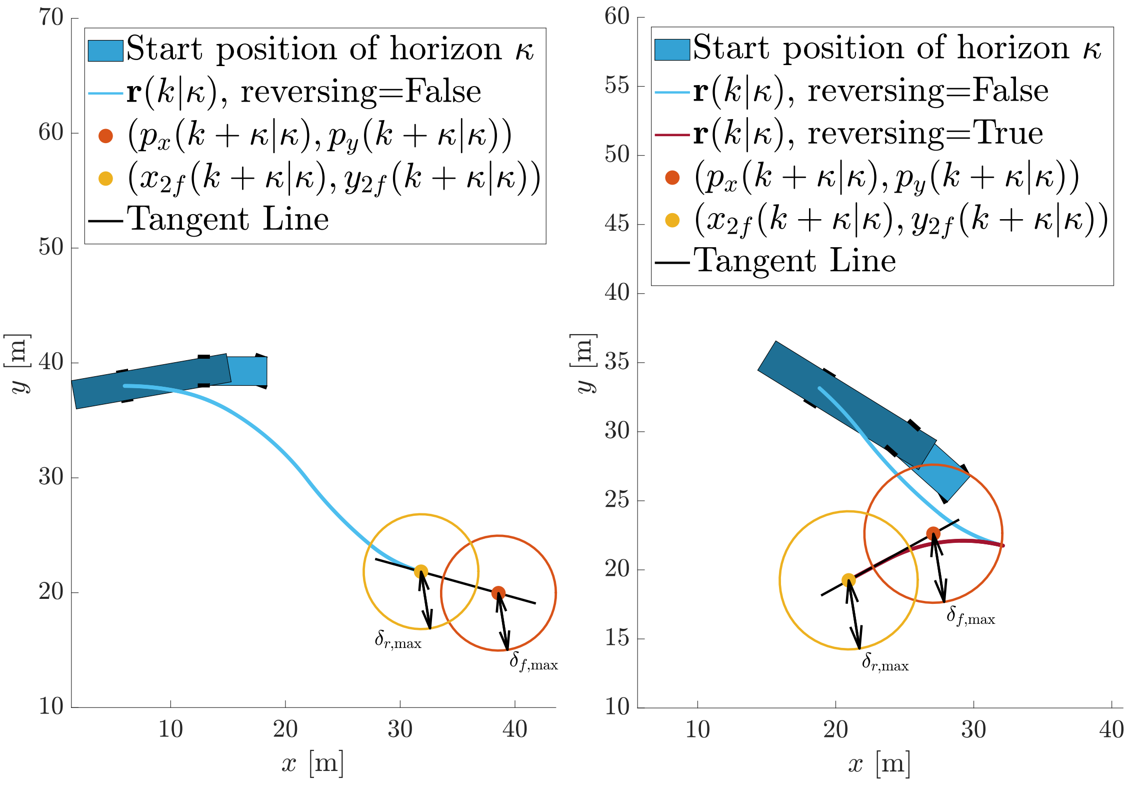


Fig. 4.3. Illustration of how the target position is soft constrained and takes the intended movement direction into account. The circles describe the target set of the horizon and the line connecting them is tangent to the reference path at $\mathbf{r}(\kappa + k_H|\kappa)$.

4.3.4 Optimisation problem

Together with penalising deviation from the reference path and the soft constraint variables δ_r, δ_f, the acceleration, steering rate, jerk and lethargy are also penalised in the objective function $\sum_{\kappa}^{\kappa+k_H} \left(\tilde{J}(\mathbf{x}, \mathbf{u}, \mathbf{r}, z)\right) + J_f(\delta_r, \delta_f)$, where $\tilde{J}_f(\delta_r, \delta_f)$ is the target cost. These are formulated as

$$\begin{aligned}
\tilde{J}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), \mathbf{r}(k|\kappa), z(k|\kappa)) &= w_1\Bigg(\left\| \begin{bmatrix} x_2(k|\kappa) \\ y_2(k|\kappa) \end{bmatrix} - \mathbf{r}(k|\kappa) \right\|^2\Bigg) + \ldots \\
&\quad + w_2\left(u_{\text{acc}}(k|\kappa)\right)^2 + w_3\left(\vartheta_{\dot{\alpha}}(k|\kappa)\right)^2 \ldots \\
&\quad + w_4\left(\vartheta_{\text{jerk}}(k|\kappa)\right)^2 + w_5 z(k|\kappa) \qquad (4.18) \\
\tilde{J}_f(\delta_r, \delta_f) &= w_6\delta_r(\kappa) + w_7\delta_f(\kappa) \\
\\
\forall\, k &\in [\kappa, \kappa + k_H]
\end{aligned}$$

where w_i are weightings of the different terms and tuned to achieve the desired behaviour.

The complete optimisation problem with the constraints (2.25) and objective function (4.18) looks as follows

$$
\begin{aligned}
\min_{x,u,z} \quad & \left(\sum_{k=\kappa}^{\kappa+k_H} \tilde{J}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), \mathbf{r}(k|\kappa), z(k|\kappa)) \right) + \tilde{J}_f(\delta_r, \delta_f) && \text{(4.19a)} \\
\text{s.t.} \quad & \mathbf{x}(k+1|\kappa) = \tilde{f}(\mathbf{x}(k|\kappa), \mathbf{u}(k|\kappa), z(k|\kappa), k) && \text{(4.19b)} \\
& d_{ij}(k|\kappa) \geq d_{\min} && \text{(4.19c)} \\
& u_{\alpha,\min} \leq u_\alpha(k|\kappa) \leq u_{\alpha,\max} && \text{(4.19d)} \\
& u_{\text{acc},\min} \leq u_v(k|\kappa) \leq u_{\text{acc},\max} && \text{(4.19e)} \\
& \varphi_{\min} \leq \varphi(k|\kappa) \leq \varphi_{\max} && \text{(4.19f)} \\
& \left\| \begin{bmatrix} x_2(\kappa+k_H|\kappa) \\ y_2(\kappa+k_H|\kappa) \end{bmatrix} - \mathbf{r}(\kappa+k_H|\kappa) \right\|^2 \leq \delta_r(\kappa)^2 && \text{(4.19g)} \\
& \left\| \begin{bmatrix} x_1(\kappa+k_H|\kappa) \\ y_1(\kappa+k_H|\kappa) \end{bmatrix} - \begin{bmatrix} p_x(\kappa+k_H|\kappa) \\ p_y(\kappa+k_H|\kappa) \end{bmatrix} \right\|^2 \leq \delta_f(\kappa)^2 && \text{(4.19h)} \\
& 0 \leq \delta_r(\kappa) \leq \delta_{r,\max} && \text{(4.19i)} \\
& 0 \leq \delta_f(\kappa) \leq \delta_{f,\max} && \text{(4.19j)} \\
& z_{\min} \leq z(k|\kappa) \leq z_{\max}. && \text{(4.19k)}
\end{aligned}
$$

The same solver as in the MPC path planner is used for this problem, namely the Ipopt. This means that it is also sensitive to the initial guesses. However, since the path provided by either path planner is designed with the vehicle dynamics in regard, it is assumed to be a good initial guess. After each horizon is solved, its solution is provided as the initial guess to the next one.

5

Results

In this section the performance of the algorithms will be evaluated separately and a short explanation of the behaviour will be given.

5.1 Path planners

Each path planner was designed to provide a reference curve between two given points. The curve contains information about position and driving direction, either moving forward or reversing. The position at each point along the curve indicates where the rear axle of the trailer is supposed to be, the position of the truck is estimated by the trajectory planner using the vehicle dynamics.

The driving direction is represented by a boolean and the position is given as euclidean coordinates. An expected orientation can be extrapolated by using the tangent between two adjacent points on the curve.

Each reference curve of the respective algorithms will be evaluated separately and used as input to the trajectory planner. In the illustrations shown below for each algorithm the colour of the curve visualises intended moving direction. Burgundy represents driving forward and teal indicates reversing.

In addition to the main scenario presented in Section 1.4, the separate algorithms is tested on two other scenarios in order to further investigate their strengths and shortcomings. These two scenarios are presented in Fig. 5.1.

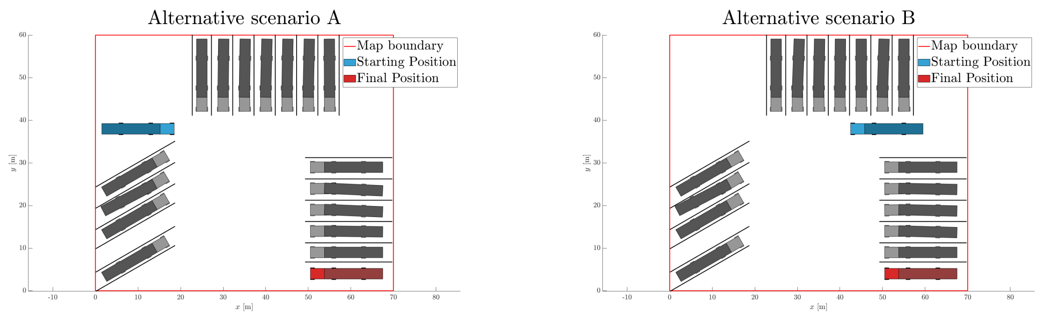


Fig. 5.1. Two alternative scenarios to evaluate the path planning algorithms (a) Scenario A requires a rotation of π before parking (b) In scenario B the final orientation should be the same as the starting one.

5.1.1 Graph search path planner

The path computed by the graph search algorithm is visualised using points and curves drawn between them. Points represent the nodes chosen by the algorithm which return the cheapest path. The connections consist of arcs, which are computed using the equations shown in Section 3.4.1. The start node is marked in purple and the final node is green, any node in between these will be blue. The turning radius T_r was set to 6 m for all the scenarios. This is considered close to the actual turning radius of a truck.

The designed graph search algorithm was applied on the scenario described in Section 1.4 to achieve the following reference curve illustrated in Fig. 5.2

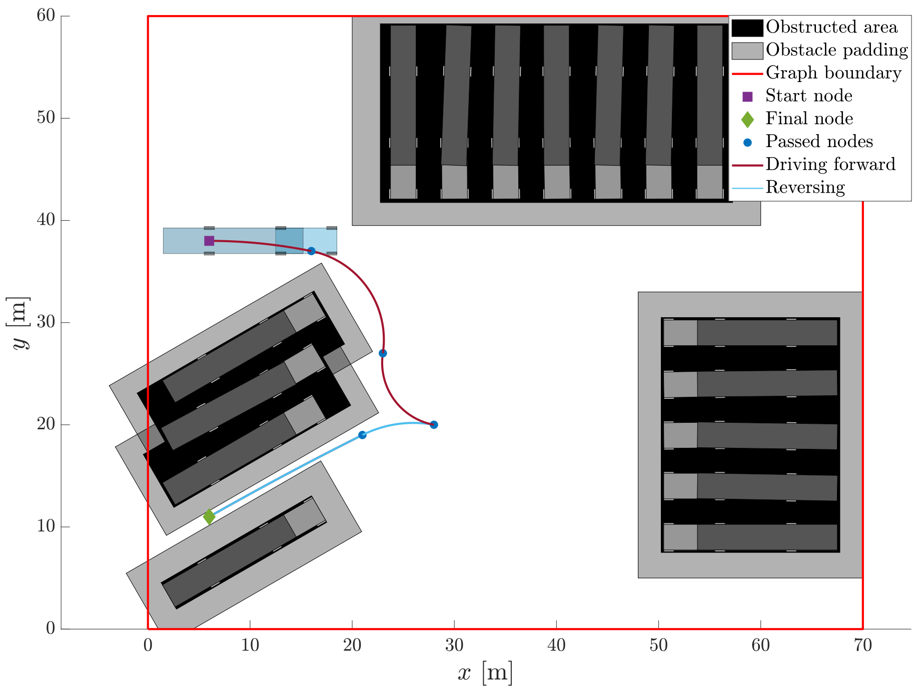


Fig. 5.2. Results generated by the graph search algorithm for the main scenario

The algorithm manages to find a suitable point to begin the reversing. The padding is set to 2 m giving a very narrow path to the final position. This helps the path planner to keep the path centered between obstacles.

For the alternative scenarios, the following reference curves were computed. Scenario A can be seen in Fig. 5.3 and scenario B is found in Fig. 5.4.

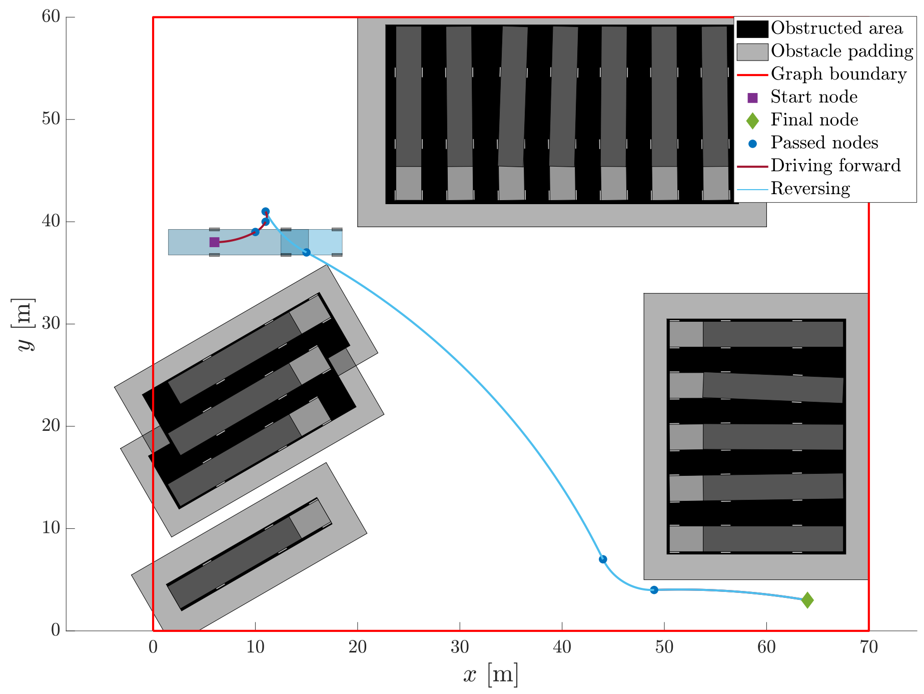


Fig. 5.3. Results generated by the graph search algorithm for alternative scenario A

The reason it chooses to reverse this early in the path, is because it produces the shortest path, while minimising the cost of having a differing heading. For scenario B, the following reference curve was obtained

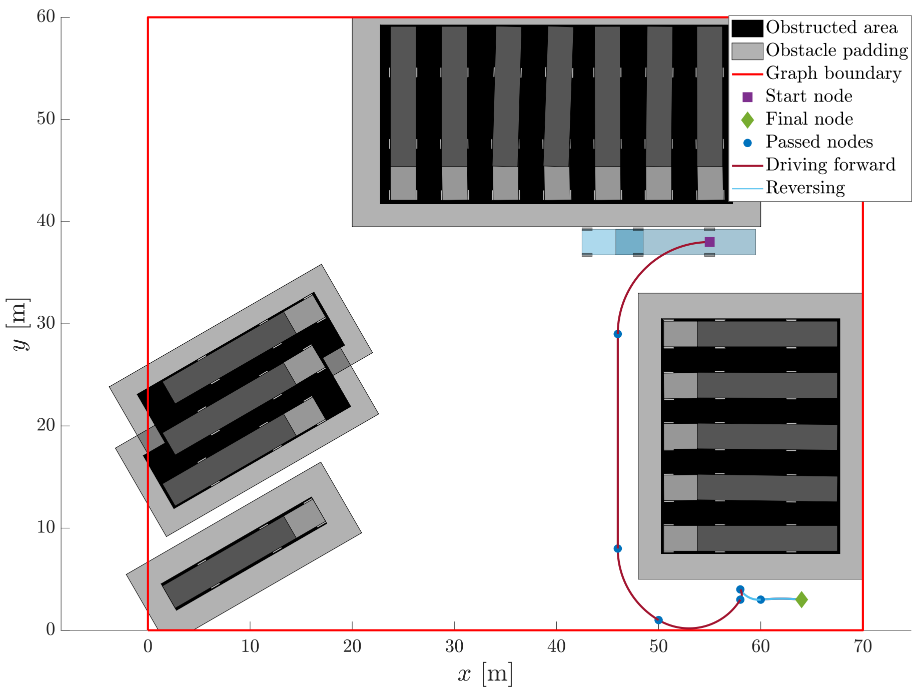


Fig. 5.4. Results generated by the graph search algorithm for alternative scenario B.

In this situation, the issue of the graph search not taking the vehicle size in regard can be seen. Here, the truck will probably end up inside an obstacle, since the rear axle of the trailer is supposed to match the position given by reference curve.

5.1.2 MPC path planner

The weightings in the objective function (4.8) are listed in Table I and kept the same for all investigated scenarios. They, together with the parameters, were designed to give the desired behaviour.

The number of points along the center of the truck were kept as low as possible, any less and the minimum safety margin needed would not leave room for the truck in the parking space.

The velocity limits are designed such that, together with the minimisation of lethargy, promotes driving forward. This turned out to be important in order to get reasonable result. The absolute value of these does not matter, only their relation. The constraint on the steering were designed to match what the trajectory planner is able to follow.

TABLE I
LIST OF WEIGHTINGS IN THE OBJECTIVE FUNCTION IN (4.19)

Weight	**Magnitude**	**Description**
w_1	10^3	Acceleration
w_2	1	Time

TABLE II
LIST OF PARAMETERS USED IN THE OPTIMISATION PROBLEM

Parameter	**Size, unit**	**Description**
N/A	5	Number of points along the center of truck to measure distance to objects from
$u_{1,\text{max}}$	18 km/h	Maximum speed allowed
$u_{1,\text{min}}$	−1.8 km/h	Minimum speed allowed
$u_{\alpha,\text{max}}$	$^{5\pi}/_{18}$	Maximum steering angle allowed
$u_{\alpha,\text{min}}$	-$^{5\pi}/_{18}$	Minimum steering angle allowed
d_{min} or m	2.4 m	Minimum distance to keep to obstacles, also described as padding
z_{min}	0.1	Minimum value of $z(\kappa)$
z_{max}	10	Maximum value of $z(\kappa)$

Since the MPC path planner is in need of an initial guess of the states in order to generate a path quickly, but more importantly, to ensure that the problem is feasible. The path from an A* search provided this for the states corresponding to the position and is plotted in Fig. 5.5.

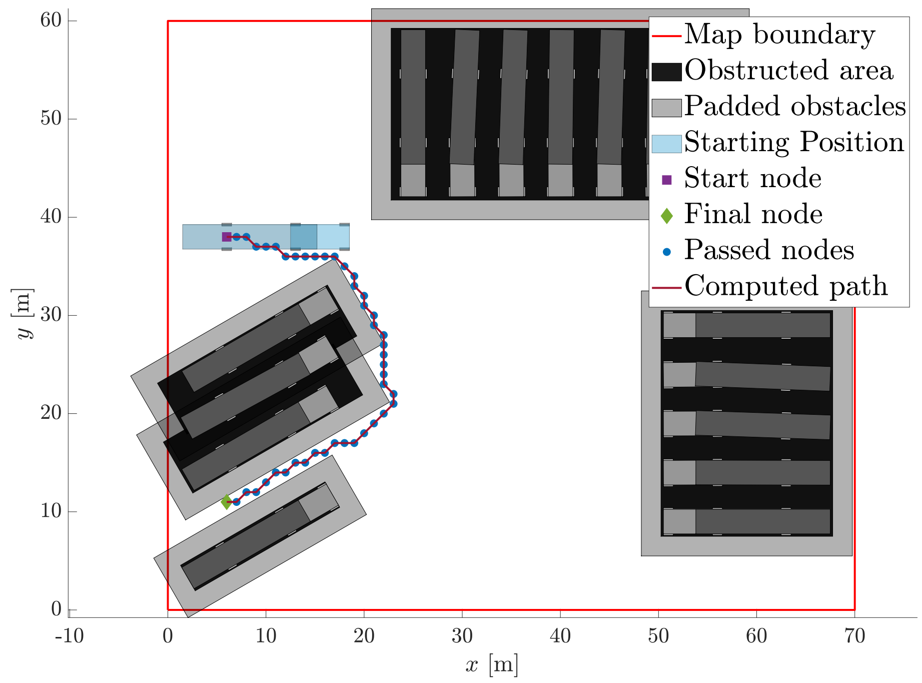


Fig. 5.5. This path is the result of an A* search.

The spacing between two points in the graph in Fig. 5.5 is about 1 m which is a rather high resolution. Since the integrator used, RK4, is fairly exact it can be reduced and still capture the dynamics. It turned out that every 4th sample is enough. This decrease in number of sampling points reduced the time it took to complete the entire problem. The resulting path is plotted in Fig. 5.6.

The solution of the MPC is displayed in orange dots and the upsampled path in burgundy and teal depending on the intended direction of movement. It can be seen that the generated path cuts through the safety margin. This is due to the absence of a constraint on the distance to objects between two samples.

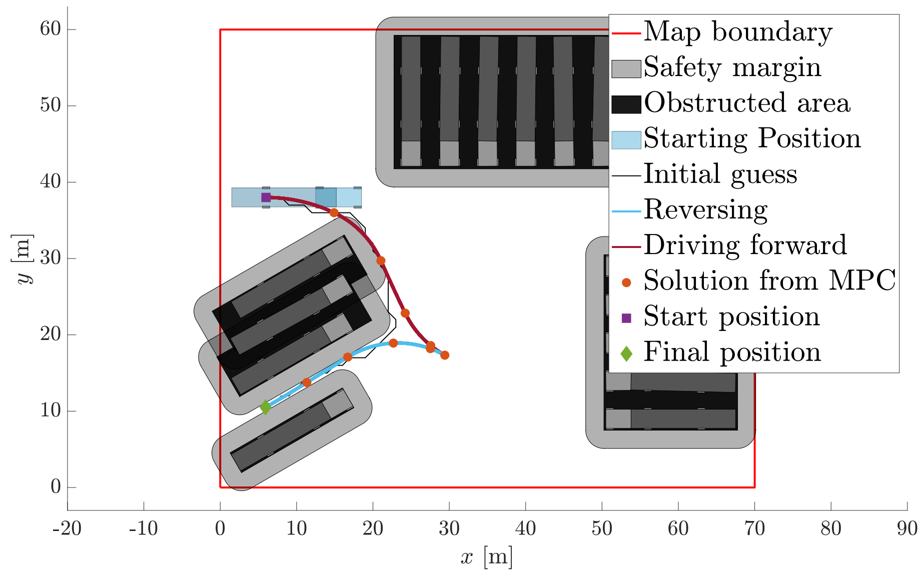


Fig. 5.6. The solution to the optimisation problem (4.9) when using every 4th sample of an A* search as an initial guess.

If the number of points was not decreased the resulting path would be as in Fig. 5.7. The path cuts signifacantly less through the safety margin with the cost of longer computation time.

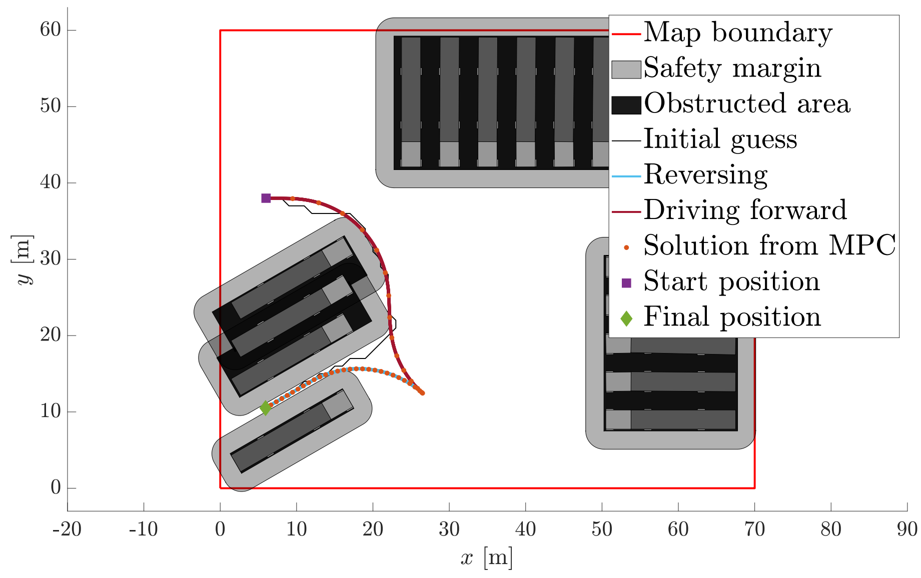


Fig. 5.7. The solution to the optimisation problem (4.9) when using an A* search as initial guess but keeping all samples as initial guess.

When applying the algorithm on the two other scenarios presented in Fig. 5.8, the importance of tuning becomes apparent. In Fig. 5.8(a), it is clear that the planner does not choose to reverse as one would expect. It chooses to reverse early on, even though the objective function is designed to favor forward motion. It is however able to suggest a path that seems to work as a reference for the trajectory planner. On the other hand, the resulting path for scenario B, see Fig 5.8(b), is good in terms of reversing distance and seems to suit the trajectory planner.

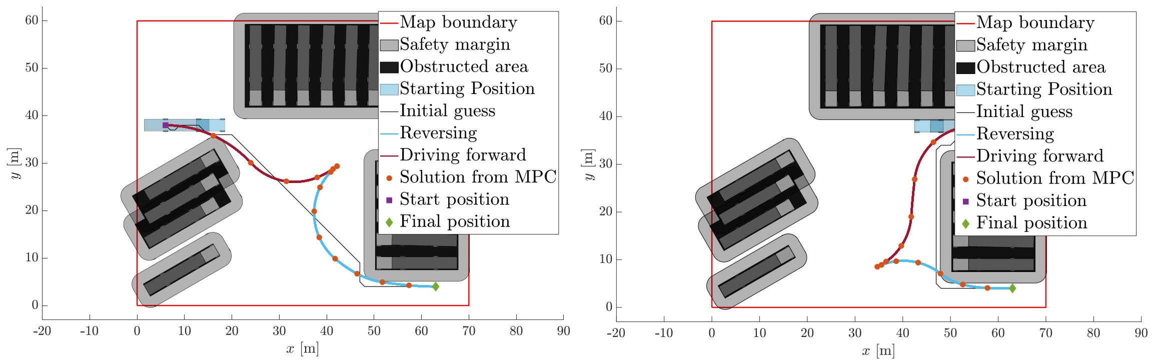


Fig. 5.8. In these two plots the MPC path planner is tested on the alternative scenario A (a) MPC path planner with all samples from the A* search. (b) MPC path planner with every 4th sample from the A* search.

5.2 Trajectory planner

The trajectory planner is given the output of one of the path planner as input, and attempts to follow it under the constraints of the vehicle dynamics. The input of each algorithm is evaluated jointly and presented side by side. Both of the simulations will have the same objective function with the weights listed in Table III and miscellaneous parameters listed in Table IV.

The weightings have been tuned by looking at their effect on the resulting trajectory. The weight that penalise a deviation from the reference has been set to zero since that gave the best results. By not including it in the objective function, the trajectory planner is free to find an even more optimal path than what is given from the path planners. Including it would be of interest if the path has more information than the trajectory planner has access to, which is not the case here. The reference is still used to determine the pose of the semi-trailer truck at the end of each horizon.

The penalty of acceleration is also removed since, in respect to comfort, jerk is more important to minimise.

TABLE III
LIST OF WEIGHTINGS IN THE OBJECTIVE FUNCTION IN (4.19)

Weight	**Magnitude**	**Description**
w_1	0	Deviation from reference
w_2	0	Acceleration
w_3	$1 \cdot 10^{-3}$	Steering rate
w_4	$1 \cdot 10^{-2}$	Jerk
w_5	$1 \cdot 10^{-2}$	Time
w_6	$5 \cdot 10^{-2}$	Target rear
w_7	$5 \cdot 10^{-2}$	Target front

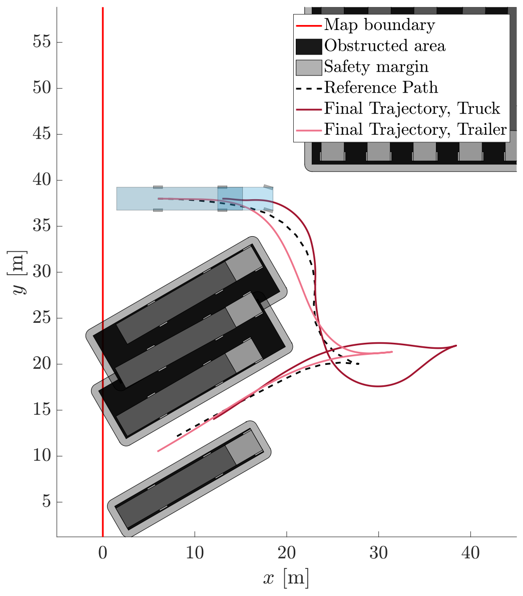
The miscellaneous parameters are partly designed to produce a reasonable behaviour, but also feasible solutions. The horizon ζ_H is set to 25 m which is a bit longer than needed in order to get a smooth trajectory, often 15 m was sufficient. The number of points along the truck, together with the number of obstacles, affected the computation time the most. It was however needed to have a large number of points approximating the shape of the semi-trailer truck, in order to be able to lower the safety margin d_{min}. A too large margin would not leave enough space for the truck to enter the parking space. The other values were tuned in order to get the desired behaviour.

TABLE IV
LIST OF PARAMETERS USED IN THE OPTIMISATION PROBLEM

Parameter	**Size, unit**	**Description**
ζ_H	25 m	Length of reference within horizon
N/A	10	Number of points along the center of semi-trailer truck to measure distance to objects from
$\Delta\zeta$	2 m	Distance between two reference points
$v_{\max}$	7 km/h	Maximum speed allowed
$v_{\min}$	−5 km/h	Minimum speed allowed
$u_{\alpha,\max}$	$2\pi/9$ rad	Maximum steering angle allowed
$u_{\alpha,\min}$	$2\pi/9$ rad	Minimum steering angle allowed
$u_{\mathrm{acc},\max}$	$1\,\mathrm{m/s^2}$	Maximum acceleration allowed
$u_{\mathrm{acc},\min}$	$-2\,\mathrm{m/s^2}$	Minimum acceleration allowed
$\varphi_{\max}$	$\pi/3$ rad	Maximum trailer angle allowed
$\varphi_{\min}$	$\pi/3$ rad	Minimum trailer angle allowed
$d_{\min}$ or m	2.05 m	Minimum distance to keep to obstacles, also described as padding
$\delta_{r,\max}$	10 m	Maximum deviation from target rear position
$\delta_{f,\max}$	10 m	Maximum deviation from target front position
$z_{\min}$	0	Minimum value of $z(k\vert\kappa)$
$z_{\max}$	2	Maximum value of $z(k\vert\kappa)$

The following three figures, Fig. 5.9-5.11, show the results of using the paths from Fig. 5.2 and 5.6 as reference. In Fig. 5.9, the trajectory of the semi-trailer truck is plotted. As can be seen, both references are good enough for the trajectory planner to be able to successfully follow and enter the parking spot in reverse. The left plot is the result of using graph search as a path planner and the right an MPC.

The results are very similar, but it is clear that the path from the MPC path planner coincides better with the trajectory of the trailer. That shows that the model and parameters used in the MPC are more similar to the model used in the trajectory planner. It can also be seen that the suggested reversing points from both path planners are not what the trajectory planner deems as best. The graph search also underestimates the space needed in order to not collide with the obstacles. The trajectory planner needs to drive further before reversing than what is suggested. The MPC path planner on the other hand, with its sense of size of the truck, does not suffer from this problem.



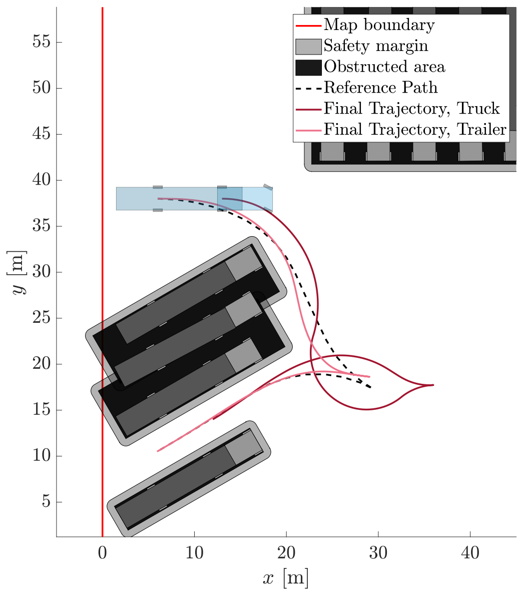


Fig. 5.9. These trajectories were the result from the trajectory planner when the reference path, $(r_x(t|\tau), r_y(t|\tau))$, was generated using a) Graph search b) MPC.

In Fig. 5.10 it is shown how the algorithm plans to position the semi-trailer truck at the end of the horizon. The trajectory traces out both the movement of the center of the front axle $(x_1(t|\tau), y_1(t|\tau))$ as well as the position of the trailer $(x_2(t|\tau), y_2(t|\tau))$. It is able to do so since the reference path includes information about reversing. As described in Section 4.3.3, the target position at the end of the horizon is tangent to the last reference point, i.e. $\mathbf{r}(\kappa + k_H|\kappa)$, and rotated π radians if that point is a reversing point.

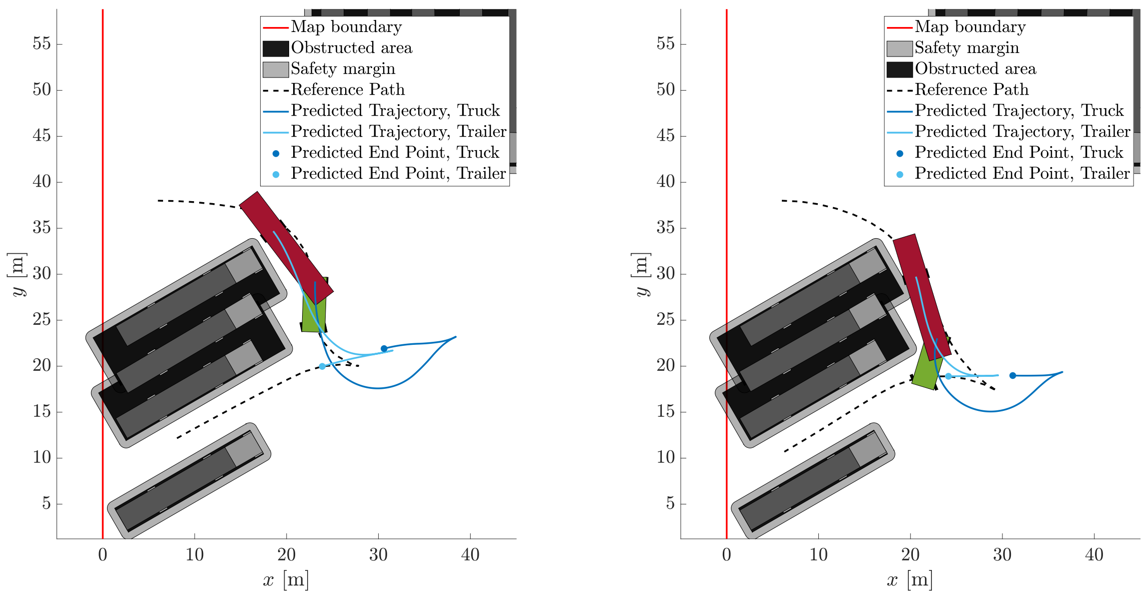


Fig. 5.10. Predictions of the horizon τ_k. The truck plans to let $(x_1(\tau_k + t_H|\tau), y_1(\tau_k|\tau_k + t_h))$ be as close to $(r_x(\tau_k + t_H|\tau), r_y(\tau_k|\tau_k + t_h))$ as possible and $(x_2(\tau_k + t_H|\tau), y_2(\tau_k|\tau_k + t_h))$ to be in a direction tangent to the reverence path. The reference path was generated using a) Graph search b) MPC.

The velocity, acceleration and steering angle are plotted over time in Fig. 5.11. The semi-trailer truck is, as expected, driving as fast as possible since it tries to complete the parking situation in as little time as possible. The downside to this is that the acceleration has high spikes. However, it successfully minimises the jerk by smoothing the intermittent accelerations and brakes to achieve higher comfort. The same is true for the steering, which, when possible, gradually changes in order to have a low steering rate.

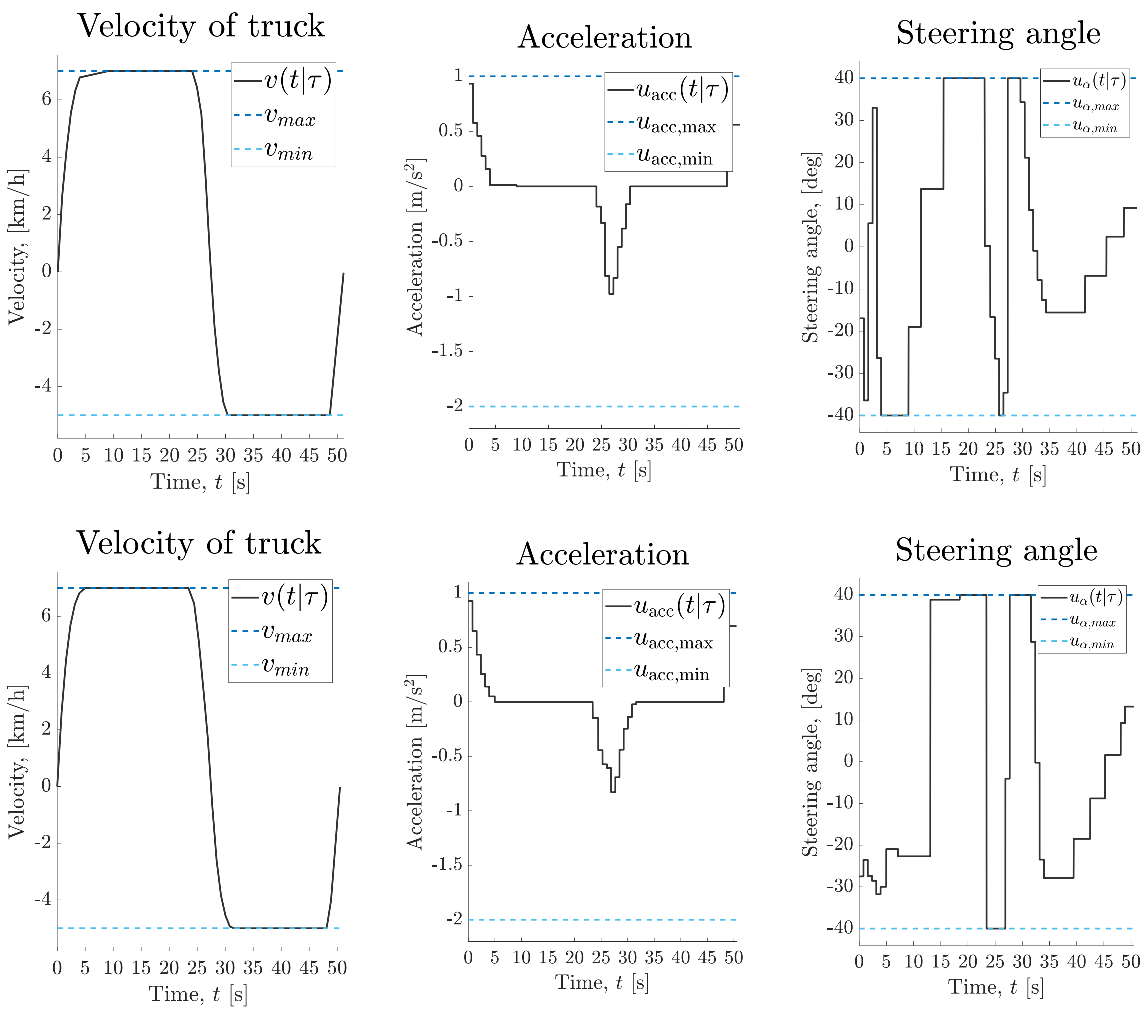
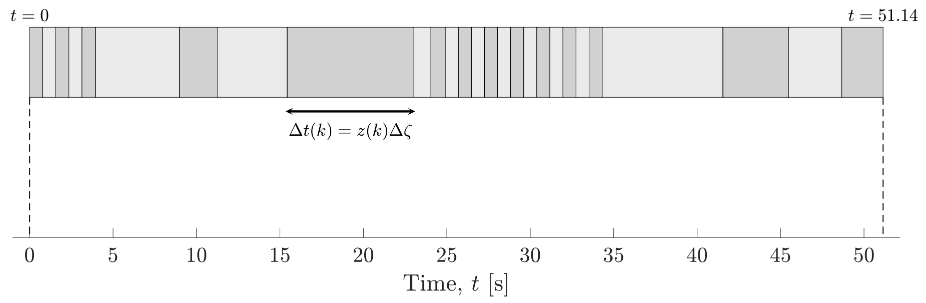


Fig. 5.11. These plots show the speed and control inputs along the parking situation. The reference path was generated using a-c) Graph search d-f) MPC.

In Fig. 5.12 the result of the variable $z(k)$ is illustrated by showing its product with the sampling interval $\Delta\zeta$. This evaluates the sample time at each stage. Most of the samples are shortened to the minimum value of $z(k)$ which means that the lethargy is low. This also shows the total amount of time it takes to complete the scenario which, for the graph search path is 51.14 s, and for the MPC path planner, 50.50 s.

Overall there is only a small difference between the resulting trajectory on the two different path planners. Much is due to the fact that the reference path only is used to suggest reversing point and does not restrict the way to that point. The difference is however much larger when looking at the scenarios A and B, which can be seen in Appendix A-B.



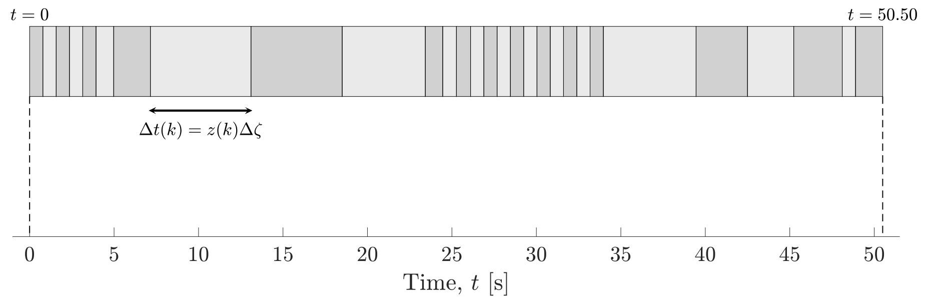


Fig. 5.12. Each time step is adjusted by optimising the variable $z(k|\kappa)$. These plots show the implication of the time consumption after optimising these variables. This is the result of the MPC trajectory planner using the path generated from a) Graph search b) MPC as a reference.

6
Analysis

All designed algorithms manage to solve the main task with realistic results. Once presented with alternative scenarios, some of them show design flaws. This part aims to break down the specific choices leading to these flaws and how they could potentially be improved or counteracted.

6.1 Path planners

Both path planners achieve similar results for the main task. When applied to the alternative scenarios, the results start to differ. This is partly due to the design choices made and the fundamental structure of the algorithms.

The path planners mainly focus on finding a feasible solution to the problem. Merely minimising distance is not a good measurement for what resembles a good path. The same argument can be made for minimising turn angles over the entire process. Therefore, the performance needs to be based on what effect it will have on the trajectory planning.

6.1.1 Graph search path planner

Using graph search for path planning works well if a constant movement towards the target destination can be made. Since a node in a 2D graph can only be traversed once, problems requiring the same node to be visited multiple times can not be solved.

An issue appeared where the algorithm had a lower cost driving straight towards the goal, rather than reversing. Since the change of direction led to a longer path, the cost was never lower than diving straight in towards the goal. The problem was solved by using angular bins. Straight forward and reversing were thus handled separately, due to the angular difference. The different alternatives allowed the goal node to be approached in multiple ways based on the number of bins. Another way to handle this could be to add check points that need to be visited before reaching the target destination, once a check point has been reached, the closed list is cleared so everything can be evaluated again. This would increase the computational complexity and requires knowledge of where the check points should be placed.

Another issue is the design of the cost function. Its focus lies on minimising the distance travelled which is not the desired behaviour. By constructing a cost function that reinforces the wanted behaviour, one would probably achieve better solutions and avoid the issues seen in the scenario A and B.

The computed path for scenario A is a result of minimising driving distance. By changing driving direction early in the process the total distance driven is reduced. Reversing might not necessarily be a downside for autonomous vehicles, as they will probably have equal knowledge of the surrounding in both directions. This does also depend on the dynamics of the vehicle, since semi-trailer trucks are a more stable system when driven forward. If reversing for longer distances is to be avoided, a penalty could be added to the cost function. This way, the algorithm would aim to drive forward for as long as possible.

Scenario B is the result of the graph search not having the size of the vehicle in regard. The suggested three point turn is not in line with the oblong shape of a semi-trailer truck. An attempt to handle this was made, but was deemed insufficient to stay. The proposed solution was to see if there was enough space in front of each node when evaluating the visibility of it. This created more problems and the algorithm instead failed to find a solution. Another suggestion is to also introduce padding to the graph boundary, which would solve the situation presented in Scenario B, but the problem itself would still be present.

Graph search is a suitable tool for complex environments consisting of a multitude of obstacles. Since these are just binary values, the amount does not affect the performance of the algorithm. They can be of complex non-convex shape and the computational effort would still be the same, the padding process would however be more time consuming than using approximated rectangles. One way to make the padding easier, would be if the center of a non-convex object and all points relative to this point are known. One could simply add a small distance to each point to use as padding. This knowledge, is however, not trivial to obtain or approximate.

In this project, the knowledge of the entire environment and its obstacles is kept in the memory at all times, this can become an issue as the graph grows larger. For very large graphs, there should be some filter to keep the memory usage down. By removing useless knowledge or keeping some parts of the graph unknown and let them be discovered over time, this would probably be solved.

6.1.2 MPC path planner

The MPC performs well as a path planner. It produces reasonable results for all tested scenarios partly due to its ability to factor the shape of the vehicle in the planning but also because of its reasonable motion model. The computation time does however scale badly with the number of obstacles and length of path. For each new obstacle, two more non-convex constraints are added. Also, for every $\Delta\zeta$-step the number of object related constraints as well as modelling and control signal constraints are increased. All of which are non-convex, yielding a slow optimisation.

The number of $\Delta\zeta$-steps scales linearly with the length of the path and can not be lowered more than about $\frac{1}{4}$ of the original number of samples. Below that point the vehicle will cut through the safety margin too much between two samples. The number of obstacles could probably be lowered by only considering the ones of interest. How to decide which to include is however not trivial since only those objects that are not affecting the solution can be removed. For a large parking lot, one might be able to remove those objects that are not in line of sight of the points in the A* solution. In some configuration of obstacles that would mean a significant reduction in the number of obstacles. Although in the case of the scenarios presented in this project, there would not be any difference.

One drawback of using MPC to solve the problem of path planning is that the solver needs an initial guess of the solution. It was solved by considering the positions $(x_1(k), y_1(k))$ to be the solution of the A* algorithm, $u_1(k) = 0$, $u_2(k) = 0$, and $\theta_1(k)$ a gradient transition from θ_0 to θ_f for all k. The latter may seem unnecessary, but proved to be essential for yielding a feasible solution. The formulation did however cause an issue when the starting angle and target angle were equal (Scenario B). The solver was not able to find a feasible solution.

To remedy the situation a small noise of magnitude $0.1°$ centered around zero was introduced. At first, the small difference in starting point of the solver resulted in different solutions with similar cost. To remove the stochastic behaviour, the weighting of acceleration was increased which probably joined some of the local minimums, yielding the same solution each time the path planner was run. There is however a chance that for some scenarios, that have not been investigated in this project, need an even higher noise in order to return a feasible solution, which in turn may lead to a stochastic behaviour. Although, if the objective function is well defined, all solutions should be acceptable and traceable for the trajectory planner.

6.2 Trajectory planner

The result of the trajectory planner looks promising. It performs well on the reference from both types of path planners. It is however a bit constrained in the way that the position at each horizon must end up within a specific area that is decided by the path planner and the parameters δ_r, δ_f. This in turn affects where the truck starts to reverse from. It can be seen in the plots in Fig. 5.9 that the reversing point suggested by both of the path planners is changed by the trajectory planner.

The suggested reversing point of the MPC path planner is affected by both its model and its objective function. As it is now, the path planner favors a short distance of reversing. Although, even if the objective function was better tuned, the reversing points would not coincide in other scenarios, since the models of the path planner and trajectory planner differs. The same is true for the path planned using graph search. Since its model does not have the same physical behaviour as the one used in the trajectory planner, it will at some point suggest a bad reversing point. This is clearly shown in the path from scenario A and B.

If the reference path is bad enough to yield issues with feasibility for the trajectory planner, there is still a possibility to remedy the situation by increasing the parameters δ_r, δ_f. This results in a less constrained final state. There is also a possibility to remove these two states completely from the problem formulation and only considering minimising the distance to the target state. This is the same as setting $\delta_{r,\max}, \delta_{f,\max}$ to infinity. This does, however, introduce more tuning to force the final state to be reasonably close to the target state.

Another solution to the infeasibility is to increase the horizon. The longer the horizon, the more time the truck has to position itself at the final state. The downside to increasing the horizon is that it increases the number of samples used within each horizon, yielding a longer computation time. The variable $z(k)$ also has the possibility to extend the duration of each sample and in turn, extend the horizon in time. By increasing the maximum size of $z(k)$, the truck can in theory take as long time as it wants. There is however a significant downside to allowing a too large $z(k)$. Increasing the time between two samples hinders the ability to correctly avoid obstacles, since the distance to obstacles is only taken into account at each sample. This means that the more sparse sampling, the greater the risk of collision. The maximum time between two samples therefore needs to be designed with great caution.

By carefully testing different weightings it was concluded that the trajectory planner was better off not minimising the distance to the reference path. The usage of simplified models in both path planners is argued to be the reason. By only considering the last position on the reference path as the target position, the trajectory planner is free to find a trajectory that better follows the dynamics of the vehicle. This means that the target state that is defined from the reference path needs to contain appropriate information. The information should include the best orientation at that point in order to successfully arrive at the positions that are not yet visible to the trajectory planner. Since this type of information is included in the reference paths, the trajectory planner is able to know that it needs to reverse without knowing where it should end up later.

6.3 Conclusion

The designed solutions are deemed to be viable for the presented scenario. The task is solved with a satisfactory result while avoiding collision.

The two path planners are both suitable for solving the issue, but have their own individual strengths. The graph search is more suitable for complex environments with a multitude of obstacles, while this would increase the computational effort of the MPC path planner. The MPC path planner is better at taking the vehicle dynamics in regard. Since the graph search estimates the vehicle as a point mass, it can potentially create paths which lead to dead ends as seen in Fig. 5.4.

Obstacle avoidance is of great importance for both path and trajectory planning. By using a discrete trajectory planner, objects are not taken in regard between samples. This means that obstacle avoidance can not be guaranteed. The larger the time between two samples, which is affected by $z(k)$, the more likely a collision will be.

Both planners generate smooth curves which to some extent resemble what the trajectory planner deems to be optimal. This means that any given point along the reference curve is a good enough estimate of where the vehicle should be. I.e, instead of strictly following the reference curve, it is sufficient to provide an end pose for each horizon. This gives the trajectory planner more freedom in order to reach the desired pose.

7 Discussion

In this project, the trajectory planner penalises deviation from an end pose at each horizon, rather than minimising the distance to a given reference curve. This approach is suitable if the path planner has equal or less knowledge than the trajectory planner. One example where minimising the distance to the reference curve is of interest is when the trajectory planner is unable to compare all obstacles simultaneously. Assuming the path planner has knowledge of all objects when planning, it is considered to have more knowledge about the environment than the trajectory planner has.

If the trajectory planner does not minimise the distance to the reference path and does not take all objects into account, there might arise a situation where it plans to cut a corner where an object, which it not yet can see, is, in order to save time. This might lead to an infeasible solution or, even worse, a collision. By penalising the deviation from the reference curve, the trajectory planner would not be able to plan trajectories deviating too much from the reference curve, and hence indirectly avoid obstacles.

The obstacles in the environment are estimated using rectangles, which is an approximation that has a lot of downsides. For complex objects, with non-convex shapes, a multitude of rectangles might be needed in order to describe them accurately. This leads to an immense increase in computational effort for the algorithms using MPC. However, its effect on the graph search is negligible as stated in Section 6.1.1. If instead an attempt of using fewer rectangles with the cost of worse approximation is made, it would either fail to cover the entire object or remove useful space. This could at worst lead to collision, or simply removing the only possible solution by obstructing space needed to solve the task at hand. On the other hand, in this specific case of a parking lot for semi-trailer trucks, rotated rectangles turns out to be very suitable approximations of shape.

For the designed algorithm to be truly useful, there are a few things that need to be further evaluated. As of now, the algorithm assumes perfect knowledge of the surroundings. By simply adding sensors to the vehicle, one would not obtain the information about the environment used in this project. A radar or camera would not provide any data of what is behind corners, since this is obstructed by the object. To accurately describe the entire surroundings is a complex process, which needs to be solved in order for this implementation to work.

The computation time for both path planners is sub optimal for real time implementation. The graph search could potentially utilise a lookup table for the heuristics to avoid computing these for each node. This would also allow more advanced heuristics to more accurately estimate the remaining cost. An example is the Reeds-Shepp curves commonly used for vehicles which are able to move both backwards and forwards.

An improvement of the cost function could also reduce the computational effort of the graph search. By accurately describing the remaining cost to the goal node, irrelevant nodes could be filtered out and neglected completely. Issues occurred where nodes close to the goal node always had a low cost, even though the assigned heading was too far from the desired one. The attempt of penalising the heading was insufficient to completely remove this behaviour.

The MPC could run faster by only handling the objects in the vicinity of the truck. This reduction in computational effort could potentially allow larger environments. In this project, all obstacles were of relevance during the entire process and were few enough for it to not have a large enough impact to make a difference. Some logic to categorize objects as relevant would probably be the best way to solve this. It is, however, critical to not disregard objects of relevance for safety reasons. Although, together with penalising the distance to the reference curve, the solution could still be feasible and collision free. This in turn puts more emphasis on the model of the path planners since they would restrict the freedom of the trajectory planner.

7.1 Future work

There are two main issues that needs to be addressed for this type of application to be a reality. These are guaranteed collision avoidance and minimal computation time.

The presented implementation of trajectory planner assures that no obstacle will be hit at a stage, but is unable to ensure that no collision occurs between two stages. However, based on the sampling interval and maximum velocity one should be able to calculate their link to the safety margin and points on the vehicle such that no collision can occur. Too sparse sampling, would lead to a wider safety margin, which might obstruct the intended goal position.

Instead of increasing the sampling interval and reducing the spacing between points on the truck, one should investigate the possibility to check for collision between two stages. This way, one could ensure a collision free trajectory. Currently, there is enough confidence that the solution is collision free, but it can never be guaranteed.

The computation time can be lowered in many different ways. First of all by implementing the algorithms in a different language, e.g. C/C++. Secondly, by making the MPC problem formulation more convex or entirely convex. From our investigation, it does not seem to be possible to represent the semi-trailer truck dynamics entirely convex although there most certainly is a possibility to simplify the model further. The benefits of having a convex model are many and would remove some issues encountered during project. E.g. different, or even infeasible, solutions depending on initial guess to the Ipopt.

Bibliography

[1] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, *Chapter 4 - Path Planning*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds. Butterworth-Heinemann, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128042045000044

[2] M. Nolte, M. Rose, T. Stolte, and M. Maurer, "Model predictive control based trajectory generation for autonomous vehicles — an architectural approach," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 798–805.

[3] P. Zhang, "Chapter 19 - industrial control system simulation routines," in *Advanced Industrial Control Technology*, P. Zhang, Ed. Oxford: William Andrew Publishing, 2010, pp. 781 – 810. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781437778076100191

[4] K. A. Abdel-Malek and J. S. Arora, *Chapter 4 - Recursive Dynamics*, K. A. Abdel-Malek and J. S. Arora, Eds. Boston: Academic Press, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124051904000040

[5] G. Sasi Kumar, B. Shravan, H. Gole, P. Barve, and L. Ravikumar, "Path planning algorithms: A comparative study," 12 2011.

[6] J. Petereit, T. Emter, C. Frey, T. Kopfstedt, and A. Beutel, "Application of hybrid a* to an autonomous mobile robot for path planning in unstructured outdoor environments," 01 2012.

[7] Y. Singh, S. Sharma, R. Sutton, D. Hatton, and A. Khan, "A constrained a* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents," *Ocean Engineering*, vol. 169, pp. 187 – 201, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0029801818311193

[8] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, *Chapter 7 - Autonomous Guided Vehicles*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds. Butterworth-Heinemann, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128042045000044

[9] H. Febbo, P. Jayakumar, J. L. Stein, and T. Ersal, "Real-time trajectory planning for automated vehicle safety and performance in dynamic environments," 2020.

[10] L. Sciavicco and B. Siciliano, *Trajectory Planning.* London: Springer London, 2000, pp. 185–212. [Online]. Available: https://doi.org/10.1007/978-1-4471-0449-0_5

[11] S. Edelkamp and S. Schrödl, *Heuristic Search - Theory and Applications.*, 01 2012.

[12] X. Li, C. Claramunt, and C. Ray, "A grid graph-based model for the analysis of 2d indoor spaces," *Computers, Environment and Urban Systems*, vol. 34, pp. 532–540, 11 2010.

[13] J.-W. Choi, "An efficient heuristic estimate for non-holonomic motion planning," 2012.

[14] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," vol. 28, pp. 239–253, 01 2005.

[15] Lozano-Perez, "Spatial planning: A configuration space approach," vol. C-32, pp. 108–120, 2 1983.

[16] D. Chibisov, E. Mayr, and S. Pankratov, "Spatial planning and geometric optimization: Combining configuration space and energy methods," 09 2004, pp. 156–168.

[17] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *J. Artif. Intell. Res. (JAIR)*, vol. 39, 01 2014.

[18] L. De Filippis and G. Guglieri, *Advanced Graph Search Algorithms for Path Planning of Flight Vehicles*, 02 2012.

[19] H. Kim, D. Kim, J.-U. Shin, H. Kim, and H. Myung, "Angular rate-constrained path planning algorithm for unmanned surface vehicles," *Ocean Engineering*, vol. 84, pp. 37 – 44, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0029801814001292

[20] P. Yap, "Grid-based path-finding," 09 2003.

[21] G. Mathew, "Direction based heuristic for pathfinding in video games," *Procedia Computer Science*, vol. 47, pp. 262–271, 12 2015.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[23] A. Felner, U. Zahavi, R. Holte, J. Schaeffer, N. Sturtevant, and Z. Zhang, "Inconsistent heuristics in theory and practice," *Artificial*

Intelligence, vol. 175, no. 9, pp. 1570 – 1603, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370211000221

[24] A. Nash and S. Koenig, "Any-angle path planning," *AI Magazine*, vol. 34, no. 4, pp. 85–107, Sep. 2013. [Online]. Available: https://aaai.org/ojs/index.php/aimagazine/article/view/2512

[25] A. Nash, S. Koenig, and C. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d." vol. 1, 01 2010.

[26] L. Di Jasio, *12.2.13 Bresenham Algorithm.* Elsevier, 2012. [Online]. Available: https://app.knovel.com/hotlink/khtml/id:kt00B7BV5J/programming-16-bit-pic/bresenham-algorithm

[27] N. R. Ruchika, "Model predictive control: History and development," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 4, pp. 2600–2602, 1 2013.

[28] N. Murgovski, G. R. de Campos, and J. Sjöberg, "Convex modeling of conflict resolution at traffic intersections," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 4708–4713.

[29] J. Karlsson, N. Murgovski, and J. Sjöberg, "Computationally efficient autonomous overtaking on highways," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2019.

[30] A. Wächter, "Short tutorial: Getting started with ipopt in 90 minutes," in *Combinatorial Scientific Computing*, ser. Dagstuhl Seminar Proceedings, U. Naumann, O. Schenk, H. D. Simon, and S. Toledo, Eds., no. 09061. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2009/2089

[31] U. Larsson, C. Zell, K. Hyyppa, and A. Wernersson, "Navigating an articulated vehicle and reversing with a trailer," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, May 1994, pp. 2398–2404 vol.3.

A

Alternative scenario A

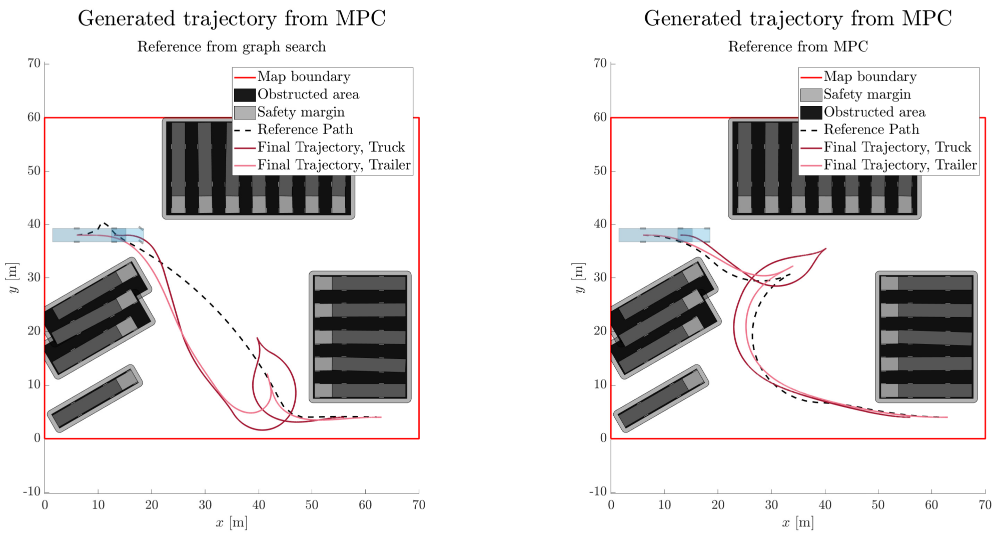


Fig. A.1. Generated trajectory for scenario A with reference from a) Graph search b) MPC

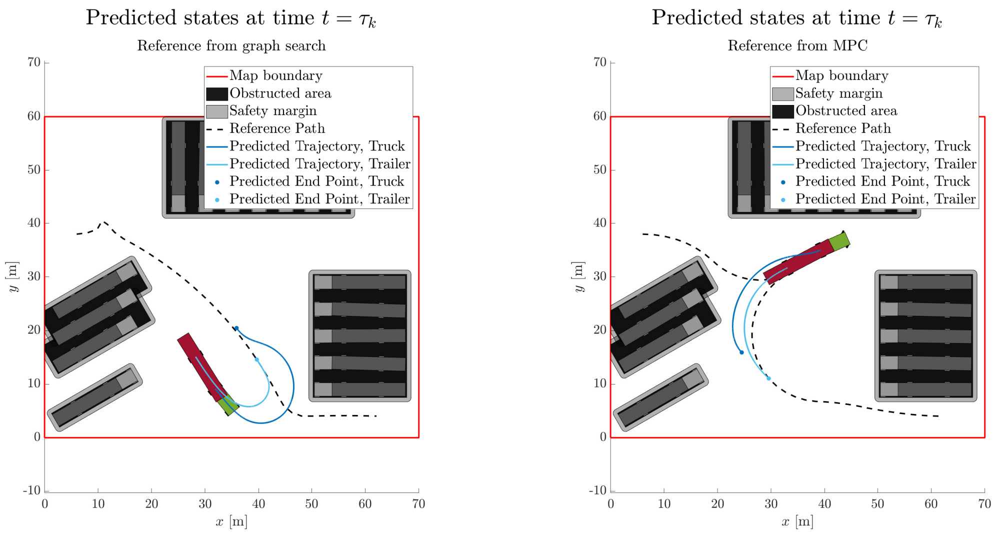


Fig. A.2. Predicted states for scenario A with reference from a) Graph search b) MPC

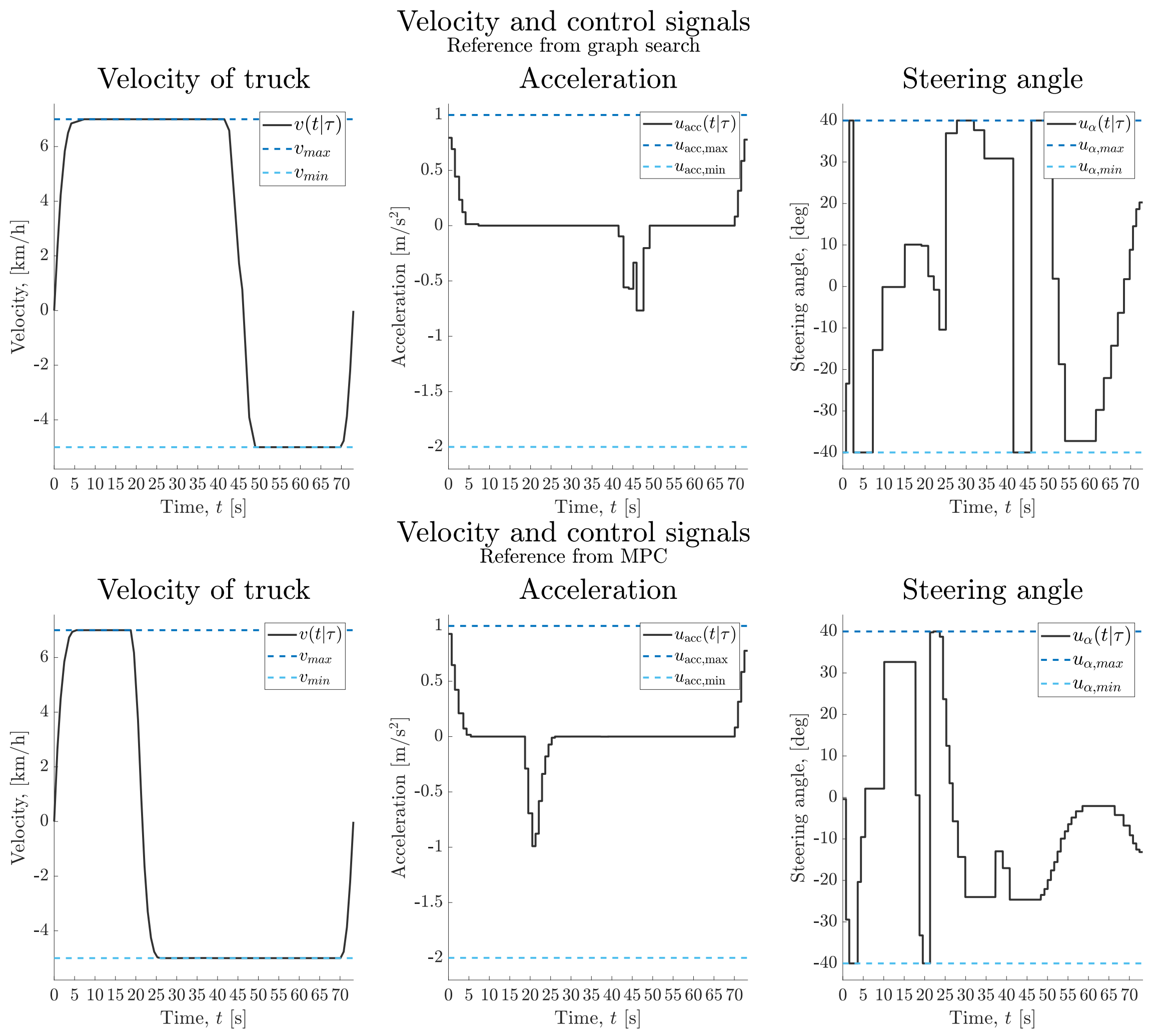
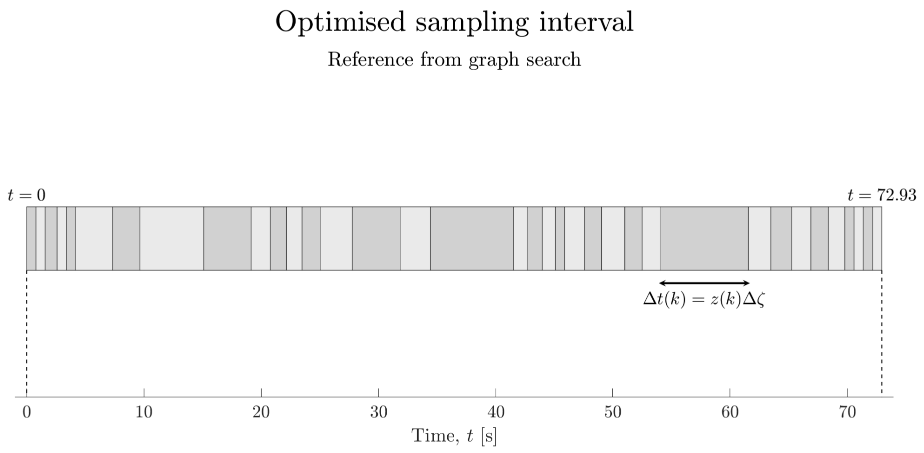


Fig. A.3. Final control signals for scenario A with reference from a-c) Graph search d-f) MPC



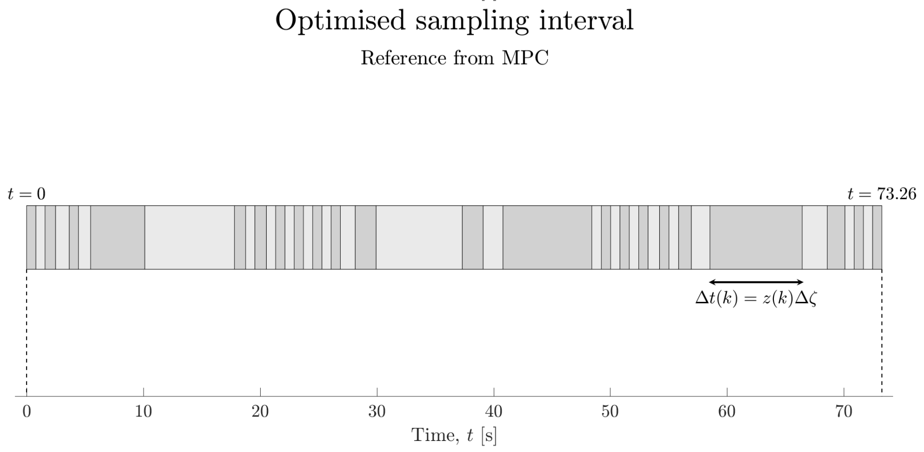


Fig. A.4. Time control for scenario A with reference from a) Graph search b) MPC

B

Alternative scenario B

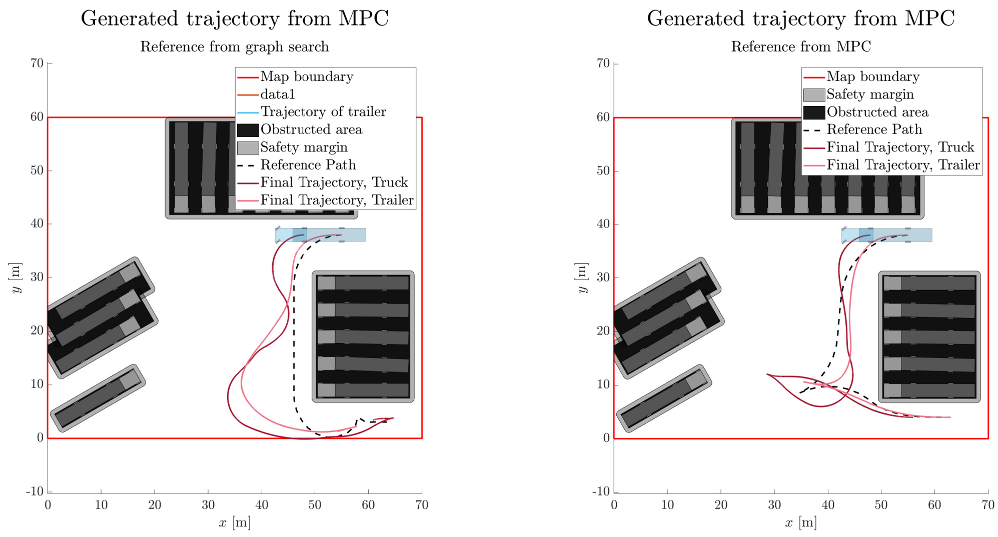
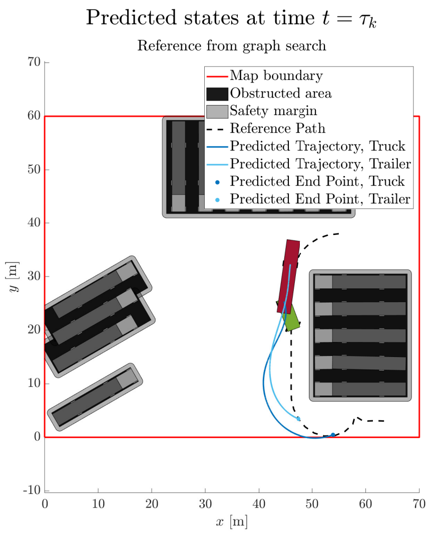


Fig. B.1. Generated trajectory for scenario B with reference from a) Graph search b) MPC



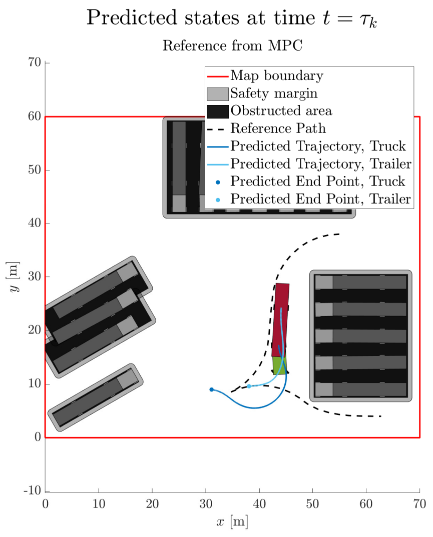


Fig. B.2. Predicted states for scenario B with reference from a) Graph search b) MPC

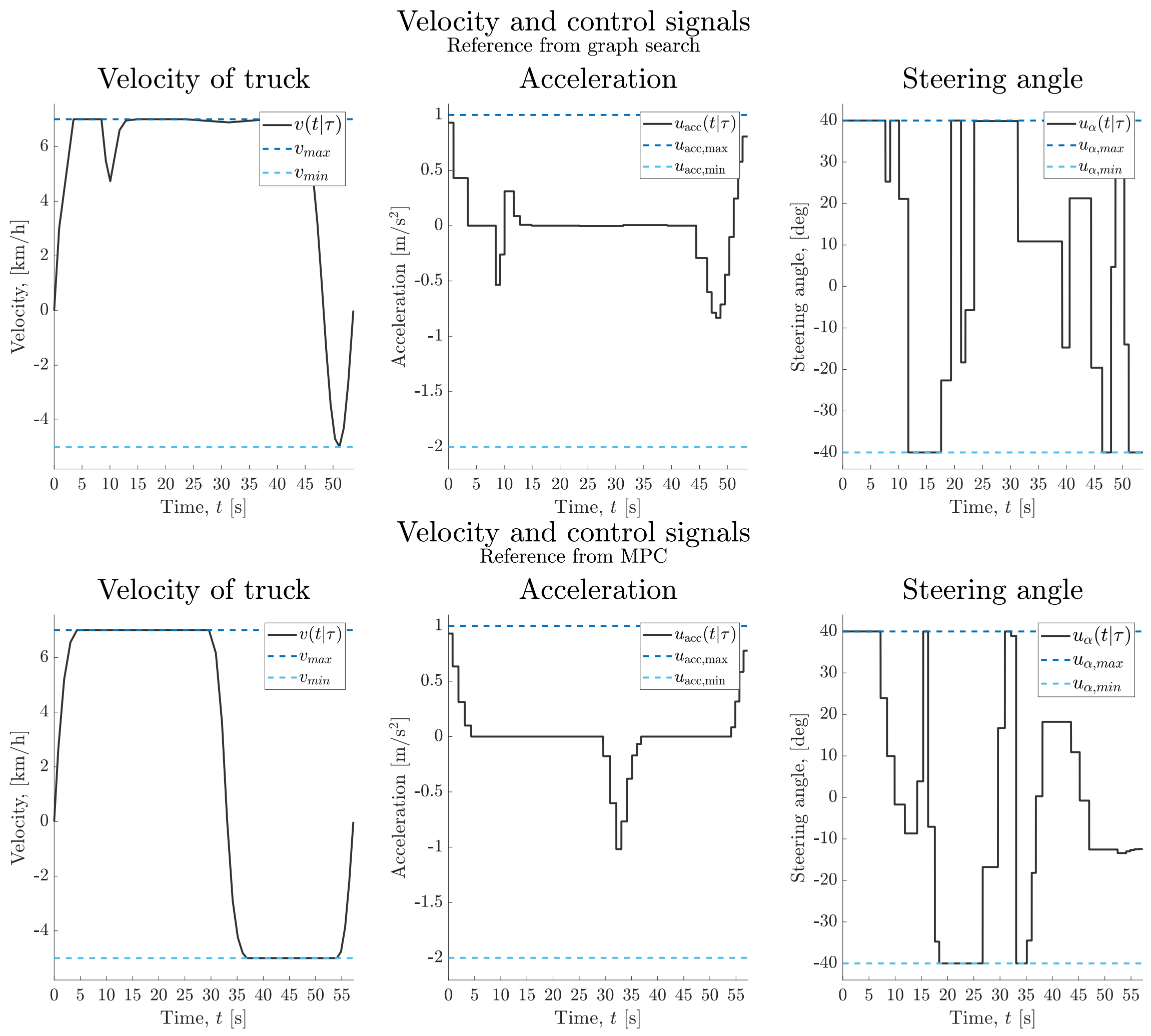
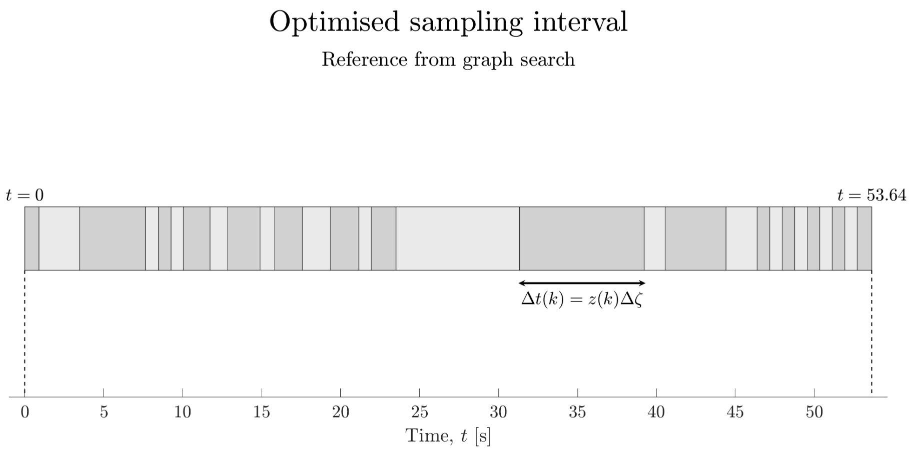


Fig. B.3. Final control signals for scenario B with reference from a-c) Graph search d-f) MPC



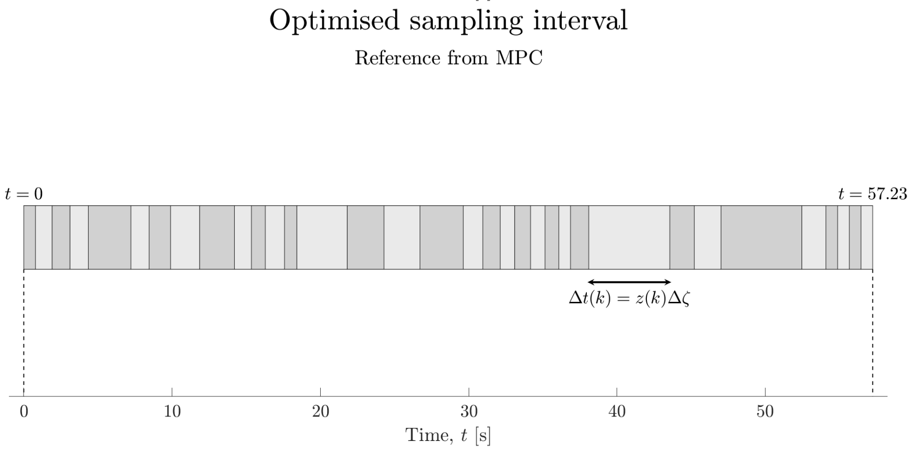


Fig. B.4. Time control for scenario B with reference from a) Graph search b) MPC