POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Trajectory Planning and Motion Control of Autonomous Electric Boats

Supervisors

Mauro BONFANTI, PhD

Candidate

Vinicius Massaki BENEVIDES KANAOKA

March 2023

Abstract

For cities with ample access to rivers and canals it is possible to use them as an alternative to reduce the load on land transportation infrastructure such as bridges and roads. Traditional water-based transportation relies on the pilot to plan the trajectory and control the vessel, both of which depend on the experience of the pilot and are prone to human error. This thesis aims to address the generation of feasible trajectories and their tracking applied to autonomous electric boats for the transportation of people and goods on canals. The trajectory tracking is divided in two stages: path planning and velocity generation. The first stage generates a feasible path by means of a Dubins curve based RRT*, a sampling-based path planning algorithm that is able to take into account constraints on the curvature of the watercraft. The second stage generates a smoothed trapezoidal velocity profile considering bounds on surge velocity and acceleration of the vessel and assigns it to the reference path acting as the time law that defines the trajectory. Subsequently, this trajectory is given as a reference to a nonlinear model predictive controller (NMPC) local motion planner that solves an optimization problem for the best control input able to track the reference while avoiding collisions with walls and obstacles.

Key Words: Autonomous Electric Boats, Global Trajectory Planning, Optimal Control, Trajectory Tracking, Collision Avoidance

Acknowledgements

Gostaria de agradecer minha família pelo apoio que eu tive em toda minha vida e principalmente nessa período de duplo diploma em que tive que viver sozinho em outro país.

Também gostaria de agradecer meus amigos, pois passamos por muitas coisas na faculdade e definitivamente a minha experiência teria sido pior sem eles.

Finally, I would like to thank my supervisor Mauro Bonfanti for the assistance during the development of this thesis.

Vinicius Benevides

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Description of the subject

A consequence of population growth and tourism for cities is the increase of load on infrastructures, including transportation. For coastal and riverside cities the canals can be used as an alternative for land transportation, thus reducing the traffic on roads and bridges.

Traditionally, human operated vessels are used for transportation of goods and people on these urban canals. In this case the pilot is responsible for planning the path and steering the boat avoiding other boats while respecting regulations for collision prevention and speed limits. However, human operation is highly dependant on the pilot's experience and condition. One consequence is that human errors occur and could lead to accidents causing delays or harm to occupants.

An alternative is to use autonomous electric boats equipped with navigation and control systems able to plan trajectories and follow them without the need of human interaction, using on-board computers and sensors.

The immediate benefits of such change are the removal of human errors and variability in decision making. Furthermore, a wide adoption of autonomous boats could increase the efficiency of canals by using fleets of boats able to share information and take decisions as a group, removing the risk of collision between them.

This thesis was done as part of a project at The Marine Offshore Renewable Energy Lab (MORE Lab) at Politecnico Di Torino. In principle the aim of the project is to deploy a fleet of autonomous electric boats for transportation on urban canals. The motivation of this thesis is to apply a methodology to safely control a single vessel inside a canal.

1.2 Objectives

The objective of this thesis is to apply the algorithms necessary to guide an autonomous boat in a safe manner from a start location to a destination in a way suitable to be deployed on cities with canals. In order to accomplish this task, the work is divided in two parts: global trajectory planning and local motion planning.

The global trajectory planning is carried out offline using information known a priori about the environment and the planned trajectory must meet the following requirements:

- The path from start to goal should be collision free.

- The minimum distance from the path to walls must be higher than a safe distance.
- Must obey constraints on the vessel's radius of curvature, velocity and acceleration.

The local motion planning is performed online using information from on-board sensors to detect changes in the environment. The controller used must obey the following requirements:

- The controller should be able to track the reference trajectory even in the presence of obstacles.
- In the presence of obstacles, the control input should be selected to avoid collisions.
- States and control inputs should be subject to constraints.

Both parts can be applied to any type of boat, not necessarily electric. The type of actuation is relevant for the boat design since it affects control input bounds and geometry of the boat to accommodate the electronics. However, the design of the boat is not one of objectives of this project and an existing boat design will be used for testing.

This project focuses on the guidance of a single boat and obstacles are assumed to be known with the use of on-board sensors. One extension of this work is to include sensors with their effective range and noise and modify the controller to be more robust in the presence of noisy measurements. Another possibility is to consider a fleet of autonomous boats and tackle the guidance in a swarm-like manner with the integration of sensor data from different vessels and controlling them to achieve a shared objective.

Chapter 2

State of the Art

Autonomous vehicles consist of perception, decision-making and control system [1] and are able to function without human intervention by collecting information about the surrounding environment and taking decisions accordingly. Among the enabling techniques for such vehicles is the autonomous navigation system responsible for path planning and the control system responsible for performing the objective task.

This chapter is divided in two parts introducing the background in path planning and control of autonomous vehicles. The studies shown are not limited to watercrafts since there is overlap in the techniques used regardless of the type of vehicle.

2.1 Path Planning

The procedure of finding a collision free path from a initial state to a desired goal state is called path planning and it is one of the most important steps for autonomous vehicles. According to Vagale et al. [2] it is possible to define global path planning and local path planning. The former utilizes known data such as the location of static obstacles to create a global reference path. The latter utilizes data obtained online using sensors for perception and focuses on replanning locally a portion of the reference path to avoid new obstacles. In this section procedures for global path planning will be discussed.

The Dijkstra algorithm proposed in 1959 was one the first grid search path planning algorithms for finding shortest paths between two nodes in a weighted graph. This technique can be extended for autonomous vehicles by transforming the configuration space (2D) in a grid and expanding the path through its cells. This algorithm was modified by Hart et Al. [3] to create A^*, which introduced an heuristic to make the path search more efficient by assigning a cost from a cell to the goal and traversing cells with lower cost first. While these algorithms could find optimal paths, they could not be easily applied to all kinds of vehicles since the paths found were composed by straight segments which is not suitable for vehicles subject to kinematic constraints. One way to address this issue is to apply post-smoothing of the path to guarantee at least heading angle continuity.

Nonetheless, the complexity of search algorithms increases with the dimension of the configuration space which is not ideal. In contrast, sampling-based algorithms do not suffer from this problem since paths are generated by randomly sampling points in the search space which demonstrated to be efficient for highly dimensional configuration spaces.

One such algorithm is the Rapidly-Exploring Random Trees (RRT), proposed by LaValle in 1998 [4]. This algorithm will be further discussed in the following chapters and its main advantage is that it can be applied to nonholonomic and kinodynamic planning with the

choice of proper steering functions when connecting randomly sampled points.

Karaman and Frazzoli [5] proposed in 2011 a modification of RRT called RRT*. This new algorithm is able to find optimal paths which was not possible with the original implementation due to the lack of costs associated to paths. In the same year, Karaman et al. implemented a version of RRT* with a Dubins curve steering function that was able to generate low cost paths suitable for curvature constrained vehicles [6].

Webb and Berg presented in 2013 [7] the kinodynamic RRT* algorithm. In their work an LQR optimal control problem is solved to connect a new sample to the solution tree. This was done considering controllable systems with linear dynamics but can be generalized for nonlinear system through linearization. The kinodynamic steering function may also be performed with other optimal control solvers like MPC and its variants.

2.2 Control and Collision Avoidance

There are studies for the control of autonomous marine vessels in open waters, however the challenges faced in urban environments differ since they consider narrow and crowded paths. In this case, the problems are closely related to those faced by mobile robots whose studies focus on locomotion in environments with both static and dynamic obstacles like humans. In this section control strategies for trajectory tracking will be discussed.

The simplest control strategies are not able to take into account dynamics and limitations of the boat, this is the case of PID. This linear controller is simple to tune and it can be applied for trajectory tracking by using two PID controllers tuned separately for longitudinal and lateral tracking [8]. This approach falls short for two reasons. The first problem is that the controller has no knowledge of the boat's dynamics and its limitations and acts only on the feedback error between actual and reference trajectories and this could lead to control inputs that are not feasible. The other problem is that for systems with nonlinear dynamics which is the case for boats the performance is not exceptional which could be solved with the use of more advanced techniques such as gain scheduling with a set of PID controllers that are switched depending on the situation. Another remark about PID is that collision avoidance is not achieved at the control level but by a local path planner that modified the original path around obstacles

Khatib [9] introduced in 1985 the artificial potential field (APF) algorithm, one of the earliest attempts at including collision avoidance at the control level. In this algorithm the autonomous vehicle moves in a field of forces where the goal point is an attractive pole and obstacles are repulsive surfaces. The steering action is chosen to get closer to attractive poles while avoiding repulsive surfaces. The classic APF is fast but suffers from oscillations and being trapped in local minima [10].

Fox et al. [10] introduced in 1997 the dynamic window approach (DWA), a reactive collision avoidance algorithm for mobile robots that considers dynamic limitation based on the robot's model. This algorithm acts on the velocity space and through a two stages procedure reduces it to a set of feasible velocities and selects the best according to an objective function. In the first stage all of the velocities that generate circular trajectories are considered, then only velocities whose trajectory do not lead to obstacles are kept and finally the dynamic window restricts these velocities to those that can be reached based on acceleration limitations of the robot. The second stage maximizes an objective function based on progress towards the goal, clearance from obstacles and velocity. This approach demonstrated to be effective in avoiding collisions in the presence of obstacles.

Another optimization based controller is the model predictive control (MPC) which

was initially applied to chemical applications subject to many inputs and state constraints. Since then variations of the classic algorithm were created and applied to different fields including mobile robots. All of them are based on the same principle of optimizing a user defined cost function subject to constraints.

Brito et Al. [11] applied a variation of the nonlinear MPC called Local Model Predictive Contouring Control (LMPCC) able to control mobile robots in unstructured dynamic environments in the presence of both static and dynamic obstacles with better performance than DWA.

Vries et Al. [12] expanded on the work of Brito et Al. by creating a framework for control of autonomous surface vessels (ASV). In this work the cost function was modified to account for maneuvers compliant with the International Regulations for Preventing Collisions at Sea (COLREGs) in a so-called regulation aware MPCC (RA-MPCC). These regulations were created to standardize how vessels should behave to avoid collisions, some of the rules are shown on Fig. 2.1.

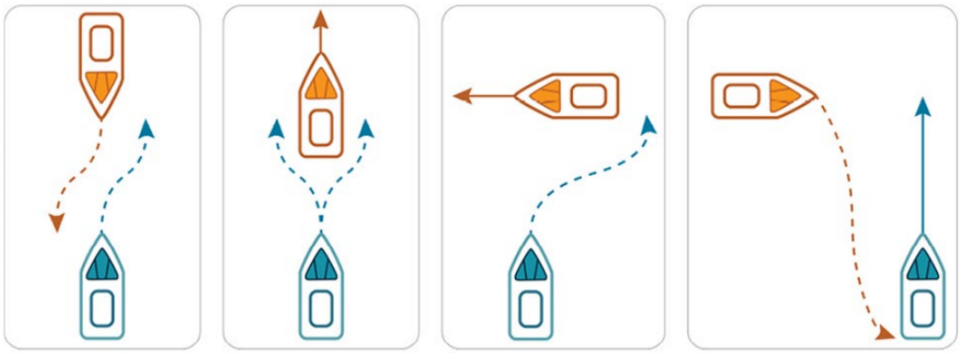

Figure 2.1: Some maneuvers compliant with COLREGs rules. Extracted from [2]

The RA-MPCC is able to perform well in terms of collision avoidance and trajectory tracking compliant with kinodynamic constraints. Moreover, it was shown that the number of COLREGs violations using this method was substantially lower than with LMPCC.

Chapter 3

Global Path Planning

Consider the configuration space with dimension d, $Q \subset R^d$. Let $Q_{obs} \subset Q$ denote the obstacle region and $Q_{free} = Q \setminus Q_{obs}$ denote the obstacle-free region. Finally, let $q_{start} \in Q_{free}$ denote the start state and $Q_{goal} \subset Q_{free}$ denote the goal region. The path planning problem is finding a set of waypoints $q_i \in Q_{free}$ that when connected through segments form a path from q_{start} to a state $q_{goal} \in Q_{goal}$. The resulting path is the sequence of segments connecting the waypoints.

This chapter will focus on global path planning and can be split into two parts. The first part is about the occupancy grid and how it is used by autonomous vehicles to perceive the obstacle and obstacle-free regions. The second part is about the RRT family of sampling-based path planning algorithms and path post-processing .

3.1 Occupancy Grid

The occupancy grid is a representation used in robotics to discretize the environment as a matrix of cells. The size of these cells can be fixed or adaptive with smaller cells closer to obstacles. Each cell can be occupied or free according to the presence or absence of obstacles and in some cases it is also possible to have a probabilistic cell which assumes a probability of it being occupied. The occupancy grid is used by robots and autonomous vehicles to perceive the environment and to plan feasible paths by avoiding occupied cells during path planning.

The occupancy grid can be given to the navigation system a priori or it can be generated online as the vehicle traverses the environment. The latter is possible due to SLAM, a technique that uses data from the vehicle's sensors to create the occupancy grid and localize the vehicle on the generated map.

The global path planning of autonomous electric boats on canals considers that a global occupancy grid with fixed sized binary cells is known a priori. The acquisition of such grid can be done by image processing of an aerial view of the environment, as shown on Fig. 3.1.

The processed image with water distinguishable from land can be used to create the occupancy grid. First, the image is converted to a matrix of pixels that is subsequently used to fill an occupancy matrix in which pixels of water are set as free cells and all other pixels are set as occupied. The occupancy matrix is combined with the distance between cells to define the occupancy grid show on Fig. 3.2.

The occupancy grid shows free cells for particle elements, that occupy only one cell at time. However, vehicles often occupy more than one cell and their dimensions must be

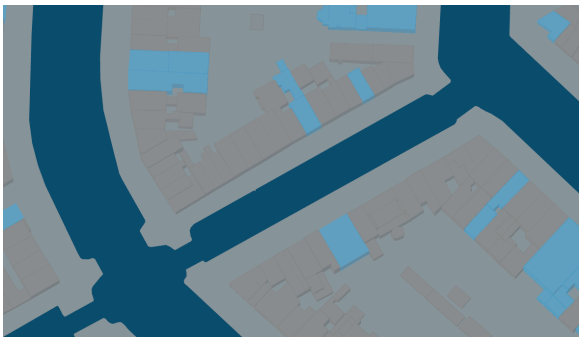

Figure 3.1: Aerial view of a canal section with distinction between water and land.

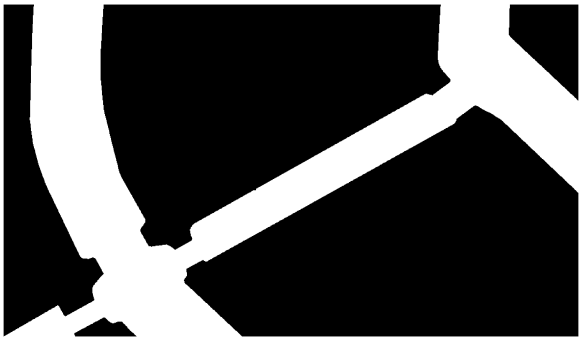

Figure 3.2: Binary occupancy grid of canal. Free cells are white and occupied cells are black.

considered during the path planning to avoid unfeasible paths. This can be achieved by inflating occupied cells by a radius related to the vehicle's dimensions as show on Fig. 3.3.

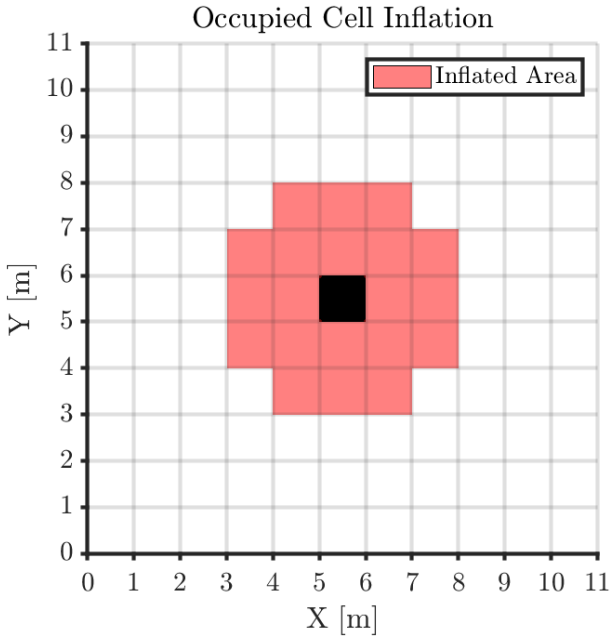

Figure 3.3: Example of a occupied cell inflation with radius of 2m.

The resulting occupancy grid after cell inflation is show on Fig. 3.4. In this grid the boat still occupies one cell but the free area accounts for the boat dimensions.

Path planning algorithms use the occupancy grid to verify if a state q belongs to Q_{free} or $Q_{obstacle}$, this is done by checking the cell state of the corresponding position of q. Moreover, it can also be used to check if a path segment is obstacle free by interpolating it as a set of states $q_i \in Q_{edge}$ and checking if all $q_i \in Q_{free}$.

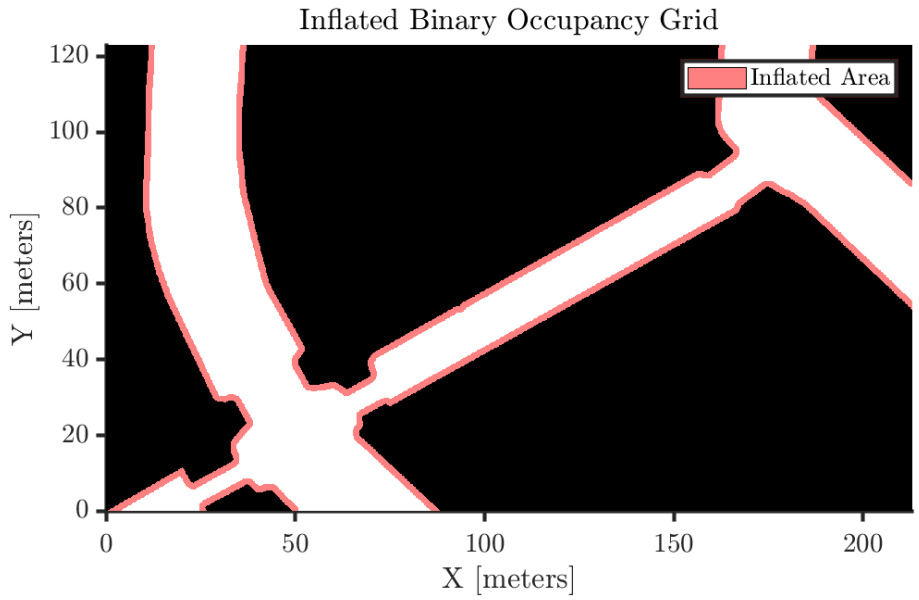

Figure 3.4: Binary occupancy grid of a canal after occupied cells are inflated

3.2 Rapidly-Exploring Random Trees (RRT)

Sampling-based path planning algorithms are able to quickly explore the state space when looking for feasible paths. They do this by sampling states $q \in Q$ and checking if they can be connected to states already included in the solution data structure.

RRT is a sampling-based algorithm introduced by LaValle [4]. The data structure used to represent the expansion of the path through the configuration space is a tree and RRT is suitable to be applied to holonomic, nonholonomic and kinodynamic constraints. Figure 3.5 exemplifies one iteration of the algorithm and its implementation is shown on Alg. 1.

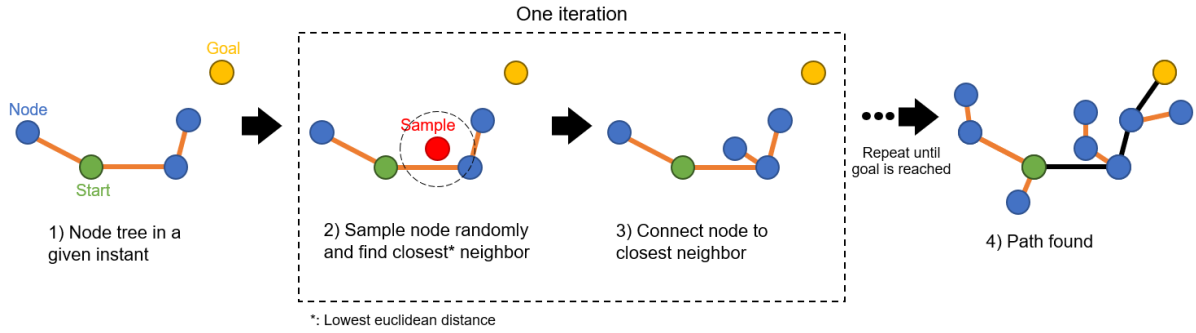

Figure 3.5: Diagram of a RRT iteration

Algorithm 1 Basic RRT

$V \leftarrow \{q_{init}\}$
$E \leftarrow \emptyset$
for i=1,...,n **do**
  $q_{rand} \leftarrow SampleFree$
  $q_{nearest} \leftarrow Nearest(G = (V, E), q_{rand})$
  $q_{new} \leftarrow Steer(q_{nearest}, q_{rand})$
  if $ObstacleFree(q_{nearest}, q_{new})$ **then**
    $V \leftarrow V \cup \{q_{new}\}$
    $E \leftarrow E \cup \{(q_{nearest}, q_{new})\}$
  end if
end for
return $G = (V, E)$ ▷ Tree data structure

RRT is an iterative method that creates a tree rooted at a start state and expands randomly through the configuration space. At each iteration a new node $q_{rand} \in Q_{free}$ is created and the nearest node to it, $q_{nearest}$, is found using some distance criterion such as euclidean distance. Subsequently, a steering function creates a new node q_{new} between q_{rand} and $q_{nearest}$, including the latter. If the edge that connects q_{new} to q_{near} belongs to Q_{free} then q_{new} and the edge are added to the vertex set V and to the edge set E respectively. Otherwise the algorithm goes to the next iteration. The algorithm ends when the number of iterations is reached, but it can also end if a feasible path is found.

3.3 RRT*

RRT* is a modification of the basic method proposed by Karaman and Frazzoli [5]. The main drawback of RRT is the lack of asymptotic optimality since no cost distinction is made between feasible paths. RRT* addresses this by introducing a cost criterion assigned to each path as well as the introduction of a rewiring technique for restructuring the solution tree. Under these modifications and with a straight steering function, RRT* is proven to be asymptotically optimal. Figure 3.6 exemplifies one iteration of the algorithm and its implementation is shown on Alg. 2.

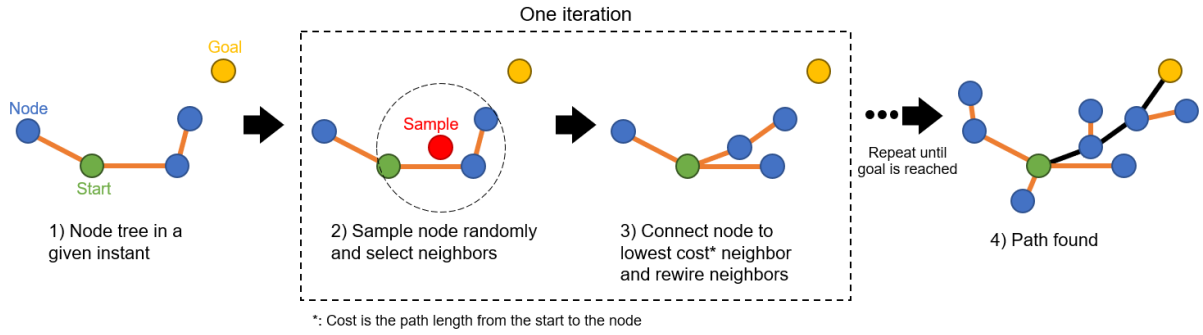

Figure 3.6: Diagram of a RRT* iteration

Algorithm 2 RRT*

$V \leftarrow \{q_{init}\}$
$E \leftarrow \emptyset$
for i=1,...,n **do**
  $q_{rand} \leftarrow SampleFree$
  $q_{nearest} \leftarrow Nearest(G = (V, E), q_{rand})$
  $q_{new} \leftarrow Steer(q_{nearest}, q_{rand})$
  if $ObstacleFree(q_{nearest}, q_{new})$ **then**
    $a \leftarrow Card(V)$
    $Q_{near} \leftarrow Near(G = (V, E), q_{new}, min\{\gamma_{RRT*}(log(a)/a)^{\frac{1}{d}}, \eta\})$
    $V \leftarrow V \cup \{q_{new}\}$
    $q_{min} \leftarrow q_{nearest}$
    $c_{min} \leftarrow Cost(q_{nearest}) + Cost(q_{nearest}, q_{new})$
    for each $q_{near} \in Q_{near}$ **do** ▷ Connect q_{new} to minimum cost path
      if $CollisionFree(q_{near}, q_{new})$ **and**
        $Cost(q_{near}) + Cost((q_{near}, q_{new})) < c_{min}$ **then**
        $q_{min} \leftarrow q_{near}$
        $c_{min} \leftarrow Cost(q_{near}) + Cost((q_{near}, q_{new}))$
      end if
    end for
    $E \leftarrow E \cup \{(q_{min}, q_{new})\}$
    for each $q_{near} \in Q_{near}$ **do** ▷ Rewire tree
      if $CollisionFree(q_{near}, q_{new})$ **and**
        $Cost(q_{new}) + Cost((q_{near}, q_{new})) < Cost(q_{near})$ **then**
        $q_{parent} \leftarrow Parent(q_{near})$
        $E \leftarrow (E \setminus \{(q_{parent}, q_{near})\}) \cup \{(q_{near}, q_{new})\}$
      end if
    end for
  end if
end for
return $G = (V, E)$ ▷ Tree datastruct

RRT* has the same structure as RRT until the steering function and acquisition of node q_{new}. After q_{new} is assigned, all nodes within a radius are included in Q_{near} and q_{new} is connected to $q_{near} \in Q_{near}$ that generates the path with lowest cost. Subsequently, all other nodes inside Q_{near} are checked to see if changing their parent to q_{new} results in a path with lower cost than their current cost. If the new cost is lower than the current cost, the rewiring loop changes the parent of q_{near} to q_{new} and substitutes the edge (q_{parent}, q_{near}) with (q_{new}, q_{near}).

The creation of Q_{near} requires the implementation of the Near function that finds all nodes within a radius of q_{new}. This radius is the minimum between a variable radius $r(card(V))$ and a fixed radius η. The variable radius is defined as:

$$r(card(V)) = \gamma_{RRT*} \left(\frac{\log(card(V))}{card(V)} \right)^{\frac{1}{d}} \tag{3.1}$$

Where γ_{RRT*} is a constant that depends on Q_{free} and Q, d is the dimension of the configuration space Q and $card(V)$ is the cardinality of the Vertex set. From this equation,

as the number of nodes inside V increases, the radius decreases. Some implementations set γ_{RRT*} to a fixed value dismissing calculations.

3.4 Dubins curves and RRT* modification

Consider a configuration space Q in which $q_i = (x_i, y_i, \theta_i)$. In this case the basic steering function, that follows a straight line between two points, would result in a C^0 path with tangential discontinuity [13]. In this case, there is a discontinuity in the angle between two sequential path segments, this type of path is not suitable for vehicles because it requires the vehicle to slow down to change direction and there are cases where kinematic constraints make such path unfeasible.

A solution for this problem is to apply path smoothing techniques, that receive a set of waypoints and returns a path that is at least C^1. Some of the techniques commonly used are polynomial interpolation [1], Bézier curves [14] and Dubins curves [6]. In this thesis, Dubins curves will be used for path smoothing.

The Dubins curves, also referred to as Dubins path, are used to connect oriented points in space when subject to forward only motion and a maximum curvature constraint. This kind of curve is formed by a combination of circular arcs C = {"L": left, "R": right} and straight lines "S" tangent to the arcs. The possible combinations are of the form CCC = {LRL, RLR} and CSC = {LSL, LSR, RSL, RSR} as shown on Fig. 3.7. In addition to having a restricted curvature, a segment between two points formed by Dubins curves has the shortest length possible given the constraints [13].

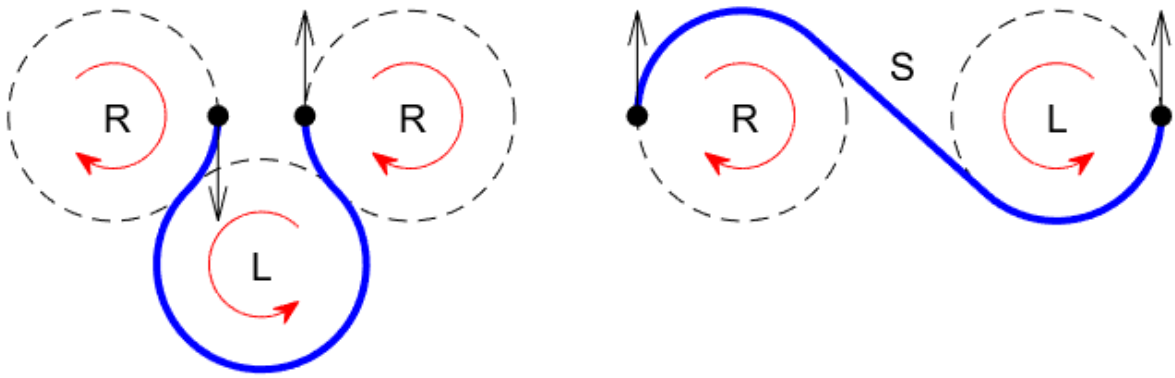

Figure 3.7: The two types of Dubins curves: CCC (left) and CSC (right).

In order to consider curvature constraints with RRT* path planning, it is possible to use a steering function based on Dubins Vehicles [6]. In this case, the edge (q_{near}, q_{new}) is a Dubins curve and the cost is the length of such curve.

The Dubins steering function works by connecting $q_{nearest}$ to q_{rand} with all types of Dubins curves and then selecting the lowest cost curve. In this case q_{new} coincides with q_{rand} while in the straight steering q_{new} could be an intermediate point between q_{rand} and q_{near}.

3.5 Path post-processing

The Dubins curve modification of RRT* is near optimal instead of optimal [6][7], this is caused by the Dubins steering function which is the same as a constant control input connecting two states. The fact that the path is not optimal is evident by the presence of unnecessary waypoints that contribute negatively to the overall path length. This problem caused by extra waypoints was approached in [1] and [15] by applying post-processing pruning.

In this section, an implementation of post-processing waypoint pruning will be introduced. The algorithm receives as input the path to be processed and it returns as output a path that has the same or smaller total length.

Algorithm 3 Dubins Path Post-Processing Pruning - Forward Pass

$W_{old} \leftarrow Interpolate(W_{old})$
$W_{new} \leftarrow \{W_{old}[1]\}$ ▷ W_{new} has the same start as W_{old}
$N \leftarrow Card(W_{old})$
$i \leftarrow 1$
while $i < N$ **do**
  $w_i \leftarrow W_{old}[i]$
  $j \leftarrow N$
  while $j > i + 1$ **do**
    $w_j \leftarrow W_{old}[j]$
    if $ObstacleFree(DubinsPath(w_i, w_j))$ **then**
      break ▷ Stop inner loop if obstacle-free segment is found
    end if
    $j \leftarrow j - 1$
  end while
  $W_{new} \leftarrow W_{new} \cup \{W_{old}[j]\}$
  $i \leftarrow j$
end while
$W_{new} \leftarrow Interpolate(W_{new})$

Alg. 3 is the pseudocode for pruning a path formed by Dubins curves. The method receives as input the set of waypoints W_{old} that define a path composed of Dubins curves and interpolates the path to generate more waypoints. Then a nested while loop iterates through these waypoints to find the longest obstacle-free Dubins curves between waypoints w_i and w_j. The outer loop increments i from the start of the path ($i = 1$) to the end ($i = N = card(W_{old})$) while the inner loop decrements j from N to $i + 1$. At each inner loop iteration the Dubins curve from w_i to w_j is tested. If it is obstacle free then w_j is added to W_{new} and the outer loop iterates again with $i = j$. After the entire original path is traversed ($i = N$), the new path with waypoints W_{new} can be interpolated. This implementation is called Forward Pass since the outer loop index i moves from 1 to N.

The $Interpolate(W)$ function generates a set of waypoints that belong to the Dubins segment connecting the endpoints. The $Card(W)$ function returns the number of waypoints inside a path. The $DubinsPath(w_i, w_j)$ function connects two points using Dubins curves and returns the set of waypoints of the segment. The $ObstacleFree(W)$ function checks if a segment collides with any obstacle.

The resulting post-processed path is composed of Dubins segments connecting waypoints

from the interpolation of the input path. Two examples of the effectiveness of the algorithm are shown on Fig. (3.8, 3.9). The first figure shows a case where a single Dubins segment is found between the start and end positions. The second figure shows a case where the output path is formed by less Dubins segments than the original path, albeit longer, and it can be observed that unnecessary curves from the input path disappeared.

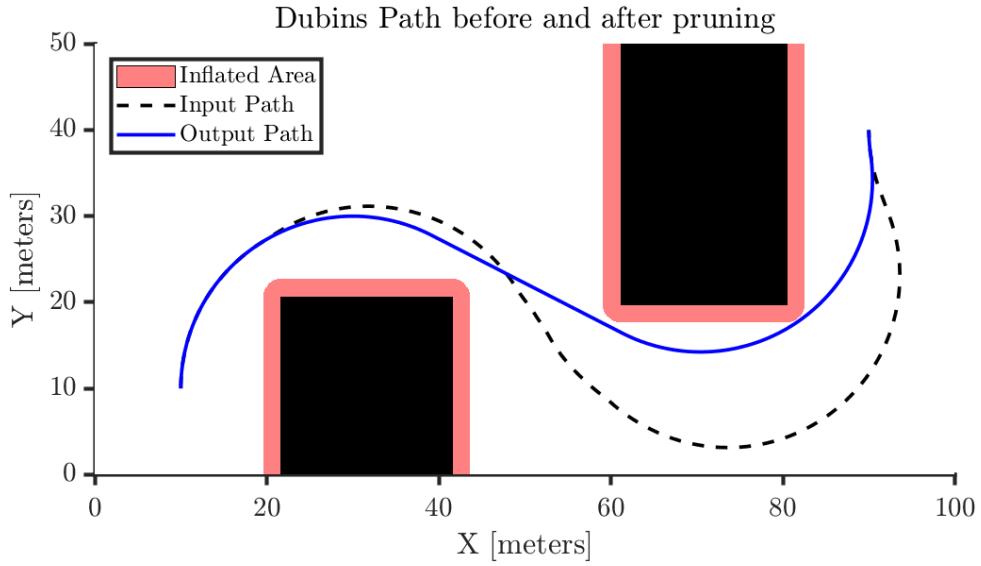

Figure 3.8: Post-Processing Pruning where a single segment is found from start to finish

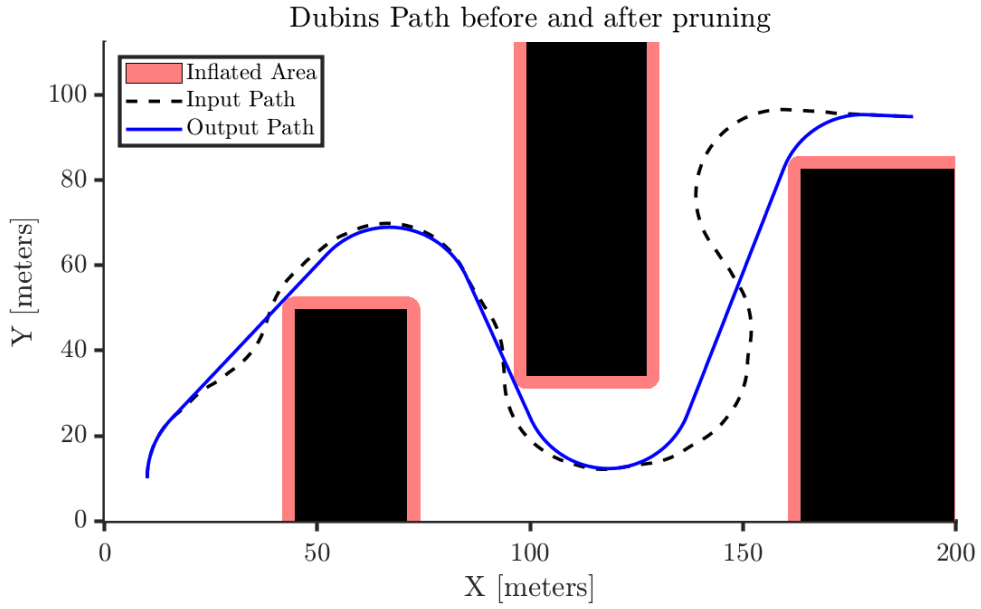

Figure 3.9: Post-Processing pruning with Forward Pass

As explained, the Alg. 3 is called Forward Pass since the outer loop is moves towards the end of the path. However, it is possible to modify this algorithm so i decrements from N to 1 and the Dubins curves formed start in w_j and end in w_i, the opposite of the original implementation. This modification is shown on Alg. 4 and is called Backward Pass since the outer loop index i moves from N to 1.

The backward pass modification can be useful when applied in series with the forward algorithm. This can be done by using the intermediate path, that is the output of one the

Algorithm 4 Dubins Path Post-Processing Pruning - Backward Pass

$W_{old} \leftarrow Interpolate(W_{old})$
$W_{new} \leftarrow \{W_{old}[1]\}$ ▷ W_{new} has the same start as W_{old}
$N \leftarrow Card(W_{old})$
$i \leftarrow N$
while $i > 1$ **do**
  $w_i \leftarrow W_{old}[i]$
  $j \leftarrow 1$
  while $j < N - 1$ **do**
    $w_j \leftarrow W_{old}[j]$
    if $ObstacleFree(DubinsPath(w_j, w_i))$ **then**
      break ▷ Stop inner loop if obstacle-free segment is found
    end if
    $j \leftarrow j + 1$
  end while
  $W_{new} \leftarrow W_{new} \cup \{W_{old}[j]\}$
  $i \leftarrow j$
end while
$W_{new} \leftarrow Interpolate(W_{new})$

algorithms, as the input path of the other algorithm. Although an improvement over the intermediate path is not guaranteed, it may happen as shown on Fig. 3.10.

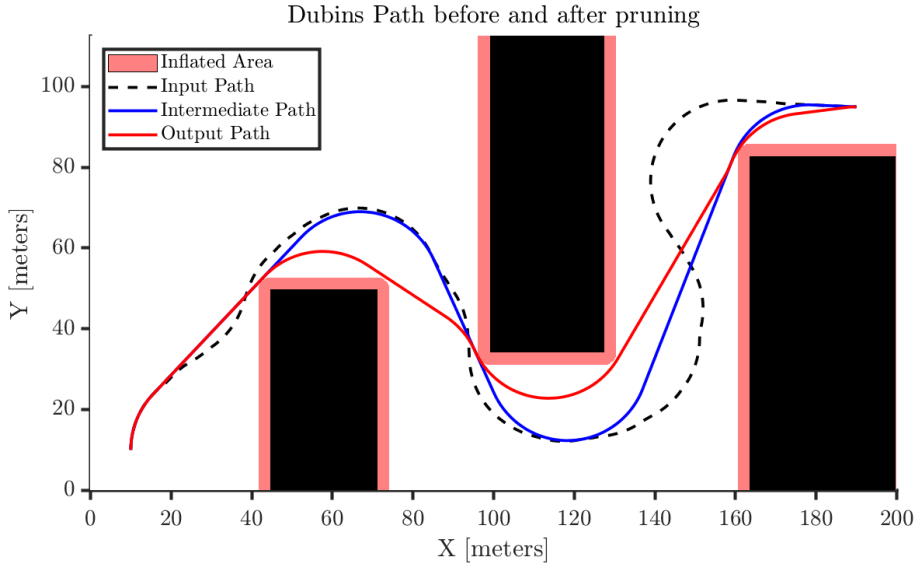

Figure 3.10: Post-Processing pruning where output of Forward Pass Fig. 3.9 is used as input of Backward Pass.

Chapter 4

Trajectory Planning

Trajectory planning refers to the process of binding a time law to a path. Moreover, it also includes dynamic constraints such as acceleration and velocity limitations [2]. In this chapter, a reference trajectory will be derived from the global path reference and a reference velocity profile.

The velocity profiles that will be introduced are the trapezoidal and smooth trapezoidal profiles. In these profiles the acceleration and deceleration occur only at the start and end of the trajectory, while the rest of the trajectory is kept at a constant cruise velocity.

This is useful because the Dubins curves used during path planning assume a vehicle moving at constant speed, which would be the cruise velocity. Moreover, some criteria used to measure comfort in autonomous vehicles are acceleration and jerk in the longitudinal and lateral directions [16]. Taking these in consideration, it is possible to create velocity profiles that stay within comfortable limits for passengers.

4.1 Trapezoidal Velocity Profile

The trapezoidal velocity profile refers to a velocity profile divided in three sections: ramp-up slope, cruise velocity and ramp-down slope. In this profile, the ramp-up and ramp-down slopes follow linear polynomials for the velocity $v(t) = b_0 + b_1 t$. The profile is shown on Fig. 4.1

The following functions (4.1-4.4) give the distance travelled along the path $s(t)$, the longitudinal velocity $v(t)$ and the longitudinal acceleration $a(t)$ for the entire trajectory as well as the definition of t_i.

$$s(t) = \begin{cases} \frac{1}{2} a_{max} t_1^2 \ , \ t \in [0, T_1] \\ S_1 + V_c t_2 \ , \ t \in [T_1, T_1 + T_2] \\ S_1 + S_2 + V_c t_3 + \frac{1}{2} a_{min} t_3^2 \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.1}$$

$$v(t) = \begin{cases} a_{max} t_1 \ , \ t \in [0, T_1] \\ V_c \ , \ t \in [T_1, T_1 + T_2] \\ V_c + a_{min} t_3 \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.2}$$

$$a(t) = \begin{cases} a_{max} \ , \ t \in [0, T_1] \\ 0 \ , \ t \in [T_1, T_1 + T_2] \\ a_{min} \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.3}$$

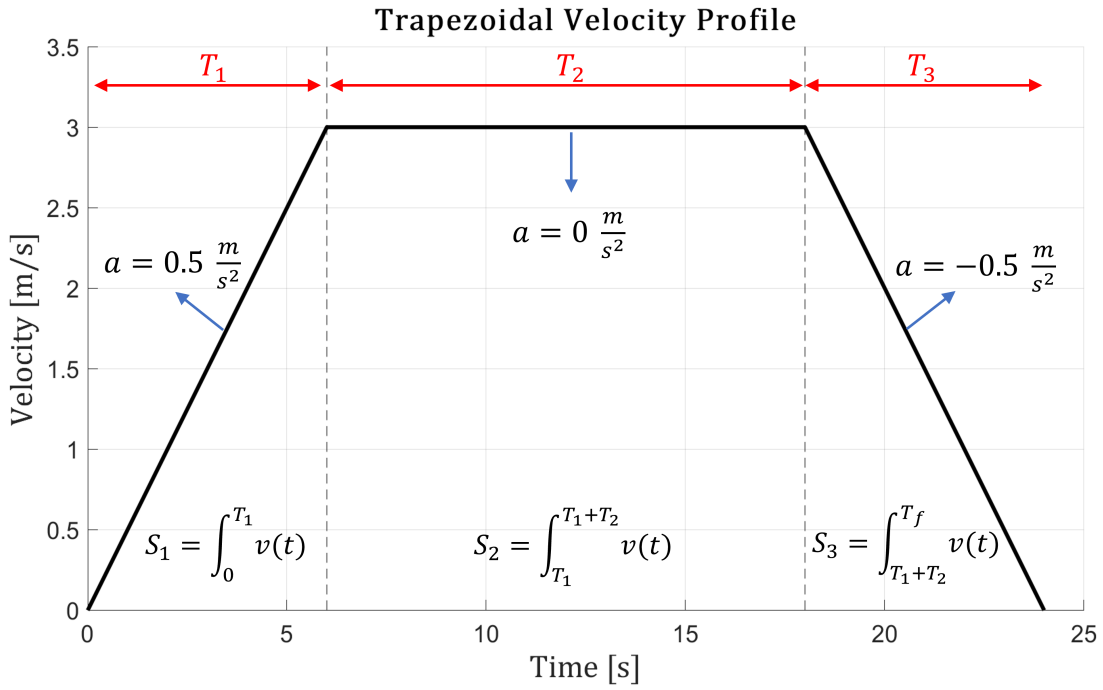

Figure 4.1: Velocity progression for a Trapezoidal Velocity Profile. Adapted from [17]

$$t_i = \begin{cases} t\ ,\ i = 1 \\ t - T_1\ ,\ i = 2 \\ t - T_1 - T_2\ ,\ i = 3 \end{cases} \tag{4.4}$$

These functions are obtained by considering the boundary conditions given by maximum acceleration $a_{max} > 0$, minimum acceleration $a_{min} < 0$, cruise velocity V_c, initial velocity $V_0 = 0$ and final velocity $V_f = 0$. The constants T_i and S_i are the time taken and the length of each section of the path, where $i = 1,2,3$ refers to ramp-up, cruise and ramp-down sections respectively. These constants can be found by considering the total time for traversing the path T_f and the total length $s(T_f) = S_f$ in the following equations (4.5-4.11):

$$T_1 = \frac{V_c}{a_{max}} \tag{4.5}$$

$$S_1 = \frac{V_c^2}{2a_{max}} \tag{4.6}$$

$$T_3 = \frac{-V_c}{a_{min}} \tag{4.7}$$

$$S_3 = -\frac{V_c^2}{2a_{min}} \tag{4.8}$$

$$S_2 = S_f - S_1 - S_3 \tag{4.9}$$

$$T_2 = \frac{S_2}{V_c} \tag{4.10}$$

$$T_f = T_1 + T_2 + T_3 \tag{4.11}$$

The drawback of using a trapezoidal velocity profile is the presence of discontinuities in the acceleration function $a(t)$ (4.3). These are perceived as impulses in the vehicle's jerk and can be uncomfortable for passengers.

4.2 Smooth Trapezoidal Velocity Profile

The smooth trapezoidal profile considered in [17] is a modification of the trapezoidal profile. This profile parameterizes the ramp-up and ramp-down velocities as cubic polynomials

$v(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$. Consequently, the acceleration is a quadratic polynomial with no impulse in the jerk. The profile is shown on Fig. 4.2.

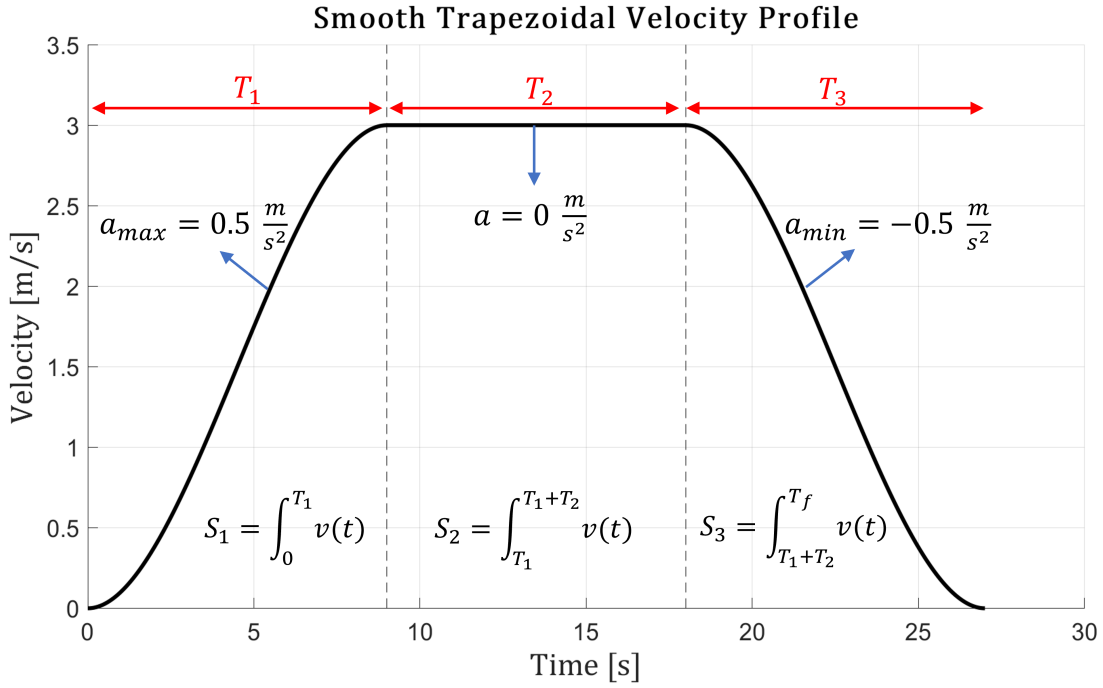

Figure 4.2: Velocity progression for a Smooth Trapezoidal Velocity Profile. Adapted from [17]

The following functions (4.12-4.14) give the distance travelled along the path $s(t)$, the longitudinal velocity $v(t)$ and the longitudinal acceleration $a(t)$ for the entire trajectory.

$$s(t) = \begin{cases} b_{u_0} t_1 + \frac{1}{2} b_{u_1} t_1^2 + \frac{1}{3} b_{u_2} t_1^3 + \frac{1}{4} b_{u_3} t_1^4 \ , \ t \in [0, T_1] \\ S_1 + V_c t_2 \ , \ t \in [T_1, T_1 + T_2] \\ S_1 + S_2 + b_{d_0} t_3 + \frac{1}{2} b_{d_1} t_3^2 + \frac{1}{3} b_{d_2} t_3^3 + \frac{1}{4} b_{d_3} t_3^4 \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.12}$$

$$v(t) = \begin{cases} b_{u_0} + b_{u_1} t_1 + b_{u_2} t_1^2 + b_{u_3} t_1^3 \ , \ t \in [0, T_1] \\ V_c \ , \ t \in [T_1, T_1 + T_2] \\ b_{d_0} + b_{d_1} t_3 + b_{d_2} t_3^2 + b_{d_3} t_3^3 \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.13}$$

$$a(t) = \begin{cases} b_{u_1} + 2 b_{u_2} t_1 + 3 b_{u_3} t_1^2 \ , \ t \in [0, T_1] \\ 0 \ , \ t \in [T_1, T_2] \\ b_{d_1} + 2 b_{d_2} t_1 + 3 b_{d_3} t_3^2 \ , \ t \in [T_1 + T_2, T_f] \end{cases} \tag{4.14}$$

$$t_i = \begin{cases} t \ , \ i = 1 \\ t - T_1 \ , \ i = 2 \\ t - T_1 - T_2 \ , \ i = 3 \end{cases}$$

These functions are obtained by considering the boundary conditions given by maximum acceleration $a_{max} > 0$, minimum acceleration $a_{min} < 0$, cruise velocity V_c, initial velocity $V_0 = 0$ and final velocity $V_f = 0$. The constants T_i, S_i and their subscripts are defined in

the same way as the trapezoidal velocity profile. The total path length is S_f and the total time is T_f.

A system of nonlinear equations must be solved to obtain b_0, b_1, b_2, b_3 and T_i for i = 1,3. The following derivation is for the ramp-up (section 1), but the procedure is the same for the ramp-down. The subscript u indicates that the variable is related to the ramp-up slope, for a ramp-down formulation the subscript would be d.

$$v_u(t) = b_{u_0} + b_{u_1}t + b_{u_2}t^2 + b_{u_3}t^3 \tag{4.15}$$

$$a_u(t) = b_{u_1} + 2b_{u_2}t + 3b_{u_3}t^2 \tag{4.16}$$

These are subject to the following boundary conditions:

$$v(0) = V_0 \tag{4.17} \qquad v(T_1) = V_c \tag{4.18}$$

$$a(0) = 0 \tag{4.19} \qquad a(T_1) = 0 \tag{4.20}$$

An additional condition is necessary to guarantee that a_{max} is reached and not exceed. It can be obtained by enforcing that a_{max} occurs at the inflection point of $a(t)$:

$$\frac{da(t)}{dt} = j(t) = 2b_{u_2} + 6b_{u_3}t$$
$$j(T_{in}) = 2b_{u_2} + 6b_{u_3}T_{in} = 0$$
$$T_{in} = \frac{-b_{u_2}}{3b_{u_3}}$$
$$a(T_{in}) = b_{u_1} - \frac{b_{u_2}^2}{3b_{u_3}} = a_{max} \tag{4.21}$$

Substituting the boundary conditions (4.17-4.21) in (4.15-4.16) results in a total of 5 equations. The complete system (4.22) is solved for b_{u_0}, b_{u_1}, b_{u_2}, b_{u_3} and T_1.

$$\begin{cases} b_{u_0} - V_0 = 0 \\ b_{u_0} + b_{u_1}T_1 + b_{u_2}T_1^2 + b_{u_3}T_1^3 - V_c = 0 \\ b_{u_1} = 0 \\ b_{u_1} + 2b_{u_2}T_1 + 3b_{u_3}T_1^2 = 0 \\ 3b_{u_1}b_{u_3} + b_{u_2}^2 + 3b_{u_3}a_{max} = 0 \end{cases} \tag{4.22}$$

This system of nonlinear equations can be simplified by considering that b_{u_1} will always be zero (4.19), reducing the total number of variables to 4. Subsequently it can be solved in MATLAB using fsolve, fmincon or any other nonlinear system solver.

After the system is solved for the ramp-up and ramp-down, T_1 and T_3 will be defined and S_1 and S_3 can be obtained by integration of $v(t)$. The last step is to calculate $S_2 = S_f - S_1 - S_3$, $T_2 = S_2 \div V_c$ and $T_f = T_1 + T_2 + T_3$. At this point the functions (4.12-4.14) are ready to be used.

The resulting smooth trapezoidal profile has low magnitudes of jerk and a continuous acceleration lower than a_{max} for all points except at the inflection. Both of these properties increase comfort for passengers.

The drawback of using a smooth trapezoidal velocity profile over the trapezoidal profile is that for the same acceleration limits the time required to traverse the path is bigger because the cruise velocity is reached slower.

4.3 Reference Trajectory Generation

Given constraints on velocity and acceleration, a velocity profile from the previous sections is created. The next step is to fit this velocity profile to the reference path given by the global path planner.

Since the velocity profile progression gives the distance travelled along the path, it is necessary to convert the path from Cartesian coordinates to Frenet coordinates shown on Fig. 4.3. This coordinate system is based on a reference frame whose origin O_F moves together with the vehicle along the path and the state of the vehicle is given by $\{s, \dot{s}, \ddot{s}, l, \dot{l}, \ddot{l}\}$, where s is the length traversed on path from its start and l is the lateral deviation perpendicular to the direction of the path.

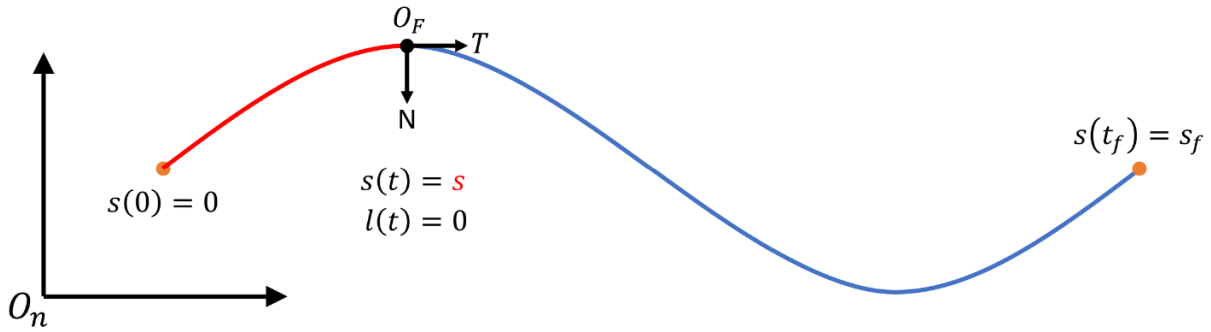

Figure 4.3: Cartesian and Frenet reference frames.

The reference trajectory is generated by sampling s_i, v_i and a_i derived from the velocity profile with a sufficiently small time step, i is the index of the discretized reference trajectory. Each sample $z_{profile,i} = (s_i, v_i, a_i)$ can be represented in the Frenet frame, hence the sample becomes $z_{Frenet,i} = (s_i, v_i, a_i, 0, 0, 0)$, since the desired lateral deviation and its derivatives are zero.

The last step is to convert the reference trajectory from Frenet coordinates to Cartesian coordinates because this is the coordinate system used by the local motion planner to control the boat. The resulting discretized trajectory is made of samples $z_i^{ref} = (x_i, y_i, \psi_i, v_{x,i})$, where the the pose is given in the inertial coordinate system and $v_{x,i} = v_i$ is the longitudinal velocity of the boat at time step i.

Chapter 5

Modelling

This chapter aims to introduce the model that will be used for simulation and control of autonomous electric boats. The theory about ship dynamics as well as a variety of models, for underwater and surfaces vehicles, have been presented by Fossen in his book "Handbook of marine craft hydrodynamics and motion control" [18].

More specifically, relevant references frames and the nonlinear differential equation of the boat will be introduced. In addition, a representation used for autonomous vehicles will be applied to the boat by dividing it in a set of discs.

5.1 Reference Frames

The pose and velocities of ships can be represented in different standard reference frames: Earth-Centered reference frames and Geographic reference frames. The former has its origin fixed in the center of the Earth and is mainly used in applications where ships travel long distances such as intercontinental routes. The latter includes two references frames that are going to be used in the following sections for the boat modelling.

NED : The North-East-Down coordinate system (NED) has its global position defined by latitude and longitude. This reference frame is tangent to Earth's surface and its axes $\{\mathrm{n}\} = (x_n, y_n, z_n)$ point to the True North, East and Down to the center of Earth respectively. The origin O_n is chosen in the local reference in which the ship operates. This reference frame is not inertial, however it can be assumed as a inertial global reference frame when dealing with navigation in a local area.

BODY: This is a body-fixed coordinate system with axes $\{\mathrm{b}\} = (x_b, y_b, z_b)$ and origin O_b is chosen as some point in the boat. The axes are chosen as:

- x_b: longitudinal axis, pointing to bow (front of the boat).
- y_b: transversal axis, pointing to starboard (the right side of the boat).
- z_b: normal axis, pointing down.

Since trajectory tracking does not consider vertical translation in Z, this axis will not be needed for further studies. Consequently, only a 2D representation of reference frames will be used. Furthermore, rotations about x_b and y_b will be removed because they are considered negligible. The resulting NED and BODY frames are shown on Fig. 5.1.

The pose of a boat (5.1), η, is the configuration of its position and attitude in NED coordinates. The vector of linear and angular velocities (5.2), ν, and the vector of forces and moments (5.3), τ, are considered in the BODY frame.

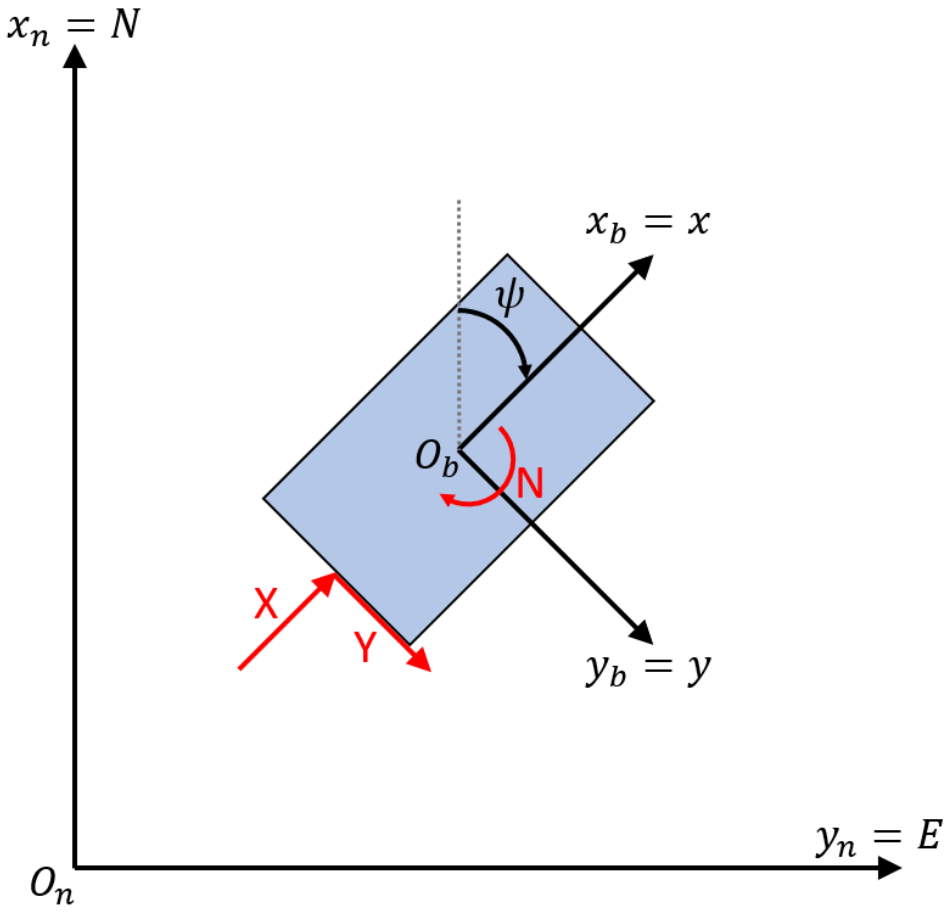

Figure 5.1: NED and BODY reference frames used for boat modelling. Red arrows are forces and moments.

$$\eta = \begin{bmatrix} N \\ E \\ \psi \end{bmatrix} \quad (5.1) \qquad \nu = \begin{bmatrix} v_x \\ v_y \\ r \end{bmatrix} \quad (5.2) \qquad \tau = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} \quad (5.3)$$

The angle ψ in η is the yaw angle between NED and BODY. This angle is used to define the rotation matrix R_b^n between NED and BODY that can be used to convert the coordinates of vectors in BODY to NED. Moreover, a property of rotation matrices is their orthogonality, consequently $(R_b^n)^{-1} = (R_b^n)^T$ and the transpose of R_b^n can be used to convert vectors from NED to BODY.

5.2 Dynamic Model

The simplifications done in the last section lead to a 3 DOF model considering surge-sway-yaw. The pose, velocity and force vectors are defined in (5.1-5.3). The dynamics of the boat follow the nonlinear differential equation:

$$\dot{\eta} = R_b^n(\eta)\nu \quad (5.4)$$

$$M\dot{\nu_r} + C(\nu_r)\nu_r + D(\nu_r)\nu_r = \tau + \tau_{wind} + \tau_{wave} \quad (5.5)$$

Where R_b^n is given by equation 5.6, ν_r is the relative velocity between the boat ν and currents ν_c, M is the mass matrix including the boat mass and added masses due to hydrodynamics, C is the Coriolis and centripetal matrix including the boat and hydrodynamic coefficients, D is the matrix of hydrodynamic damping.

$$R_b^n(\eta) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

The model (5.5) can be simplified by making some assumptions. The first assumption is that the current velocity ν_c is considered to be low so that $\nu_r = \nu - \nu_c = \nu$. The second assumption is that forces and moments τ_{wind} and τ_{wave} can be disregarded since their effect will be compensated by the controller. The simplified dynamics is given by Eq. 5.7.

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu = \tau \tag{5.7}$$

The boat's dynamics can be represented in nonlinear state space form (5.8) with state variable $x \in \mathbb{R}^6$,where $x = [\eta^T, \nu^T]^T = [N, E, \psi, v_x, v_y, r]^T$. The vector τ depends on input variables u according to the type of actuators used and how they are positioned in the boat.

$$\dot{x} = f(x,u) = \begin{bmatrix} R_b^n(\eta)\nu \\ M^{-1}(\tau(u) - C(\nu)\nu - D(\nu)\nu) \end{bmatrix} \tag{5.8}$$

5.3 Boat Representation

It is essential that the autonomous electric boat is able to maneuver without colliding with obstacles when following the reference trajectory. Hence, the control unit should be aware of the dimensions of the boat. The technique chosen is often used for robots and autonomous vehicles and consists in dividing the geometry of the vehicle in multiple discs [11]. This is shown on Fig. 5.2.

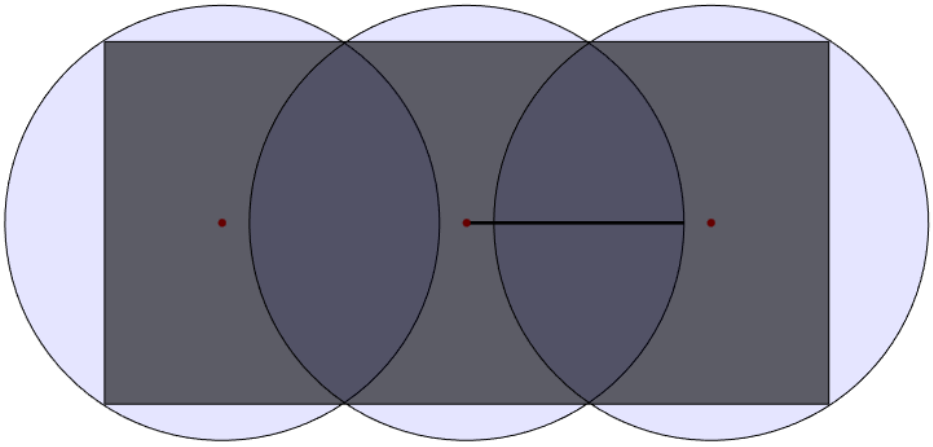

Figure 5.2: Rectangular boat represented as a set of discs. Red dots are the discs centers and the black line points from the center of the boat to its bow

Each disc i is defined by its radius r_{disc} and its position in the BODY frame $p_{d,b}^i$, that is given by equation (5.9).

$$p^i_{d,b} = \begin{bmatrix} x^i_d \\ y^i_d \end{bmatrix} \tag{5.9}$$

The disc's position in NED, $p^i_{d,n}$, is given by equation (5.10). In this equation, subscript c indicates the origin of BODY frame, that will be considered as the center of the boat. Subscript d refers to disc and the index i is the index of the disc.

$$p^i_{d,n} = \begin{bmatrix} N^i_d \\ E^i_d \end{bmatrix} = \begin{bmatrix} N_c \\ E_c \end{bmatrix} + R^n_b(\psi) p^i_{d,b} \tag{5.10}$$

Collisions can be avoided by imposing that all of the discs do not intersect with obstacles. That is to say, the distance between obstacles and the center of each disc must be higher than the radius of the disc.

Chapter 6

NMPC Local Motion Planner

A definition of motion planning in control theory is the construction of inputs to a dynamical system that drives it to a specified goal state [19]. By this definition, a local motion planner should receive a reference trajectory (goal state) from chapter 4 and guide the dynamical system from chapter 5 to it. Furthermore, it should include updated information of the environment obtained through sensors to avoid collisions. It would also be useful if the motion planner considered limitations on a vessel's control inputs during the operation.

In this chapter a Nonlinear Model Predictive Control (NMPC) local motion planner will be developed since it is able to satisfy all of the stated requirements. For this reason the MPC theory will be introduced along with formulations specific to NMPC and relevant constraints for the operation of autonomous boats.

6.1 MPC Theory

Model Predictive Control and its variants are a family of optimization-based methods for feedback control [20]. All of them follow an implementation based on finding a sequence of control inputs that minimizes a cost function under constraints over a finite horizon also called prediction horizon. Some definitions in the context of MPC are explained below, consider $\boldsymbol{t_k}$ the current time instant.

Prediction Horizon (T_p) is the time window in which the MPC predicts future states of the controlled system. In other words, the system states are predicted over $[t_k, t_k + T_p]$.

Control Horizon (T_c) is the time window in which the MPC can manipulate control inputs, after this window control inputs remain constant. In other words, the control input changes over $[t_k, t_k + T_c]$ and remains constant over $[t_k + T_c, t_k + T_p]$.

Cost Function is the objective function to be minimized. The cost function is flexible and can be tuned for each control problem. In general, there are positive cost terms for each time step of the prediction horizon and they are computed from states and control inputs.

The Fig. 6.1 shows a MPC iteration at time step K, in this depiction the horizons are discretized such that $T_p = N_p$ and $T_c = N_c$.

The predictions made by MPC algorithms use differential equations describing the dynamics of the controlled system and all of the predictions are made considering an initial

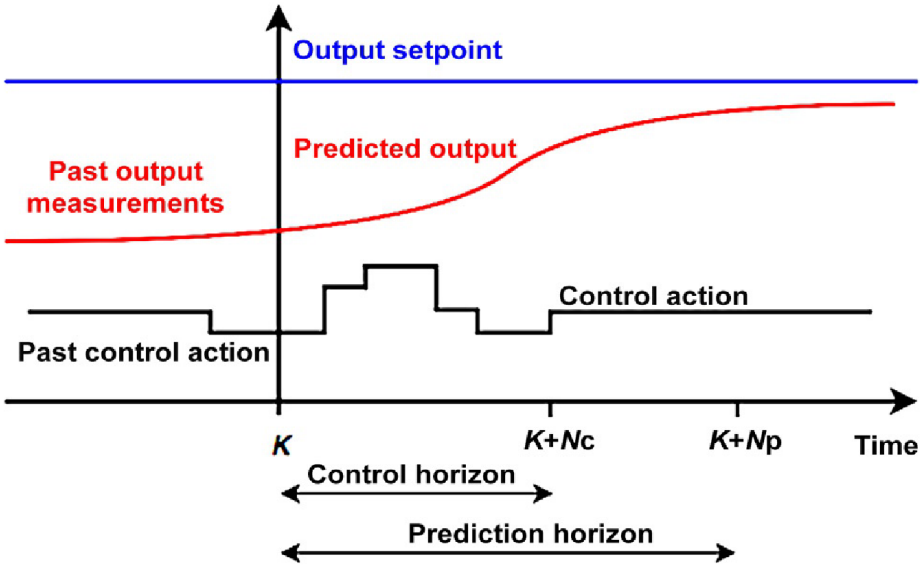

Figure 6.1: Illustration of NMPC at time step K. Adapted from [21]

state x_0 and a sequence of control inputs. This approach appears to be an open-loop control, since there is no correction of the states during the prediction horizon. However, MPC is considered to be closed-loop due to the **receding horizon control principle** [22]:

1. At time k and state x_k, solve the optimization problem for the prediction horizon $[k, k + N_p]$.

2. Apply the first control input of the solution, u_0^*, to the system.

3. Update the current time to $k + 1$, measure the current state x_{k+1} and go back to step 1. Feedback of states occurs at this step.

MPC approaches are able to handle control problems where offline computation of a control law is hard or impossible in case there is no analytical solution for the control problem. Furthermore, MPC can account for hard constraints on states and inputs by design. In particular, NMPC stands out since it can cope with nonlinear environmental constraints [23].

6.2 Optimal Control Problem

The difference between a NMPC and a linear MPC is the type of ODE describing the system dynamics. As a result, in the case of linear MPC, it is possible to find an analytical solution when no constraints are present. Whereas only numerical solutions can be found for NMPC.

The numerical solution is given by iterative optimization over the prediction horizon. The optimization problem to be solved at each iteration is known as optimal control problem (OCP) and it has the form (6.1) [24].

$$
\begin{aligned}
\min_{x(.),u(.)} \quad & \int_0^{T_p} L(x(t),u(t))\,dt + E(x(T_p)) \\
\text{s.t.} \quad & \dot{x} = f(x(t),u(t)),\ t \in [0,T_p] \quad \text{(ODE model)} \\
& h(x(t),u(t)) \leq 0,\ t \in [0,T_p] \quad \text{(Path constraints)} \\
& x(0) = x_0 \qquad \text{(Fixed initial state)} \\
& u(t) = u(T_c),\ t \in [T_c,T_p] \quad \text{(Control Horizon)}
\end{aligned}
\tag{6.1}
$$

Where the objective function has a cost $L(x(t),u(t))$ for intermediate states and a cost $E(x(T_p))$ for the final state of the horizon. This OCP can be solved efficiently by direct approaches based on a "First discretize then optimize" principle, that convert it to a non-linear program (NLP).

One direct approach is the single shooting. In this approach the optimization variable is the set of inputs $\boldsymbol{u}$ over the prediction horizon and the states are computed by forward simulation (integration) using $\boldsymbol{u}$ and the initial state $\boldsymbol{x_0}$. The disadvantage of this method is that for nonlinear ODEs the numerical integration errors increase with the integration interval.

Multiple shooting is a direct method introduced by Bock [25]. This method mitigates the downsides of single shooting and the core of this approach is to divide the prediction horizon into segments and integrate over each segment separately, this is useful because for smaller segments the numerical errors are lower. Moreover, an augmented optimization variable is used $\boldsymbol{w}$ that includes both control inputs $\boldsymbol{u}$ and node states $\boldsymbol{s}$. The algorithm is such that the node states $\boldsymbol{s}$ converge to the predicted states $\boldsymbol{x}$, hence constraints on the state $\boldsymbol{x}$ can be satisfied by imposing them to $\boldsymbol{s}$ instead. The Fig. 6.2 depicts how the optimization variables $\boldsymbol{u}$ and $\boldsymbol{s}$ are organized throughout the prediction horizon.

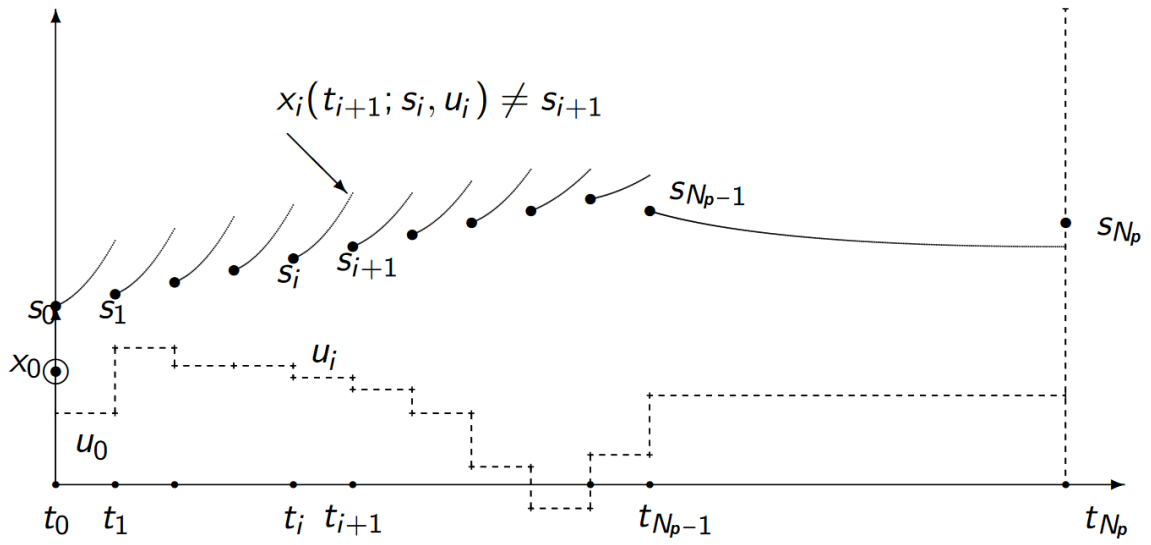

Figure 6.2: Sketch of an OCP converted to NLP using Multiple Shooting. Adapted from [24]

The algorithm for solving multiple shooting considers a NLP of the form (6.2).

$$\begin{aligned} \min_{w} \quad & J(w) \quad = \quad \sum_{0}^{N_p-1} I_k(s_k, u_k) + E(s_{N_p}) \\ \text{s.t.} \quad & s_{k+1} = x(t_{k+1}; s_k, u_k),\ k \in [0, N_p - 1] \quad \text{(Continuity)} \\ & h(s_{k+1}, u_k) \leq 0,\ k \in [0, N_p - 1] \quad \text{(Discretized path constraints)} \\ & s_0 = x_0 \qquad \text{(Fixed initial state)} \\ & u(k) = u(N_c),\ k \in [N_c + 1, N_p] \quad \text{(Control Horizon)} \end{aligned} \tag{6.2}$$

The continuity constraints exist to guarantee that the node states $\boldsymbol{s}$ converge to the predicted states $\boldsymbol{x}$ and the notation $x(t_{k+1}; s_k, u_k)$ means that $\boldsymbol{x(t_{k+1})}$ is the result of solving the ODE over one sampling time with initial state $\boldsymbol{s_i}$ and constant input $\boldsymbol{u_i}$. The prediction of $\boldsymbol{x}$ over the prediction horizon was performed by using Runge-Kutta 4th order to integrate the differential equation. This NLP can be given to a sequential quadratic programming (SQP) solver. The resulting optimization variable $\boldsymbol{w}^*$ that minimizes $J(w)$ has the optimal values of $\boldsymbol{s}^*$ and $\boldsymbol{u}^*$ for the given constraints. The first value of $\boldsymbol{u}^*$ is $\boldsymbol{u}_\mathbf{0}^*$ and it is the control input used in the receding horizon control principle.

Among the benefits of using multiple shooting is that the initial $\boldsymbol{w}$ can be exploited to speed up the convergence of the NLP [26]. Hence, $\boldsymbol{w}^*$ can be extrapolated following equation (6.3) and used as the initial $\boldsymbol{w}$ of the next NLP.

$$w_{extrapolated} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{N_p} \\ 2s_{N_p} - s_{N_p-1} \\ u_1 \\ u_2 \\ \vdots \\ u_{N_p-1} \\ 2u_{N_p-1} - u_{N_p-2} \end{bmatrix} \tag{6.3}$$

6.3 NMPC Trajectory Tracking Formulation

The trajectory given to the controller is a discretized series of poses and velocities $z_k^{ref} = [N_k, E_k, \psi_k, v_{x,k}]^T$, where k is the index of the time step. The aim of the NMPC is to lead the error $\tilde{z}_k = z_k^{ref} - z_k$ to 0 which happens when the actual trajectory converges to the reference trajectory.

Since the tracked state $\boldsymbol{z}$ has less elements than the state variable $\boldsymbol{x}$ (defined on the section about the dynamic model, 5.2), the cost function $J(w)$ will be defined to include only $\boldsymbol{z}$ and $\boldsymbol{u}$. Nonetheless, the NLP optimization variable $\boldsymbol{w}$ will still include $\boldsymbol{s}$ based on $\boldsymbol{x}$ since it is needed for prediction of future states.

The new cost function $J(z, z^{ref}, u)$ is shown on Eq. 6.4.

$$J(z, z^{ref}, u) = \sum_{0}^{N_p-1} I_i(z_i, z_i^{ref}, u_i) + E(z_{N_p}, z_{N_p}^{ref}) \tag{6.4}$$

$$\begin{cases} \tilde{z}_i = z_i^{ref} - z_i = \begin{bmatrix} \tilde{N}_i \\ \tilde{E}_i \\ \tilde{\psi}_i \\ \tilde{v}_{x,i} \end{bmatrix} = \begin{bmatrix} \tilde{p}_i \\ \tilde{v}_{x,i} \end{bmatrix} \\ I_i(z_i, u_i) = \tilde{p}_i^T \, Q_p \, \tilde{p}_i + Q_v \, \tilde{v}_{x,i}^2 + u_i^T \, Q_u \, u_i \\ E(z_{N_p}) = \tilde{p}_{N_p}^T \, Q_{pf} \, \tilde{p}_{N_p} \end{cases}$$

The matrices Q_p, Q_v, Q_{pf} and Q_u are the weights associated to deviations from the reference trajectory and to control inputs. These weights should be tuned according to the needs of the problem:

Q_p, Q_{pf} : Weights associated to path deviations. Increasing them may reduce the error distance between the reference path and the actual path.

Q_v : Weight associated to the reference velocity. Increasing it may prioritize control inputs that don't affect the velocity at the cost of path deviation.

Q_u : Weight associated to control inputs. Increasing it may reduce actuators' forces and torques at the cost of deviation from the reference trajectory.

The NMPC scheme is based on solving iterations of NLP until a maximum number of iterations is reached or the goal position is reached. At each iteration, the NLP solver receives an updated reference trajectory segment and inequalities representing constraints and finds the optimal $\boldsymbol{w}^*$ that minimizes the cost (6.4) and then $\boldsymbol{u_0^*}$ is extracted and selected as the optimal control input. The Algorithm (5) shows the NMPC scheme.

Algorithm 5 NMPC Trajectory Tracking

```
it ← 0
while it < maxIteration do
    x_0 ← MeasureState()
    if CheckGoalReached(x_0) then
        break
    end if
    Z_ref ← ReferenceSegment(x_0, N_p)
    A, b ← WallConstraints(w_old)
    c_static ← StaticObstacles()
    c_dynamic ← DynamicObstacles()
    w_new ← SolveNLP(x_0, z_ref, w_old, A, b, c_static, c_dynamic, ···)
    u_0^* ← ExtractInput(w_new)
    w_old ← ExtrapolateVariable(w_new)
    it ← it + 1
end while
```

The *MeasureState()* function returns the current state of the boat. The function *CheckGoalReached(x_0)* returns true if the current state x_0 is close enough to the destination according to some criterion, in this case euclidean distance was used.

The function *ReferenceSegment(x_0, N_p)* receives the most recent measured state of the boat and finds the point z_0^{ref} in the reference trajectory that is closest to it in

euclidean distance. The function returns a reference segment as a set of points $Z^{ref} = \{z_0^{ref}, z_1^{ref}, \cdots, z_{N_p}^{ref}\}$.

The functions *WallConstraints(w_{old})*, *StaticObstacles()* and *DynamicObstacles()* return constraints to the state $\boldsymbol{x}$ using data from the environment. These constraints are all grouped to form the discretized path constraints inequalities and they will be explained in-depth in the next sections.

The function *SolveNLP($\cdots$)* passes the current state x_0, an initial value of the optimization variable w_{old}, the reference segment Z^{ref} and the constraints to the NLP solver with the form (6.2) and cost function (6.4). The result is $\boldsymbol{w}^*$ that minimizes the cost.

The function *ExtractInput(w_{new})* extracts the first control input $\boldsymbol{u_0^*}$ according to the receding horizon principle. The function *ExtrapolateVariable(w_{new})* extrapolates the optimization variable to be used for the next NMPC iteration according to equation (6.3).

6.4 Constraints

One of the main benefits of using MPC based algorithms is the ability to consider constraints explicitly. This is done by providing conditions that should be satisfied by the NLP solver. The formulations of these constraints depend on the problem to be solved, in the case of NMPC trajectory tracking two categories of constraints can be defined: controlled system limitations and obstacle avoidance.

The first category is related to upper and lower bounds on the possible values of states and control inputs. These values come from physical limitations of the boat and its actuators. The second category is related to the description of obstacles to be avoided by the boat, such as walls and other obstacles.

In order to agree with the the notation used in the NLP formulation (6.2), the following inequality constraints are grouped inside a function $h(s_{k+1}, u_k) \leq 0$ with the variable $\boldsymbol{s}$ acting as the state variable $\boldsymbol{x}$.

6.4.1 State and input constraints

One kind of constraint commonly considered in MPC-based algorithms are limits on states $\boldsymbol{x}$ and control inputs $\boldsymbol{u}$. They can be illustrated as follows:

$$\begin{cases} x_{min} \leq x_k \leq x_{max}\ ,\ k = 1, \cdots, N_p \\ u_{min} \leq u_k \leq u_{max}\ ,\ k = 0, \cdots, N_p - 1 \end{cases} \tag{6.5}$$

In addition, the rate of change of the control inputs $\boldsymbol{\Delta u}$ can also be bounded to avoid abrupt changes that could lead to passenger discomfort or even unfeasible inputs for the boat. The rate of change is calculated as $\boldsymbol{\Delta u_k} = |u_k - u_{k-1}|$, where $k = 0, \cdots, N_p - 1$ and $\boldsymbol{u_{-1}}$ is defined as the input $\boldsymbol{u_0^*}$ of the last NLP iteration. The constraint is given by the inequality:

$$\Delta u_k \leq \Delta u_{max}\ ,\ k = 0, \cdots, N_p - 1 \tag{6.6}$$

The state vector $\boldsymbol{x} \in \mathbb{R}^6$ has its elements defined in Chapter 5 along with the nonlinear state space function (5.8). The boundary vectors x_{min} and x_{max} are on equation (6.7). In general, the pose of the boat η can be considered unbounded varying from $[-\infty, \infty]$ while the velocity vector ν has its values bounded by canal regulations and design decisions.

$$x_{min} = \begin{bmatrix} N_{min} \\ E_{min} \\ \psi_{min} \\ v_{x,min} \\ v_{y,min} \\ r_{min} \end{bmatrix} \quad x_{max} = \begin{bmatrix} N_{max} \\ E_{max} \\ \psi_{max} \\ v_{x,max} \\ v_{y,max} \\ r_{max} \end{bmatrix} \tag{6.7}$$

The input vector $\boldsymbol{u} \in \mathbb{R}^m$ has m elements each one representing an actuator. The boundary vectors u_{min}, u_{max} and Δu_{max} are shown on (6.8). In this case, the limits for u and Δu are given by the physical limitations of the actuators used.

$$u_{min} = \begin{bmatrix} u_{1_{min}} \\ \vdots \\ u_{m_{min}} \end{bmatrix} \quad u_{max} = \begin{bmatrix} u_{1_{max}} \\ \vdots \\ u_{m_{max}} \end{bmatrix} \quad \Delta u_{max} = \begin{bmatrix} \Delta u_{1_{max}} \\ \vdots \\ \Delta u_{m_{max}} \end{bmatrix} \tag{6.8}$$

6.4.2 Wall Constraints

The shape of canals might be non-convex and therefore cannot be easily included as a constraints for the optimal control problem. This issue was also encountered in other studies dealing with obstacles in unstructured environments and the techniques used to delimit the free space decompose it in convex shapes [11], [12].

The solution chosen for this thesis is to create a convex polygon around each initial $\boldsymbol{w}$ given to the NLP solver using the occupancy grid by expanding, from the center of the boat, the sides of a rectangle up to a maximum distance d_{max} or until a wall is reached. All sides expand simultaneously and if one side reaches its limit, the other sides can continue expanding. The convex polygon is shown on Fig. 6.3.

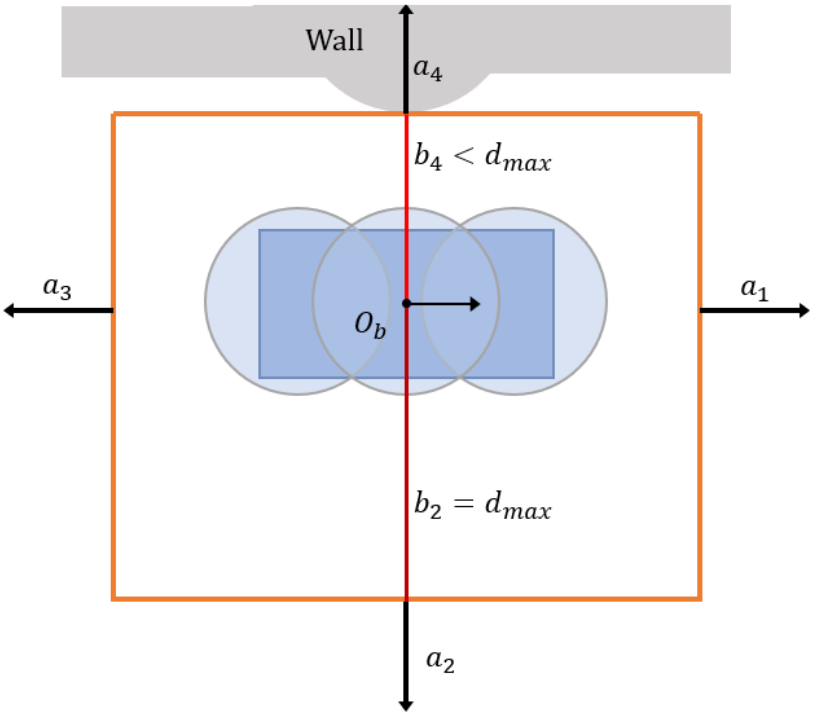

Figure 6.3: Rectangular convex polygon with one side limited by a wall.

The rectangular convex polygon considered has its sides aligned with BODY axis x_b and y_b and the closest distance from each side to the origin of BODY, O_b, is $b_i \leq d_{max}$, where $i = 1, ...,4$ is the index of a side and d_{max} is the maximum expansion distance for each side when no walls are encountered.

Each side of the convex polygon is a hyperplane of the form $a_i^T p_b = b_i$, where p_b is the position $[x_b, y_b]^T$ of a point in the BODY frame. A point p_b is inside the convex region if it satisfies the inequalities $a_i^T p_b < b_i$ for $i = 1...4$ or in matrix form (6.9).

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad A\ p_b < b \tag{6.9}$$

Considering the NMPC scheme, a convex polygon is needed for each time step of the prediction horizon. This can be achieved by expanding around the extrapolation of the predicted positions of the boat calculated at the previous NLP iteration. All of the resulting convex polygons are given to the NLP solver as constraints.

These constraints are useful because substituting p_b by the position of a disc $p_{d,b}^i$ and subtracting the disc's radius from b, it is possible to tell if the boat's disc will collide with walls or not. At each time step of the prediction horizon this check is performed for all boat's discs and only if all discs are inside the convex polygon the path is considered collision-free for the wall constraint.

The function *WallConstraints()* used in the algorithm (5) returns all matrices A and vectors b (6.9) for each time step of the prediction horizon.

6.4.3 Static Obstacles

Static obstacles, Fig. 6.4, refer to objects detected by sensors that were not present in the global occupancy grid and that have zero or near zero velocities. It is assumed that they can be represented as ellipsoids of major-axis a and minor-axis b and orientation ψ with respect to NED.

Collision avoidance for ellipses can be done by inflating the ellipse by the radius of the discs representing the boat r_{disc} and an additional safety radius δ_r. The resulting ellipse is shown on Fig. 6.5.

In order to know if a point $p = (x_p, y_p)$ is outside of an ellipse centered at (x_e, y_e) with major axis a and minor axis b, it is sufficient to consider the inequality that represents the region outside of the ellipse:

$$\frac{(x_p - x_e)^2}{a^2} + \frac{(y_p - y_e)^2}{b^2} > 1 \tag{6.10}$$

By taking into account the inflated ellipse and the transformation matrix between the ellipse local reference frame and the NED global reference frame, it is possible to re-write the inequality 6.10 in matrix form as the inequality 6.11. In this inequality, i represents the index of the boat disc, j represents the index of the obstacle and k represents the time step of the prediction horizon.

$$c_{j,k}^i = 1 - (\Delta p_{j,k}^i)^T (R_n^e(\psi))^T \begin{bmatrix} \frac{1}{(a+r_{disc}+\delta_r)^2} & 0 \\ 0 & \frac{1}{(b+r_{disc}+\delta_r)^2} \end{bmatrix} R_n^e(\psi) \Delta p_{j,k}^i < 0 \tag{6.11}$$

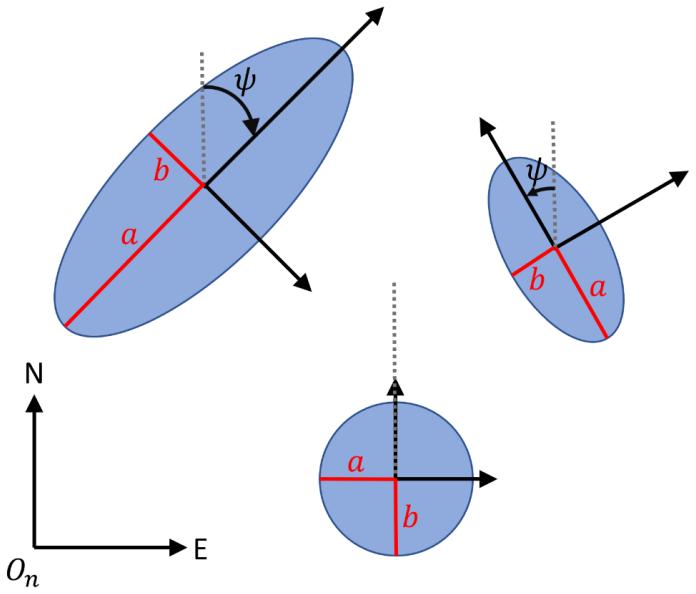

Figure 6.4: Ellipsoids representing static obstacles.

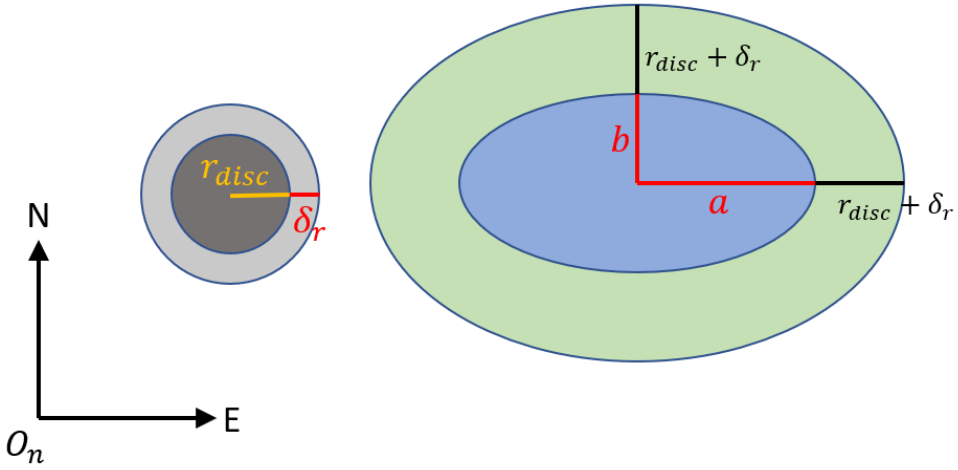

Figure 6.5: Inflation of ellipsoid obstacle to consider boat dimensions and a safety distance.

Where, $\Delta p^i_{j,k}$, defined in Eq. 6.12, is the distance from a boat disc i at time step k to the center of the obstacle j in NED. $R^e_n(\psi)$, defined in Eq. 6.13, is the transformation matrix from the NED global reference frame to the Ellipse local reference frame.

$$\Delta p^i_{j,k} = \begin{bmatrix} \Delta N^i_{j,k} \\ \Delta E^i_{j,k} \end{bmatrix} = \begin{bmatrix} N_{i,k} - N_j \\ E_{i,k} - E_j \end{bmatrix} \tag{6.12}$$

$$\Delta R^e_n = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \tag{6.13}$$

The function *StaticObstacles()* used in Alg. 5 returns inequalities $c^i_{j,static}$ obtained by considering all discs and static obstacles.

6.4.4 Dynamic Obstacles

Dynamic obstacles, Fig. 6.6, are ellipsoids like static obstacles, however they move at a constant velocity v_x in the direction of ψ. Thus, it is necessary to account for the new positions of these obstacles during the prediction horizon.

The position of each dynamic obstacle j is predicted for each time step k of the horizon using equation (6.14), where T_s is the sample time of the horizon and $R_e^n(\psi) = (R_n^e(\psi))^T$ is the transformation matrix from the Ellipse local reference frame to the NED global reference frame

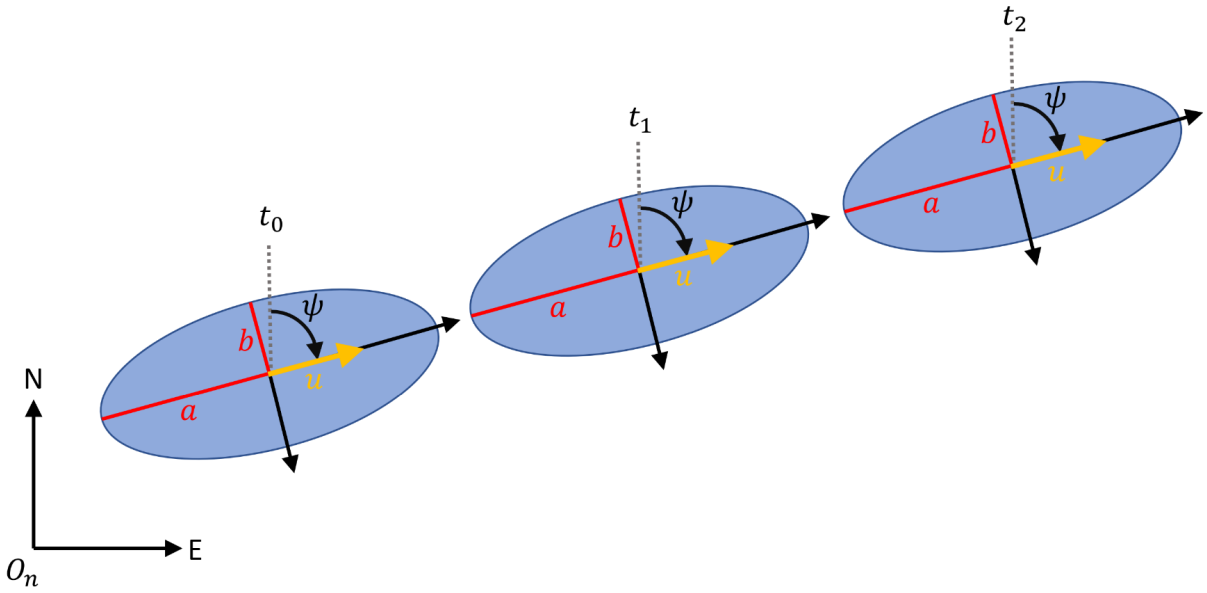

Figure 6.6: Ellipsoid representing a dynamic obstacle moving at constant velocity u.

$$\begin{bmatrix} N_{j,k+1} \\ E_{j,k+1} \end{bmatrix} = \begin{bmatrix} N_{j,k} \\ E_{j,k} \end{bmatrix} + T_s R_e^n(\psi) \begin{bmatrix} v_x \\ 0 \end{bmatrix} \tag{6.14}$$

After predicting the positions of the dynamic obstacles, inequality constraints (6.11) are created for each time step of the prediction horizon. The only difference is that $\Delta p_{j,k}^i$ is redefined to account for the obstacle's position over time (6.15).

$$\Delta p_{j,k}^i = \begin{bmatrix} \Delta N_{j,k}^i \\ \Delta E_{j,k}^i \end{bmatrix} = \begin{bmatrix} N_{i,k} - N_{j,k} \\ E_{i,k} - E_{j,k} \end{bmatrix} \tag{6.15}$$

6.5 Infeasible Solutions and Collisions

One of the main concerns of autonomous vehicles is passenger safety and how to deal with collisions. As seen in previous sections, the NMPC controller can avoid obstacles by giving their constraints to the NLP solver. However, this is not enough to avoid all collisions since there might be cases where the solution of the optimization problem is infeasible.

An infeasible solution is such that the constraints given to the solver are not met. Since these constraints refer to obstacles, an infeasible solution indicates a possible collision. In these situations, it was decided that it is preferable to halt the boat's motion rather than risking a collision. This is performed by overriding the control input with a propulsion in

the opposite direction of the boat (or null input if the boat velocity is near 0). Therefore, the boat is able to stop quickly albeit with an impulse in its jerk which is acceptable since comfort is not a priority during collision avoidance.

Chapter 7

Methodology

The global path planning was performed using a global occupancy grid created from an aerial view obtained from Google My Maps with a filter to distinguish between landmasses and waterways. Afterwards, this grid is given to the Dubins RRT* path planner to find an initial path to the destination. Finally, this path goes through the post-processing described in subsection 3.5 using a Forwards-Backwards pass in this order and the output is the path used in the next steps.

The global trajectory planning was performed by assigning longitudinal velocities to points of the reference path. The velocity profile follows a smoothed trapezoidal profile and each point of the path is assigned a corresponding velocity and time instant depending on its distance from the start of the path when converted to the Frenet frame. The output is the reference trajectory.

The local motion planner is a NMPC trajectory tracker that is able to avoid collisions with walls and obstacles. The reference given to the motion planner is a sequence of positions and velocities along the discretized trajectory. At each time step, defined by a sampling time, the boat receives updated information about its position and obstacles positions.

The methods were implemented in MATLAB since it has toolboxes for autonomous navigation and nonlinear optimization. The access to different functions for plotting the solution and the occupancy grid was also a benefit of using MATLAB. The methodology can be organized in the following steps:

1. Define boat parameters and the differential equations for a 3 DOF model.
2. Prepare the global occupancy grid from an aerial view of the canal.
3. Generate a feasible path from the start point to the destination using Dubins RRT* and apply the post-processing pruning.
4. Generate the velocity profile with given cruise velocity and acceleration limits then create reference trajectory by combining the reference path with the velocity profile.
5. Initialize the NMPC parameters (horizons, weights and bounds) and use it to track the reference trajectory.

7.1 Boat Parameters

The autonomous electric boat considered in the simulations is based on the ROBOAT II [27], an autonomous surface vessel developed in a joint research between the Massachusetts

Institute of Technology (MIT) and the Amsterdam Institute for Advanced Metropolitan Solutions (AMS Institute).

The ROBOAT II has a length of 2 meters and a width of 1 meter. The matrices M, $C(\nu)$ and $D(\nu)$ that appear in the model (5.7) are as follows:

$$M = diag\{m_{11}, m_{22}, m_{33}\} \tag{7.1}$$

$$C(\nu) = \begin{bmatrix} 0 & 0 & -m_{22}v \\ 0 & 0 & m_{11}u \\ m_{22}v & -m_{11}u & 0 \end{bmatrix} \tag{7.2}$$

$$D(\nu) = diag\{X_u, Y_v, N_r\} \tag{7.3}$$

Where, $m_{11} = 172$ kg, $m_{22} = 188$ kg, $m_{33} = 24$ kg·m^2, $X_u = 38$ kg/s, $Y_v = 168$ kg/s and $N_r = 16$ kg·m^2/s.

The vector τ (7.4) was calculated considering the configuration in Fig. 7.1, where two thrusters are located at the stern. The thruster aligned with x_b, generating force X, is responsible for the longitudinal acceleration of the boat and the thruster aligned with y_b, generating force Y, is responsible for steering the boat.

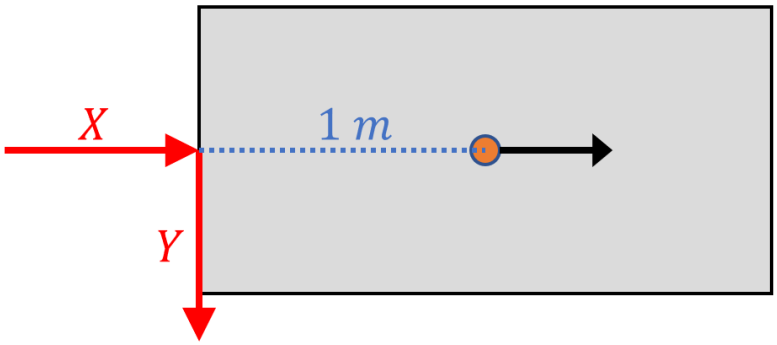

Figure 7.1: Position of the actuators in the experimental boat.

$$\tau = \begin{bmatrix} X \\ Y \\ -Y \end{bmatrix} \tag{7.4}$$

The experiments used the nonlinear ODE (7.5), derived from the symbolic nonlinear ODE (5.8) and (7.1-7.4).

$$\frac{d}{dt}\begin{bmatrix} N \\ E \\ \psi \\ u \\ v \\ r \end{bmatrix} = \begin{bmatrix} u\cos\psi - v\sin\psi \\ v\cos\psi + u\sin\psi \\ r \\ \frac{X}{172} - \frac{19u}{86} + \frac{47vr}{43} \\ \frac{Y}{188} - \frac{42v}{47} - \frac{43ur}{47} \\ -\frac{Y}{24} - \frac{2r}{3} - \frac{2uv}{3} \end{bmatrix} \tag{7.5}$$

The obstacle avoidance performed by the NMPC requires the boat representation using discs. In this case, the boat was divided into 3 discs of radius $r_{disc} = 0.6$ m, centered at $(-0.7, 0)$, $(0, 0)$ and $(0.7, 0)$ in BODY coordinates as shown on Fig. 7.2.

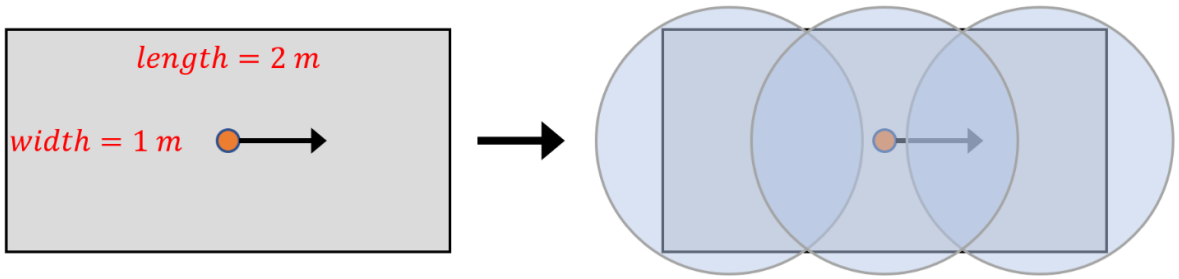

Figure 7.2: Representation of the boat used for tests as a set of discs.

Chapter 8

Results and Discussion

This chapter is divided in two sections about trajectory planning and motion control. In the first section, the trajectory planner will be discussed by separating the path planner and the velocity generation. The path planner was tested on part of a canal and its performance will be discussed in terms of path length, optimality and clearance (distance to walls). The velocity generation is done by selecting a pair of longitudinal velocity and acceleration that can be achieved by the boat under the radius of curvature constraint.

In the second section the motion control is analysed for different operating conditions that simulate real situations found on canals and the performance is discussed in terms of tracking deviation and collision avoidance.

8.1 Global Trajectory Planning

The path planner used has its parameters show on table 8.1 and to account for the time required to accelerate the boat, straight segments were added at the start and end of the path to avoid turning the boat at low velocities. The path planner will be evaluated on a portion of canal shown on Fig. 8.1, where start and goal positions are marked.

Table 8.1: Path Planner Parameters

Number of iterations	$1 \cdot 10^4$
Grid Inflation Radius	0.6 m
Radius of curvature	20 m
Straight ends length	5 m

The path planning works in two stages: path generation and post-processing. Multiple runs were performed on both stages to analyse the consistency of the paths defined as their variation in form and length.. The Fig. 8.2 shows the output paths of the Dubins curve based RRT* algorithm, before post-processing, in 30 runs.

It is clear that the paths generated are not consistent, with high variation on their form and length. Even though a theoretically optimal algorithm (RRT*) was used, it was not able to generate optimal paths under the constraints imposed to the planner (Dubins steering function), this will be further discussed later in this section. The next stage is to

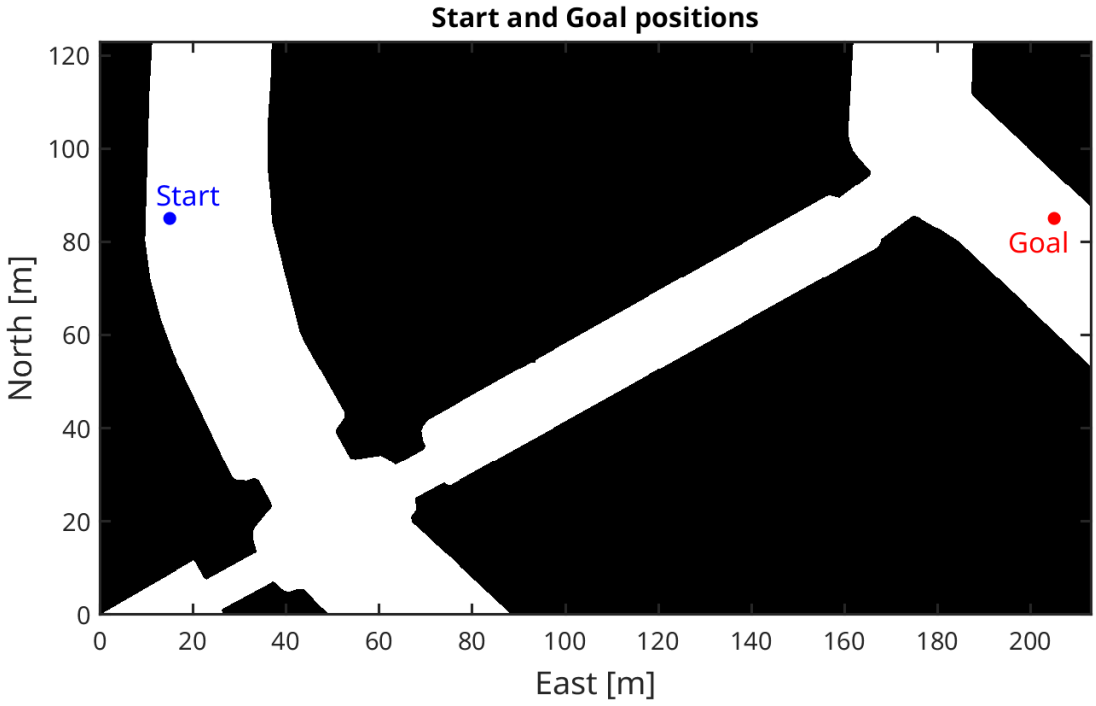

Figure 8.1: Canal map with start and goal positions

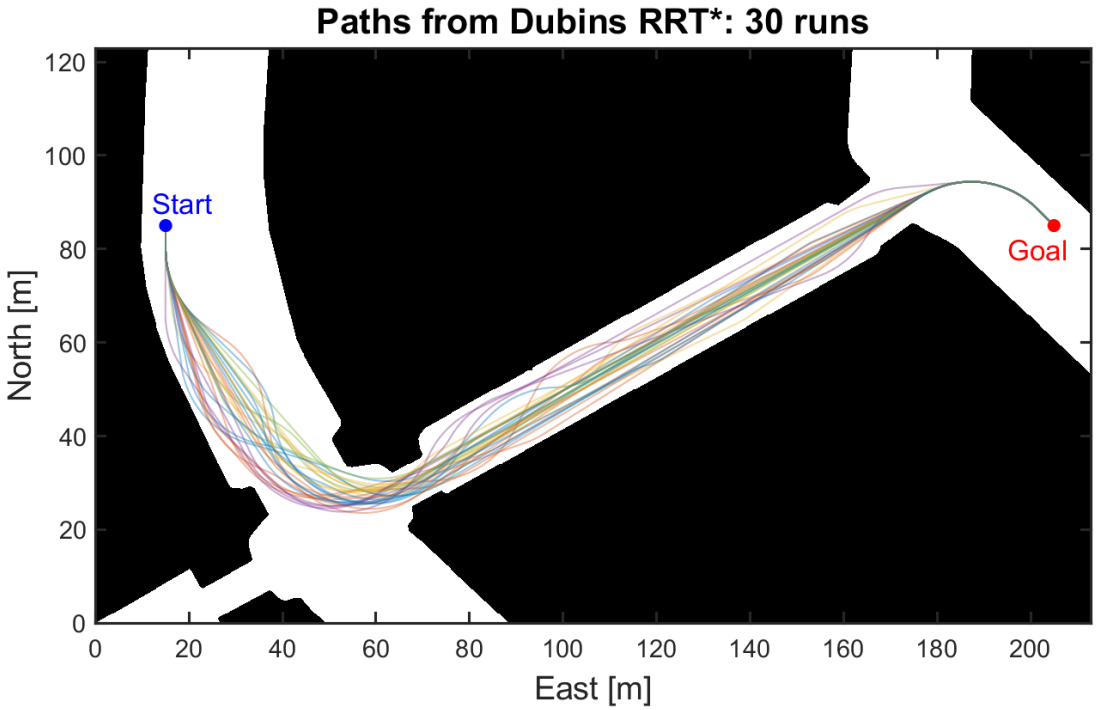

Figure 8.2: Output Paths generated in 30 runs of Dubins RRT*

apply post-processing to these paths and analyse how they change. The resulting paths are shown on Fig. 8.3.

The biggest change of Fig. 8.3 with respect to Fig. 8.2 is that the paths are more consistent without much variation in length and number of curves. This shows that the post-processing was able to prune unnecessary waypoints leading to a path with more straight segments that are easier to track.

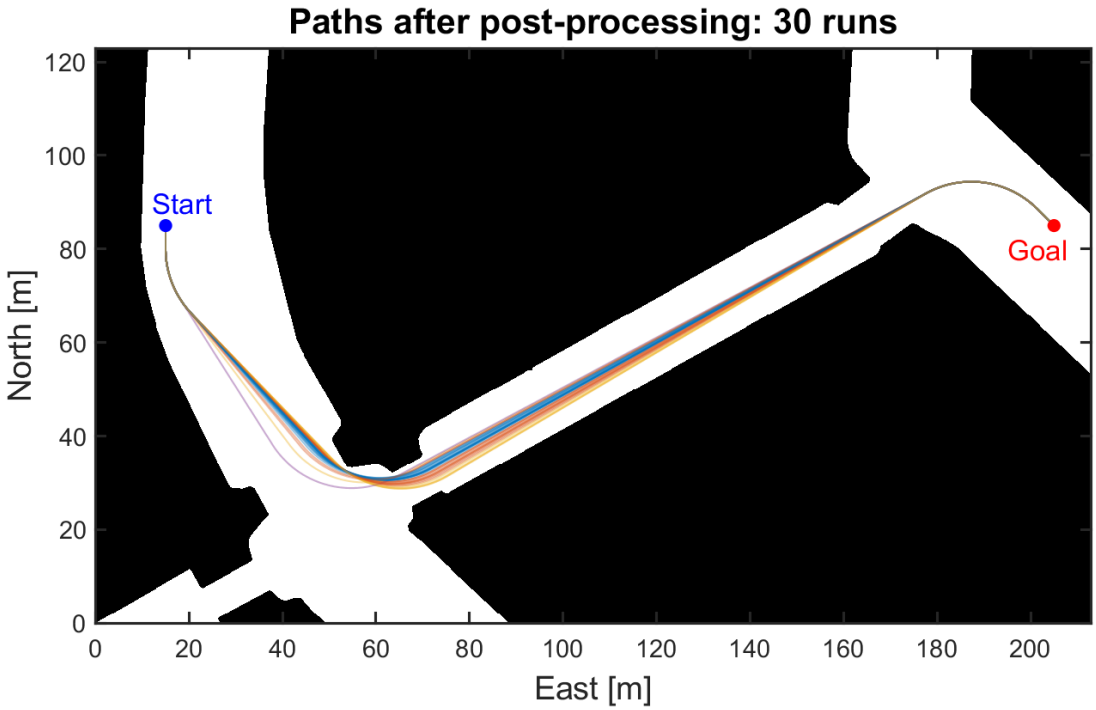

Figure 8.3: Post-processed paths generated in 30 runs

In order to quantify the performance of the post-processing algorithm, the number of runs was increased from 30 to 100 and paths were grouped in histograms, Fig. 8.4, based on their length. The histogram shows a distinction between length distributions before and after pruning.

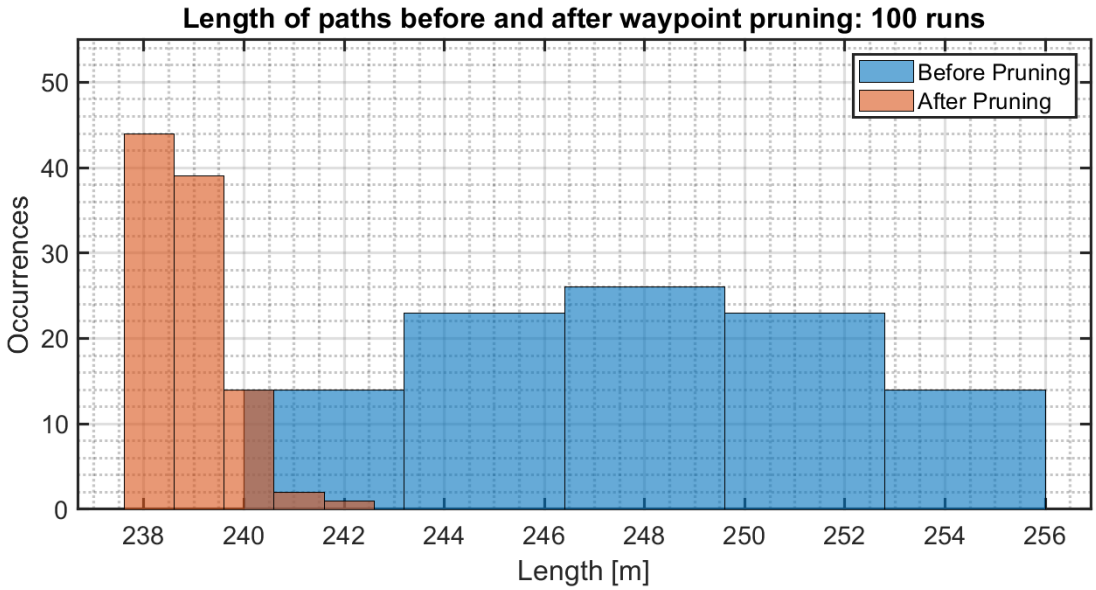

Figure 8.4: Length distribution of paths before and after waypoint pruning in 100 runs

The table 8.2 has the average length before and after pruning and their standard deviation. It also includes the average of the percentile improvement due to post-processing, defined in the following formula, where L refers to path length:

$$Improvement(\%) = 100 \cdot \frac{L_{before} - L_{after}}{L_{before}} \tag{8.1}$$

Table 8.2: Average lengths and improvement

Average Length (Before pruning)	248.09 ± 3.67 m
Average Length (After pruning)	238.94 ± 0.79 m
Average Improvement	3.67 ± 1.43 %

One remark about the post-processed paths is that they tend to stay close to walls, this is a consequence of the algorithm used for post-processing that removes waypoints to generate the most optimal path in terms of length. This effect could be undesirable for the operation of the boat since that in the presence of disturbances the boat may get too close to walls and in this case the controller should be able to avoid collisions. Another issue is that the adoption of autonomous vessels could suffer from passengers not feeling safe on vessels that steer in direction of walls or stay close to them.

In the context of Dubins curve based RRT*, a possible solution for this problem is to increase the grid inflation radius to force the boat to be further from walls even if the post-processed path is close to the inflated grid walls. However, this is not ideal when applied to sections with narrow passages because it can hinder the ability of the planner to cross them due to blockage. Another solution is to use a more complex post-processing technique that takes into account length and clearance [15].

Regarding path optimality, the RRT* implementation using Dubins curves as a steering function was not able to achieve optimality and a possible reason is that this steering function considers a fixed radius of curvature for connecting two states which is not ideal.

In light of this, if an optimal path is of utmost importance for the application it may be better to consider other variations of RRT* such as Kinodynamic RRT* which was shown to generate optimal paths when kinodynamic constraints are included with the downside of a more complex steering function that solves an optimal control problem to connect two states. On the other hand, if paths of low cost, but not necessarily optimal, are enough for the application the approach in this thesis is capable of generating such paths. Furthermore, considering the performance over multiple runs, it could be useful to run the entire algorithm a few times (3-5 times) and select the best path among them.

After the path planning algorithm finishes, the velocity generation is performed considering a cruise velocity (surge velocity) respecting canal regulations and boat limitations. These limitations are the maximum surge velocity of the boat and the maximum velocity in which the boat can make the turn with the planned radius of curvature. The surge acceleration is chosen according to boat dynamical limitations. The table 8.3 has the parameters used to generate the smoothed trapezoidal velocity curve shown on Fig. 8.5.

After the velocity profile is generated, it is used as a time law for the path and the reference North (N), East (E) and Yaw (ψ) variables are derived. They are shown on Fig. (8.6, 8.7). At the end of this step the trajectory planning is complete.

Table 8.3: Velocity Generation Parameters

Cruise velocity	$2\ \frac{m}{s}$
Maximum surge acceleration	$0.2\ \frac{m}{s^2}$

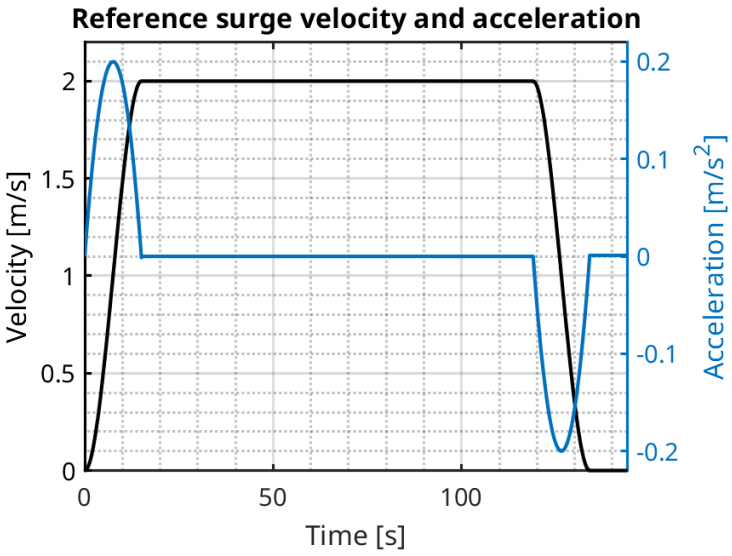

Figure 8.5: Velocity and acceleration profiles

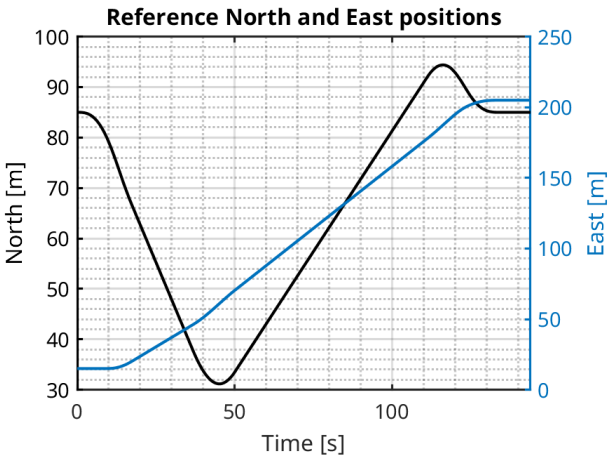

Figure 8.6: Reference North and East variables derived from the reference velocity

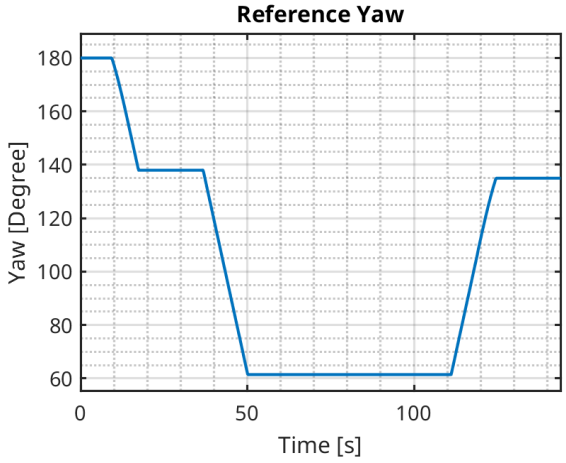

Figure 8.7: Reference Yaw derived from the reference velocity

8.2 Local Motion Planner

The NMPC controller was evaluated on different test cases that simulate real situations that could happen in a canal. The NMPC controller used in all of the test cases is the same with parameters shown on tables (8.4, 8.5, 8.6). The motion planner is considering an additional safety distance of 0.2 m, so the minimum distance to walls and obstacles is 0.8 m.

Table 8.4: NMPC Horizons

Prediction Horizon (Tp)	8 s
Control Horizon (Tc)	4 s
Sampling Time (Ts)	1 s

Table 8.5: NMPC Weights

Position Weight (Q_p)	$diag(1 \cdot 10^3, 1 \cdot 10^3, 1 \cdot 10^4)$
Velocity Weight (Q_v)	$1 \cdot 10^5$
Input Weight (Q_u)	$diag(1, 1, 1)$
Final Position Weight (Q_{pf})	$diag(2 \cdot 10^3, 2 \cdot 10^3, 2 \cdot 10^4)$

Table 8.6: NMPC Bounds

State Upper Bounds (x_{max})	$[\infty, \infty, \infty; 2.2, \infty, 0.1]^T$
State lower Bounds (x_{min})	$-[\infty, \infty, \infty, 2.2, \infty, 0.1]^T$
Input Upper Bounds (u_{max})	$[100, 100]^T$
Input lower Bounds (u_{min})	$-[100, 100]^T$
Input Variation Bounds (Δu)	$[20, 20]^T$

The boat's surge velocity and acceleration are the same as the previous section's table 8.3. The choice of a cruise velocity of 2 m/s is reasonable since this is near the limit on Venice's canals and it is also near the boat's designed limit.

In the figures that show the trajectory taken by the boat, the obstacles will be presented in the following ways:

- Static Obstacles: two concentric ellipses. The inner ellipse, in black, is the actual size of the obstacle and the outer ellipse, in green, is the inflated ellipse considering the boat's dimensions. Collision is avoided if the boat's simulated trajectory does not enter the green ellipse.
- Dynamic Obstacles: their initial position is shown as a purple ellipse without considering the ellipse inflation. In order to show the avoidance of this obstacle, a rainbow

colormap associated with time flow is used to color all of the boats. If two boats are on the same spot and with different colors there is no collision, but if they have the same color they collided.

8.2.1 Case 1: Regular canal with Mixed Obstacles

This example is an extension of the previous section's example. The reference trajectory shown on Fig. (8.5-8.7) is going to be tracked, but static and dynamic obstacles were added to check if the motion planner is able to operate correctly. The Fig. 8.8 shows the simulated trajectory as a red dashed line.

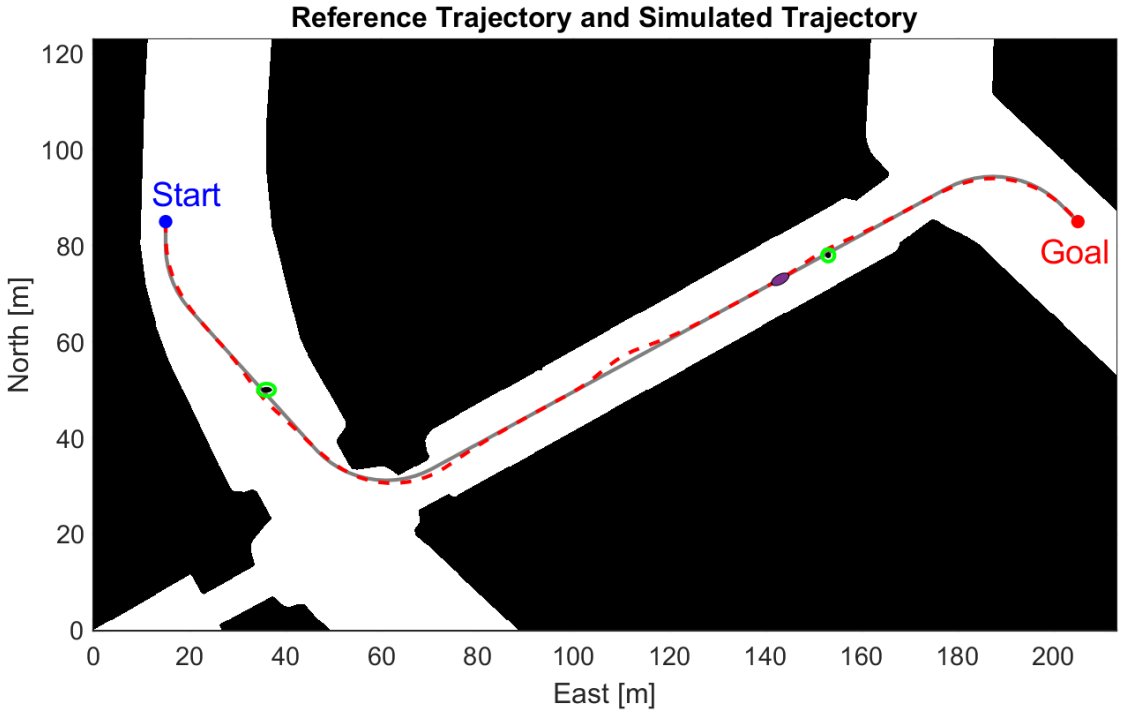

Figure 8.8: Case 1: Trajectory tracking in a canal with mixed obstacles.

In order to check collision avoidance, the progression of the autonomous boat will be analysed with more details near obstacles. The Fig. (8.9, 8.10) show that the boat was able to avoid both static obstacles.

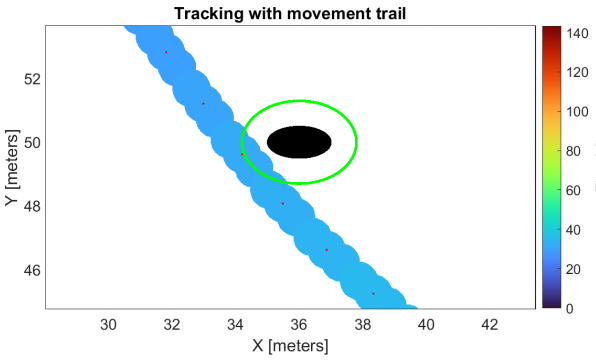

Figure 8.9: Case 1: Closer view of the first static obstacle.

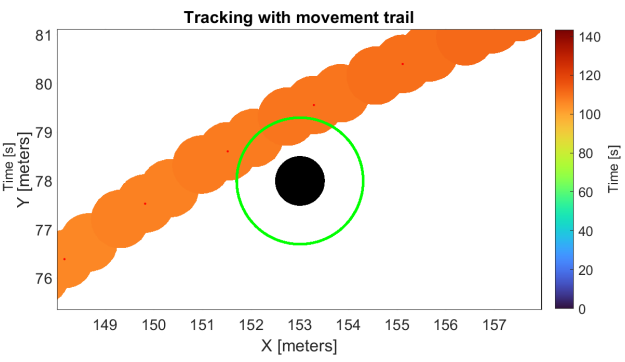

Figure 8.10: Case 1: Closer view of the second static obstacle.

The Fig. 8.11 shows a time interval of the simulation where the autonomous boat starts steering at around 60 seconds and by 80 seconds it has already avoided the dynamic obstacle. It is important to note that this maneuver was not COLREGs compliant, since the autonomous boat should have steered to starboard. In order to account for these regulations, the NMPC cost function should be modified to increase the cost of non-compliant maneuvers.

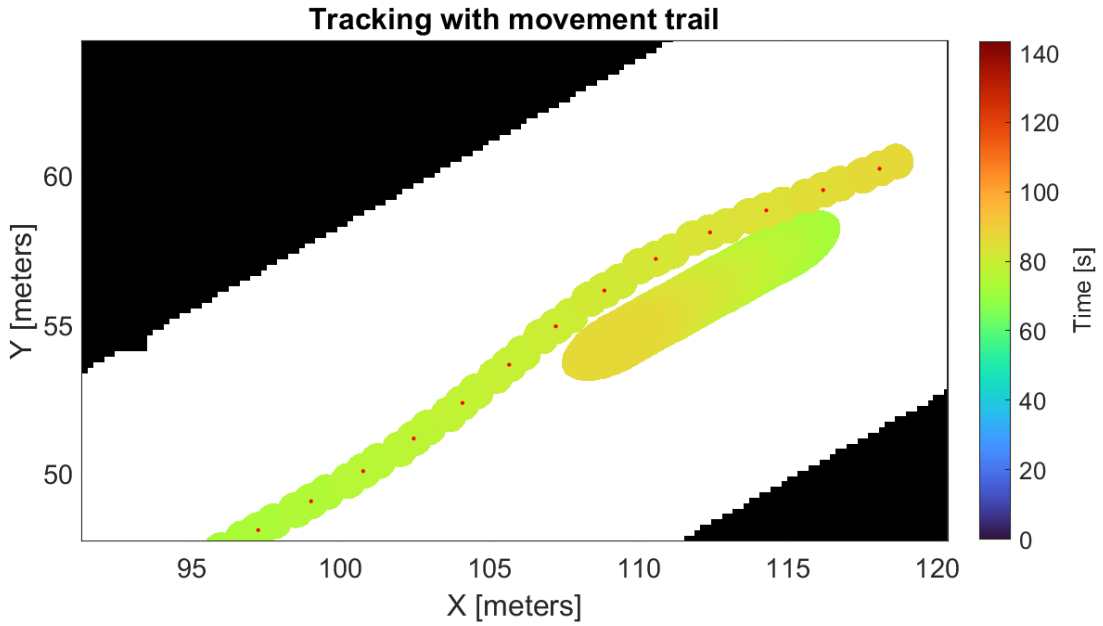

Figure 8.11: Case 1: Intermediate view of the time progression.

The next step is to check the tracking performance of the controller. The Fig. (8.12-8.14) show well the simulated state variables were tracked. The position tracking performs well in general with worse performance near obstacles and on turns.

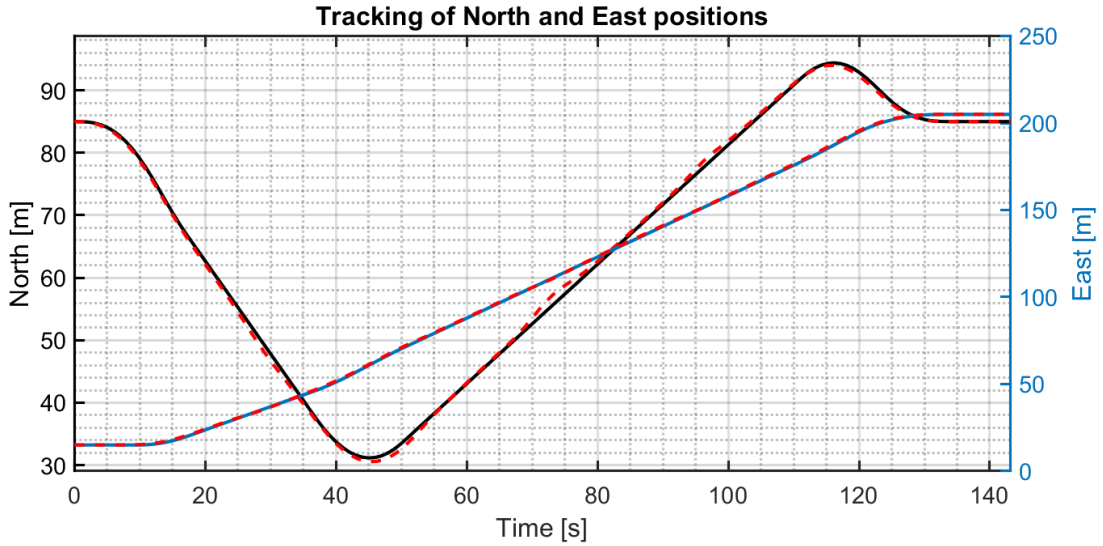

Figure 8.12: Case 1: Tracking of the North and East state variables.

The simulated yaw angle changes a bit earlier than the reference and has a small overshoot which is not necessarily a problem. A possible reason is that the NLP solver

found that the lowest cost solution is achieved by sacrificing yaw tracking and improving position and velocity tracking.

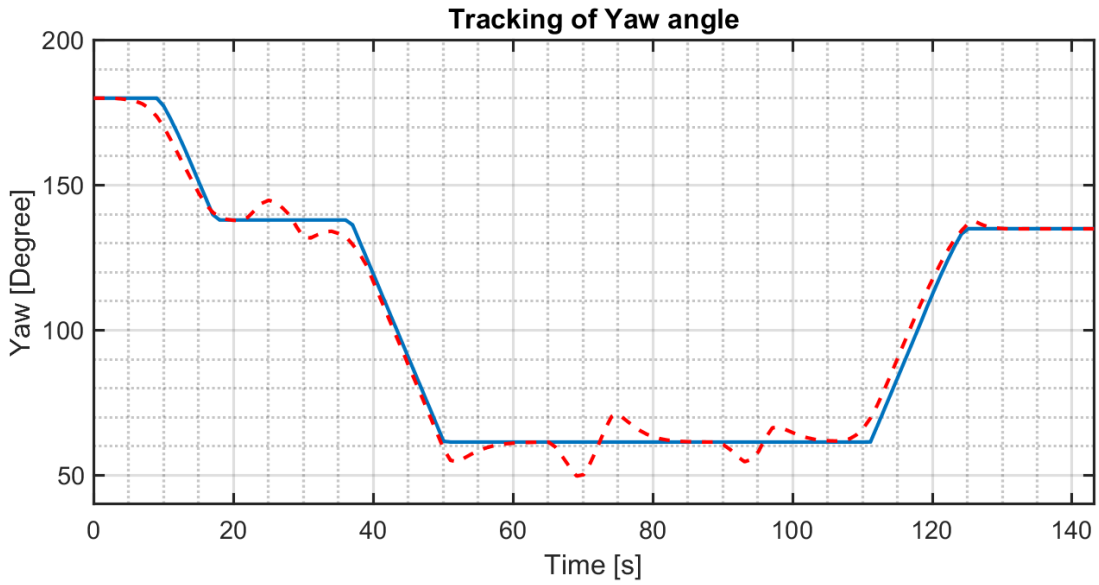

Figure 8.13: Case 1: Tracking of the yaw state variable

The velocity tracking is performing well with just a small variation around reference values. This is expected since a high weight was given to the velocity to prioritize a constant cruise velocity.

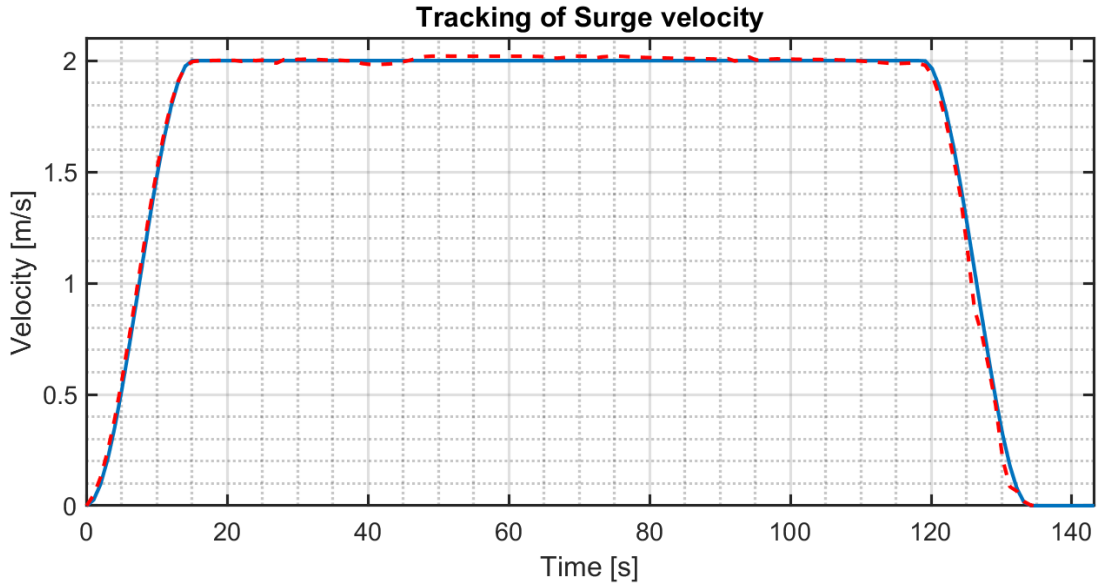

Figure 8.14: Case 1: Tracking of the surge velocity state variable.

The Fig. 8.15 shows the deviations of the boat's pose from the reference. The positioning deviation is lower than the boat's body length and it increases whenever the yaw deviation is different from zero because in this case the lateral deviation increases. The yaw deviation magnitude increases when turning and reduces when following a straight segment. At the end of the trajectory, both deviations are near zero and the boat is able to reach the goal state almost perfectly.

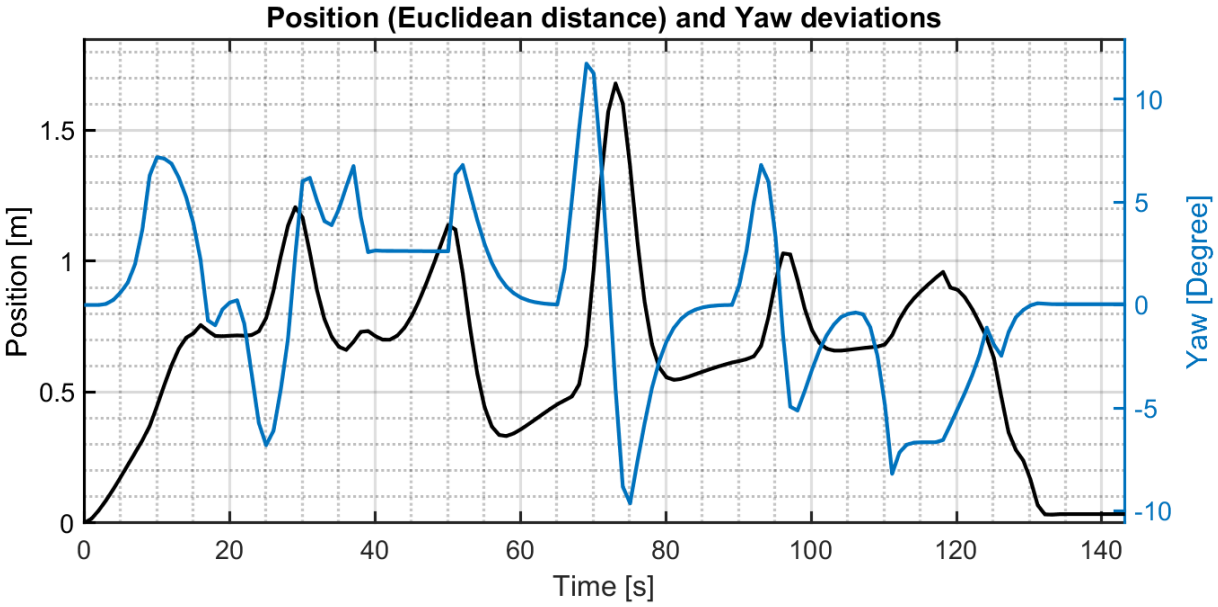

Figure 8.15: Case 1: Position and Yaw deviations.

The last step is to analyse the control inputs during the operation. Fig. 8.16 shows that the longitudinal input follows the same trend as the velocity profile, maintaining the necessary force to stabilize the cruise velocity at the reference. The lateral thruster changes direction constantly to make turns and steer the boat out off and into the reference trajectory when an obstacle appears. The lack of change in the direction of the longitudinal thruster indicates that no infeasible solution was found during the operation.

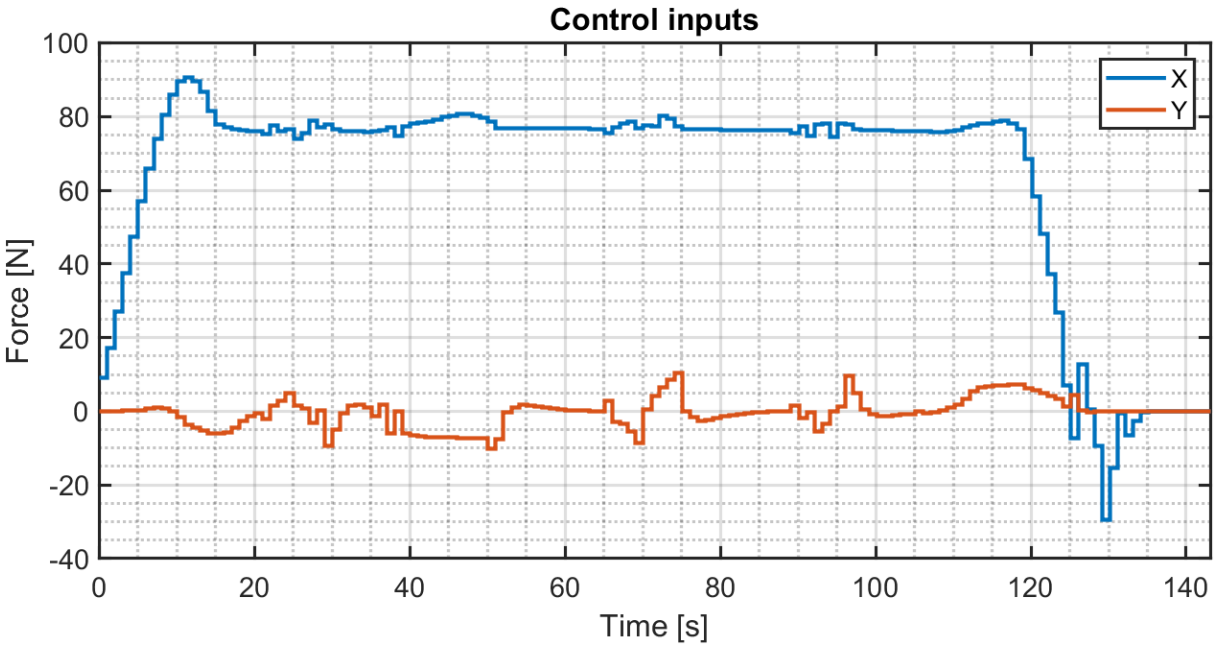

Figure 8.16: Case 1: Control inputs.

8.2.2 Case 2: Narrow Canal with Mixed Obstacles

This example was crafted to show what happens when a static obstacle is blocking the reference trajectory and another vessel is crossing the only passage available. The expected behaviour is to first wait for the passage to be clear and then continue the movement. The Fig. 8.17 shows the reference path in grey and the simulated path in red.

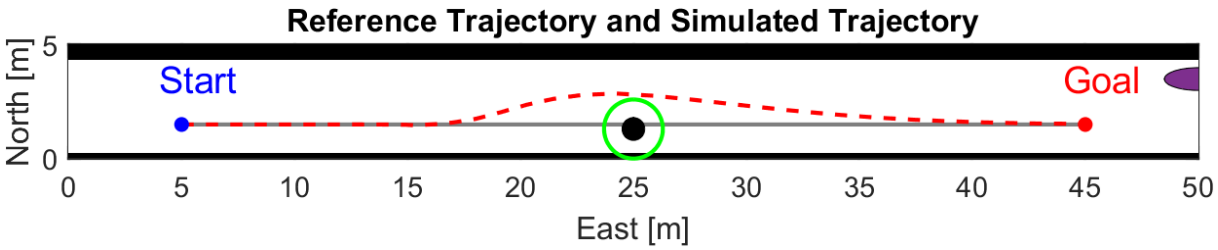

Figure 8.17: Case 2: Trajectory tracking in a narrow passage with mixed obstacles.

The figure makes it clear that the simulated trajectory did not collide with the static obstacle blocking the canal. The Fig. 8.18, with the time progression of both vessels, shows that they did not collide and that the autonomous boat stopped and waited the other vessel before crossing the obstacle.

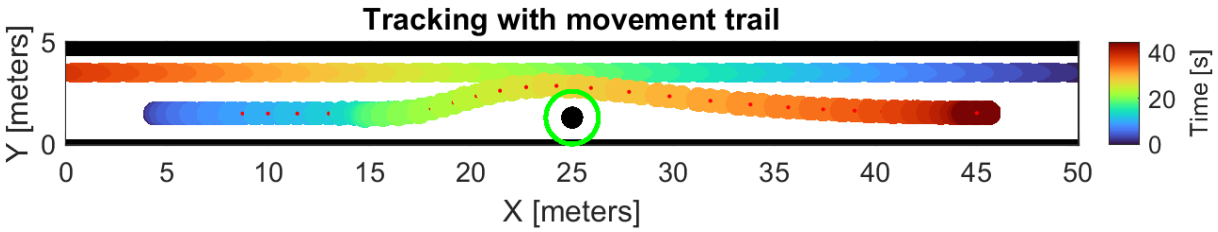

Figure 8.18: Case 2: Time progression of the autonomous boat and the dynamic obstacle.

The surge velocity tracking, Fig 8.19, shows that the autonomous boat almost stopped and started moving again at around 15 seconds. Inspecting Fig 8.18 at 15 seconds, the dynamic obstacle has already crossed the static obstacle and the passage is clear for the autonomous boat.

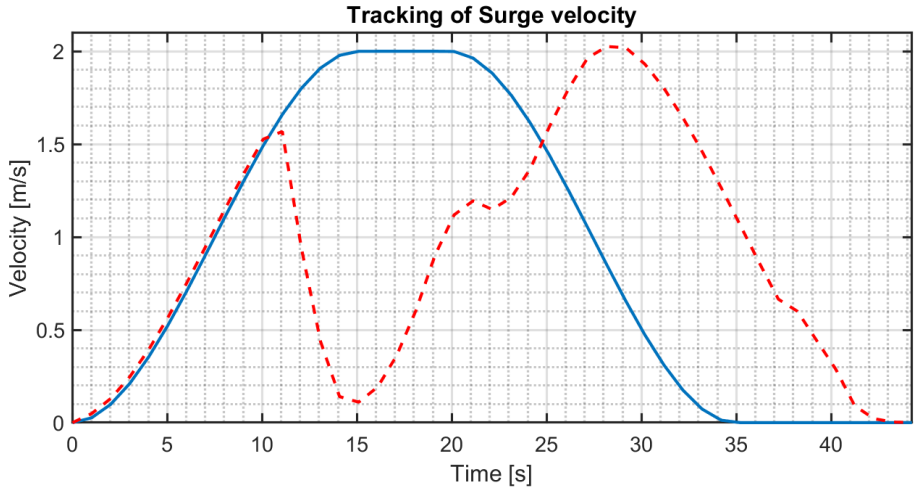

Figure 8.19: Case 2: Tracking of surge velocity.

The position and yaw deviations from the reference trajectory are shown on Fig 8.20. The yaw angle difference is due to the steering necessary to avoid the static obstacle and the high value of the position difference happened because the blocked passage caused a delay. The autonomous boat had to make a unplanned stop and consequently its position lagged behind the expected trajectory. Nonetheless, in the end the boat was able to reach the goal state with almost no deviation on position and yaw.

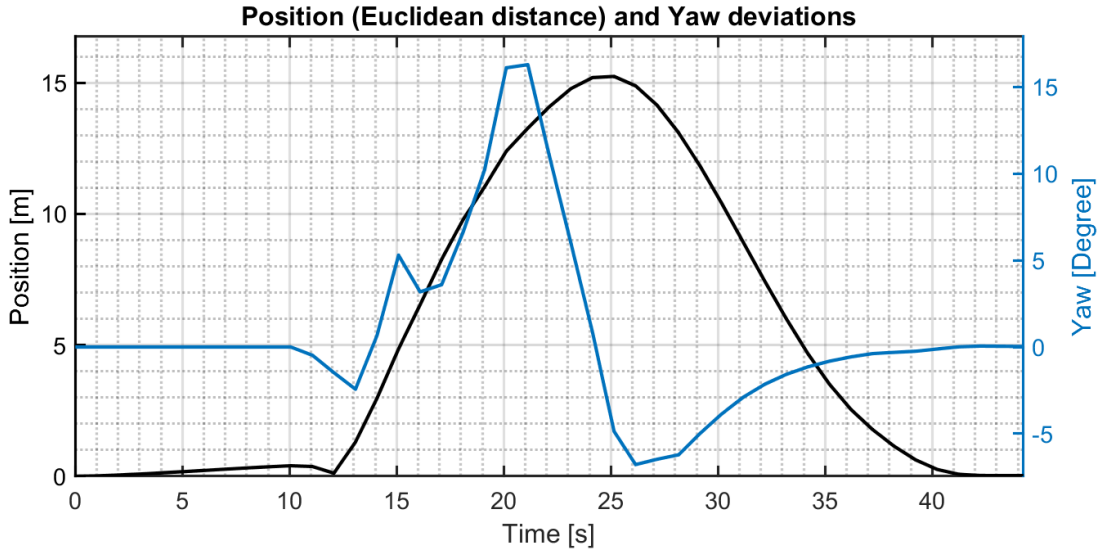

Figure 8.20: Case 2: Position and Yaw deviations.

The last step is to analyse the control inputs during the operation. Fig. 8.21 shows that the longitudinal thruster has a curve similar to that of the velocity, which is expected. The important remark about the control inputs is that in this case the NMPC controller tried to reduce the velocity at 10 seconds but could not find a feasible solution at 11 seconds. Consequently, the control input was replaced with a thrust in the opposite direction which caused the steep reduction in velocity observed between 11 and 15 seconds. This shows that the algorithm for dealing with infeasible solution is working as expected.

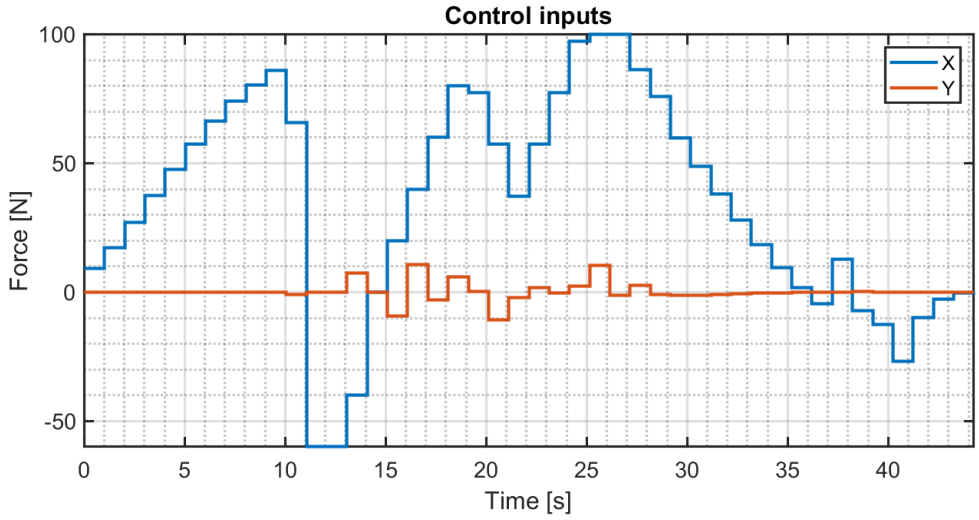

Figure 8.21: Case 2: Control inputs.

Chapter 9

Conclusion

The objective of this thesis was to apply a methodology able to replace human intervention in trajectory planning and motion control on vessels operating on urban canals. This task raised two questions: how to create a low cost trajectory that is feasible under the boat's kinodynamic constraints and how to track this trajectory safely.

In order to generate a global trajectory (path + time law), it was necessary to study how to find feasible paths for boats connecting the start and goal states and then associate this path to a time law. The path planner used was a Dubins curve based RRT*, which generated paths with unnecessary curves and therefore higher length than the optimal path. The solution for this issue was to apply a post-processing pruning to remove the waypoints causing the extra curves which yielded paths with more straight segments. These paths are near optimal and are acceptable for the given task since path deviations will happen for obstacle avoidance and there is no guarantee that the optimal path will be strictly followed.

The time law was generated by means of a smoothed trapezoidal velocity profile with cubic velocities during acceleration and deceleration and a constant cruise velocity. This curve was chosen because it reduces the boat's jerk which is more comfortable for passengers and also keeps a constant velocity during the operation.

Regarding the local motion controller, a NMPC controller was chosen since it can be used for trajectory tracking while considering state and control bounds at design stage, selecting only dynamically possible control actions. Furthermore, the controller is also able to consider constraints representing walls, static and dynamic obstacles such that the boat does not collide during locomotion.

The trajectory tracking performance was good with small deviations in angle and position, except when the path is blocked in which case the boat needs to wait for the obstacles to be cleared and then it can continue moving. More importantly, the boat was able to reach the goal state with near zero error.

In addition, the controller was able to guarantee the safety of the boat by avoiding collisions by either steering the boat or by halting its movement. In the example where the controller detected an infeasible solution and slowed down the boat, it was able to restart movement after the obstacle moved out of the way. One issue of the controller design is that it is not COLREGs compliant, with a violation present on Fig. 8.11. In this case, the boat should have steered to starboard instead of port.

To summarize, the methodology applied was able to accomplish the given tasks: creating feasible trajectories and tracking them while avoiding collisions. The reference trajectory considered limitations on the boat's curvature, velocity and acceleration. In turn, the

NMPC motion planner was able to reach the goal state without collisions and with near zero errors on the final pose.

9.1 Future Work

To improve the work done on this thesis, the following propositions could be implemented:

- If the operation of the vessel requires the optimal path, it is possible to implement a kinodynamic RRT*. In this case the steering function is an optimal control problem solver and the connection between two states can be done according to some cost function. This cost function could include not only length, but also clearance.
- The current NMPC implementation was done on MATLAB and its performance is not suitable for real time implementation. It would be ideal to implement it in a compiled programming language such as C++. There are open-source software such as CasADi and ACADO that can solve optimization problems, like NLP, with high efficiency.
- This thesis considered that all sensors are ideal and positions are perfectly known, which does not happen in the real world. It would be a good idea to implement algorithms to deal with sensor noise, such as nonlinear moving horizon state estimation (NMHE) and sensor fusion. This modification would make the motion controller more robust and able to deal with real working conditions.

Bibliography

[1] Jiajia Chen, Rui Zhang, Wei Han, Wuhua Jiang, Jinfang Hu, Xiaoshan Lu, Xingtao Liu, and Pan Zhao. «Path planning for autonomous vehicle based on a two-layered planning model in complex environment». In: *Journal of Advanced Transportation* 2020 (2020). bi-layer and pruning idea. ISSN: 20423195. DOI: 10.1155/2020/6649867 (cit. on pp. 3, 11, 12).

[2] Anete Vagale, Rachid Oucheikh, Robin T. Bye, Ottar L. Osen, and Thor I. Fossen. «Path planning and collision avoidance for autonomous surface vehicles I: a review». In: *Journal of Marine Science and Technology (Japan)* 26 (4 Dec. 2021). Definition of COLReg, path planning and trajectory planning. List of many path planning algorithms incuding sampling based, pp. 1292–1306. ISSN: 09484280. DOI: 10.1007/s00773-020-00787-6 (cit. on pp. 3, 5, 15).

[3] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136 (cit. on p. 3).

[4] Steven M. LaValle. «Rapidly-exploring random trees : a new tool for path planning». In: *The annual research report* (1998) (cit. on pp. 3, 8).

[5] Sertac Karaman and Emilio Frazzoli. «Sampling-based Algorithms for Optimal Motion Planning». In: (May 2011). Pseudocodes for RRT and RRT* algorithms. Paper where RRT* is introduced as well as its characteristics about convergence and optimality. URL: http://arxiv.org/abs/1105.1186 (cit. on pp. 4, 9).

[6] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. «Anytime motion planning using the RRT». In: 2011, pp. 1478–1483. ISBN: 9781612843865. DOI: 10.1109/ICRA.2011.5980479 (cit. on pp. 4, 11, 12).

[7] Dustin J. Webb and Jur Van Den Berg. «Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics». In: 2013, pp. 5054–5061. ISBN: 9781467356411. DOI: 10.1109/ICRA.2013.6631299 (cit. on pp. 4, 12).

[8] Chinmay Vilas Samak, Tanmay Vilas Samak, and Sivanathan Kandhasamy. «Control Strategies for Autonomous Vehicles». In: (Nov. 2020). URL: http://arxiv.org/abs/2011.08729 (cit. on p. 4).

[9] O. Khatib. «Real-time obstacle avoidance for manipulators and mobile robots». In: Institute of Electrical and Electronics Engineers Inc., 1985, pp. 500–505. ISBN: 0818606150. DOI: 10.1109/ROBOT.1985.1087247 (cit. on p. 4).

[10] D. Fox, W. Burgard, and S. Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977 (cit. on p. 4).

[11] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. «Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments». In: *IEEE Robotics and Automation Letters* 4 (4 Oct. 2019), pp. 4459–4466. ISSN: 23773766. DOI: 10.1109/LRA.2019.2929976 (cit. on pp. 5, 22, 30).

[12] Jitske de Vries, Elia Trevisan, Jules van der Toorn, Tuhin Das, Bruno Brito, and Javier Alonso-Mora. *Regulations Aware Motion Planning for Autonomous Surface Vessels in Urban Canals*. 2022. DOI: 10.48550/ARXIV.2202.12069. URL: https://arxiv.org/abs/2202.12069 (cit. on pp. 5, 30).

[13] Abhijeet Ravankar, Ankit A. Ravankar, Yukinori Kobayashi, Yohei Hoshino, and Chao Chung Peng. «Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges». In: *Sensors (Switzerland)* 18 (9 Sept. 2018). ISSN: 14248220. DOI: 10.3390/s18093170 (cit. on p. 11).

[14] Armando A. Neto, Douglas G. Macharet, and Mario F.M. Campos. «Feasible RRT-based path planning using seventh order Bézier curves». In: 2010, pp. 1445–1450. ISBN: 9781424466757. DOI: 10.1109/IROS.2010.5649145 (cit. on p. 11).

[15] Juan David Hernández, Mark Moll, Eduard Vidal, Marc Carreras, and Lydia E. Kavraki. «Planning feasible and safe paths online for autonomous underwater vehicles in unknown environments». In: vol. 2016-November. Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 1313–1320. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759217 (cit. on pp. 12, 41).

[16] I. Bae et al. «Self-Driving like a Human driver instead of a Robocar: Personalized comfortable driving experience for autonomous vehicles». In: (Jan. 2020). URL: http://arxiv.org/abs/2001.03908 (cit. on p. 15).

[17] Hrishikesh Dey, Rithika Ranadive, and Abhishek Chaudhari. «Real - Time Trajectory and Velocity Planning for Autonomous Vehicles». In: *International Journal of Engineering and Advanced Technology* 10 (5 June 2021). Use of trapezoidal and smoothed trapezoidal speed profiles for trajectory planning, pp. 439–448. DOI: 10.35940/ijeat.e2880.0610521 (cit. on pp. 16, 17).

[18] Thor I. Fossen. *Handbook of marine craft hydrodynamics and motion control*. Wiley, 2011, p. 596. ISBN: 9781119991496 (cit. on p. 20).

[19] Steven M Lavalle. *PLANNING ALGORITHMS*. URL: http://planning.cs.uiuc.edu/ (cit. on p. 24).

[20] Lars Grüne and Jürgen Pannek. *Communications and Control Engineering Nonlinear Model Predictive Control Theory and Algorithms Second Edition*. URL: http://www.springer.com/series/61 (cit. on p. 24).

[21] Hichem Salhi and Faouzi Bouani. «Chapter 7 - Practical Study of Derivative-Free Observer-Based Nonlinear Adaptive Predictive Control». In: *New Trends in Observer-Based Control*. Ed. by Olfa Boubaker, Quanmin Zhu, Magdi S. Mahmoud, José Ragot, Hamid Reza Karimi, and Jorge Dávila. Emerging Methodologies and Applications in Modelling. Academic Press, 2019, pp. 241–266. ISBN: 978-0-12-817038-0. DOI: https://doi.org/10.1016/B978-0-12-817038-0.00007-X. URL: https://www.sciencedirect.com/science/article/pii/B978012817038000007X (cit. on p. 25).

[22] Maria M. Seron. *Receding Horizon Control*. https://www-eng.newcastle.edu.au/eecs/cdsc/books/cce/Slides/RecedingHorizonControl.pdf. [Online; accessed 16-May-2022]. Sept. 2004 (cit. on p. 25).

[23] Li Dai, Yuanqing Xia, Mengyin Fu, and Magdi S. Mahmoud. «Discrete-Time Model Predictive Control». In: *Advances in Discrete Time Systems*. Ed. by Magdi S. Mahmoud. Rijeka: IntechOpen, 2012. Chap. 4. DOI: 10.5772/51122. URL: https://doi.org/10.5772/51122 (cit. on p. 25).

[24] Moritz Diehl. *Direct Methods*. https://www.imtek.de/professuren/systemtheorie/events/dateien/direct. [Online; accessed 16-May-2022]. n.d (cit. on pp. 25, 26).

[25] H.G. Bock and K.J. Plitt. «A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems*». In: *IFAC Proceedings Volumes* 17.2 (1984). 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984, pp. 1603–1608. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)61205-9. URL: https://www.sciencedirect.com/science/article/pii/S1474667017612059 (cit. on p. 26).

[26] Kristoffer Bergman. «Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments». PhD thesis. Linköping University Electronic Press, 2021. ISBN: 9789179296773. DOI: 10.3384/diss.diva-174175. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-174175 (cit. on p. 27).

[27] Wei Wang, Tixiao Shan, Pietro Leoni, David Fernandez-Gutierrez, Drew Meyers, Carlo Ratti, and Daniela Rus. «Roboat II: A Novel Autonomous Surface Vessel for Urban Environments». In: (July 2020). URL: http://arxiv.org/abs/2007.10220 (cit. on p. 35).