Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Magnus Knædal

Autonomous Path Planning and Maneuvering of a Surface Vessel

Master's thesis in Engineering & ICT, Marine Cybernetics
Supervisor: Roger Skjetne
June 2020

The ReVolt model scale ship. Photo: Simen Sem Øvereng (2019).

NTNU
Norwegian University of
Science and Technology

Magnus Knædal

Autonomous Path Planning and Maneuvering of a Surface Vessel

Master thesis
Trondheim, June 8, 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Supervisor: Professor Roger Skjetne
Co-advisors: PhD Tom Arne Pedersen

Norwegian University of
Science and Technology

Preface

This thesis constitutes my master's degree in Marine Cybernetics at the Norwegian University of Science and Technology (NTNU), during spring 2020. It is a summary of my findings and proposal of how stepwise planning can be performed for an autonomous surface vessel. The thesis is mainly inspired by the work done on maneuvering control design and stepwise path generation by Professor Roger Skjetne at NTNU.

The work has mainly consisted of theoretical design and development, where the findings have been implemented and tested using the Robotics Operating System (ROS) and the ReVolt model scale ship by DNV GL as a test case. The reader is assumed to have prior knowledge in nonlinear systems and how to solve these kinds of systems numerically, as well as knowledge in mathematical optimization. Knowledge about graph and curve theory, path planning, and in particular, sampling-based planning is beneficial, but the necessary knowledge will be represented.

As for the majority of people, the spring of 2020 has been a weird and challenging one in many ways. COVID-19 has had a significant impact on our everyday life and also led to uncertainty and limitations related to my master's degree. The original plan was to perform extensive testing through both simulations and physical experiments on ReVolt. However, due to the circumstances, most of the work has been limited to simulation, except for a last-minute experiment conducted on the real ReVolt.

Acknowledgments

Research is never a solitary task. I want to thank my supervisor Roger Skjetne for valuable guidance whenever needed. Also, I would like to thank my co-supervisor, Tom Arne Pedersen, at DNV GL for assisting me with ReVolt, providing necessary equipment, and answering any questions I would have in mind. Further, I want to thank Knut Turøy and Simen Sem Øvereng for helping me carry out the experiment. Thanks to my dearest friends Magnus Kunnas and Jonas Åsnes Sagild for help with structuring and proofreading of the thesis.

I would like to thank fellow students for providing both a stimulating work environment and excellent social surroundings through my whole degree. I am left with invaluable knowledge and fantastic friends that I am deeply grateful for. Last but not least, I would like to thank my mother, Ragnhild Oanes, grandfather Eivind Oanes, and Carl Fredrik Wendt for endless love and support. They have provided me with the freedom to choose my own way in life, and I would like to dedicate this thesis partially as a token of my gratitude.

Magnus Knædal,
Trondheim, June 8, 2020

Abstract

This thesis proposes an intelligent guidance concept for a surface vessel moving from initial to target waypoint. The problem is faced in a stepwise manner to facilitate real-time execution and the ability to replan online. For convenience, we propose to divide the complete system into four submodules: the guidance, navigation, measurement, and control system.

A new way to generate a stepwise, smooth, and continuous path in the horizontal plane using the Bézier curve and quadratic optimization is developed. Also, a pragmatic approach is proposed. It is shown how the path generators, together with a dynamic assignment (which makes up the guidance system), can produce the necessary signals for the maneuvering controller in use. A control law is designed using a nonlinear adaptive backstepping technique that continuously produces the desired forces to be applied to the vessel. The forces are then distributed to the actuators by a thrust allocation algorithm provided by DNV GL.

The navigation system consists of a global low-resolution path planner working together with a local dynamic high-resolution path planner. The A* algorithm operating on a Voronoi roadmap constitutes the global planner. It generates a safe path efficiently. A rapidly exploring random tree (RRT) algorithm combining useful aspects of different RRT variants proposed in the literature serves as the local planner. It is shown how RRT can be used to avoid dynamical obstacles and gradually replan in a stepwise manner. However, a question is raised if the current solution is computationally efficient enough to tackle real-life situations.

The ReVolt model scale ship by DNV GL is used as a test case. A considerable amount of simulations are done to verify the performance of each designed subsystem. The simulations provide a sound basis for experimental testing, resulting in a sea trial where the guidance and control systems are tested on the real ReVolt.

Sammendrag

Denne oppgaven tar for seg et intelligent veiledningskonsept for en overflatefarkost som beveger seg fra start- til endelig veipunkt. Problemet blir løst stegvis for å legge til rette for utførelse i sanntid og for å kunne planlegge på nytt underveis. For enkelhets skyld foreslår vi å dele inn systemet i fire undermoduler: veiledning-, navigasjon-, måling- og kontrollsystem.

En ny måte å generere en stegvis, jevn og kontinuerlig bane i horisontalplanet ved bruk av Bézier-kurven og kvadratisk optimering er utviklet. En pragmatisk løsning er også foreslått. Det vises hvordan banegeneratorene sammen med en dynamisk oppgave (som utgjør veiledningssystemet), kan produsere de nødvendige signalene for regulatoren i bruk. En regulator designet ved bruk av en ikke-lineær adaptiv *backstepping*-teknikk, beregner kontinuerlig kreftene som skal påføres farkosten. De ønskede kreftene blir deretter fordelt til aktuatorene ved hjelp av en allokeringsalgoritme levert av DNV GL.

Navigasjonssystemet består av en global baneplanlegger av lav oppløsning som jobber sammen med en lokal dynamisk baneplanlegger av høy oppløsning. Den globale baneplanleggeren består av A*-algoritmen som opererer på et Voronoi-veikart som deler opp arbeidsområdet. Den lokale planleggeren er en *rapidly exploring random tree* (RRT)-algoritme som kombinerer nyttige aspekter av ulike RRT-varianter foreslått i litteraturen. Det er vist hvordan RRT kan brukes for å unngå dynamiske hindringer og stegvis planlegge en ny bane. Det stilles spørsmålstegn til om den nåværende løsningen er effektiv nok til å takle virkelige situasjoner.

Modellbåten ReVolt, laget av DNV GL, brukes som casestudie. Flere simuleringer blir gjort for å verifisere prestasjonen til hvert delsystem. Simuleringene gir et godt grunnlag for eksperimentell testing og resulterer i et eksperiment hvor veilednings- og kontrollsystemene testes på den virkelige ReVolt.

Table of Contents

List of Tables

List of Figures

Preliminaries

Mathematical Definitions, Notations, and Abbreviations

Table 1: Mathematical abbreviations.

Symbol	Meaning
$\in$	*element of*
$\forall$	*for all*
$\triangleq$	*defined as*
$\preceq$	*component wise inequalities*
$\subset$	*is a proper subset of*
$\cup$	*set union*
$\setminus$	*set minus*
$\emptyset$	*the empty set*
$\wedge$	*logical and*
$\Longrightarrow$	*implies*
$\leftarrow$	*set to*
$\rightarrow$	*goes to* or *converge to*

- All vectors are written in boldface, for example $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{p}$, while scalars are not.

- Total time derivatives of a function $x(t)$ are denoted $\dot{x}, \ddot{x}, x^{(3)}, \ldots, x^{(n)}$. A superscript with an argument variable will denote partial differentiation with respect to that argument, that is $\alpha^t(x, \theta, t) \triangleq \frac{\partial \alpha}{\partial t}$, $\alpha^{x^2}(x, \theta, t) \triangleq \frac{\partial \alpha}{\partial x^2}$, and $\alpha^{\theta^n}(x, \theta, t) \triangleq \frac{\partial \alpha}{\partial \theta^n}$, etc.

- An index set of positive natural numbers, $\mathbb{N}_{>0}$, from 1 to n is denoted $\mathcal{I}^n = \{1, 2, 3, \ldots, n\}$.

- The p-norm of a vector is defined as:

$$|\boldsymbol{x}|_p \triangleq \Big(\sum_{n}^{i=1} |x_i|^p \Big)^{1/p},$$

  where the most commonly used is the 2-norm, or the Euclidean norm, denoted $|\boldsymbol{x}| \triangleq |\boldsymbol{x}|_2 = (\boldsymbol{x}^\top \boldsymbol{x})^{1/2}$. For a scalar, the 2-norm reduces to the absolute value.

- A diagonal matrix is often written as $\text{diag}(a, b, c, \dots)$, and the identity matrix is simply written $\boldsymbol{I}$ where its dimension should be clear from the context.
- A matrix $\boldsymbol{R} \in SO(n)$, is part of a *special orthogonal group of order n*:

$$SO(n) = \{\boldsymbol{R} | \boldsymbol{R} \in \mathbb{R}^{n \times n},\ \boldsymbol{R} \text{ is orthogonal and } \det(\boldsymbol{R}) = 1\}.$$

  A matrix $\boldsymbol{R} \in SO(n)$ satisfies:

$$\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}, \quad \boldsymbol{R}^{-1} = \boldsymbol{R}^\top, \quad \det(\boldsymbol{R}) = 1.$$

- An n-dimensional body in $\mathbb{R}^n$, which can translate and rotate is part of the *special Euclidean group* $SE(n) = \mathbb{R}^n \times SO(n)$.
- A matrix $\boldsymbol{S} \in SS(n)$, that is the set of skew-symmetric matrices of order n, is said to be skew-symmetric if:

$$\boldsymbol{S} = -\boldsymbol{S}^\top.$$

  This implies that the off-diagonal elements of $\boldsymbol{S}$ satisfy $s_{ij} = -s_{ji}$ for $i \neq j$ while the diagonal elements are zero (Fossen, 2011d).
- In control design, the subscript 'd' as in $x_d(t)$ or $y_d(t)$ means 'desired'. It will always be used for a varying desired function.
- A uniform sample x from a domain $\mathcal{X}$ is denoted $x \sim \mathcal{U}(\mathcal{X})$.

Linguistic Abbreviations

Table 2: Linguistic abbreviations and acronyms.

Symbol	Meaning
GNMC	*Guidance, Navigation, Measurement, and Control*
DOF	*Degree Of Freedom*
CO	*Common Origin*
CG	*Center of Gravity*
ASV	*Autonomous Surface Vessel*
PC	*Parametric Continuity*
NED	*North East Down*
UAV	*Unmanned Aerial Vehicle*
DP	*Dynamic Positioning*

OGM	*Occupancy Grid Map*
RRT	*Rapidly exploring Random Tree*
RT-RRT	*Real-Time RRT*
PRM	*Probabilistic RoadMap*
VD	*Voronoi Diagram*
UGES	*Uniformly Globally Exponentially Stable*
CLF	*Control Lyapunov Function*
OSP	*Open Simulation Platform*
OS	*Operating System*
CPU	*Central Processing Unit*
GPS	*Global Positioning System*
OS	*Operating System*
AIS	*Automatic Identification System*
IMU	*Inertial Measurement Unit*
EKF	*Extended Kalman Filter*
ROS	*Robotic Operating System*
GUI	*Graphical User Interface*
COLREG	*The international regulations for preventing collisions at sea*
NTNU	*Norwegian University of Science and Technology*
s.t.	*subject to* or *such that*
e.g.	*exempli gratia* (“*for example*”)
i.e.	*id est* (“*in other words*”)
etc.	*et cetera* (“*and so on*”)
Q.E.D	*Quod Erat Demonstrandum* (“*which was to be demonstrated*”)

Chapter 1

Introduction

1.1 Background and Motivation

The purpose of a vessel is to maneuver from one place to another, except for station-keeping. To be able to maneuver, the appropriate planning and signals for where to go next need to be produced. For centuries, humans have served as the guidance, navigation, and control system for the vessel. However, as the digital revolution emerged during the second half of the 20th century, computers have gradually prevailed. As for all sectors, including the maritime one, the need for AI and autonomy for the industry to remain relevant is growing. Consequently, the development of highly sophisticated software modules is a necessity (Liu et al., 2016).

In the marine control research community, the complete motion control system is traditionally divided into subsystems working in cascade (see Fossen (2011b) for an overview). The cascaded structure simplifies the stability analysis and makes it easier to verify the performance of each subsystem. For an autonomous surface vessel (ASV), we propose to divide the complete motion control system into:

- The *guidance system*: -used to continuously compute the reference position, velocity, and attitude of the marine craft to be used by the control system.
- The *navigation system*: -used to determine the waypoints as well as the desired speed in the operating region.
- The *measurement system*: -used to determine the vessel's position, course, distance traveled, and in some cases, velocity and acceleration.
- The *control system*: -used to continuously determine the necessary control forces and moments to be applied to the vessel to satisfy the given control objective.

A visualization of the closed-loop guidance, navigation, measurement, and control (GNMC) system is given in Figure 1.1. In recent years, there has been devoted a tremendous amount of research, driven by the economic benefits of autonomy at sea,

in developing different parts of the GNMC system. However, there are still many challenges to be solved. The system must be able to perform safe maneuvering in real-time in case of unexpected obstacles appearing on the way. This situation sets some common essential requirements for each subsystem in the GNMC system; they must be computationally efficient, reliable, and robust.

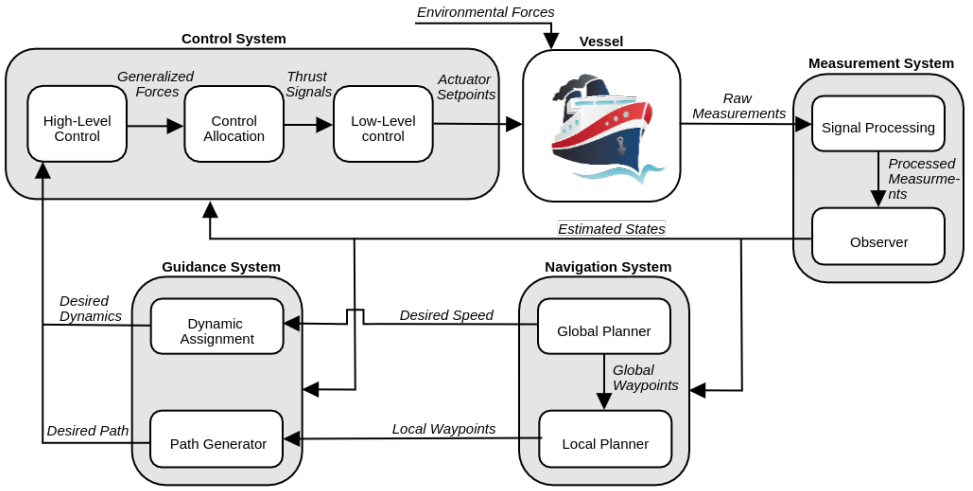


Figure 1.1: A closed loop GNMC system for a marine craft.

1.2 Objectives

The objective of this thesis is to develop and implement an intelligent guidance concept for an ASV, moving from an initial to a target point. In order to handle dynamic obstacles, the problem is faced in a stepwise manner. The planner should facilitate real-time system performance; it must be computationally efficient. A global low-resolution planner is combined with a local dynamic method to achieve a high-resolution reactive path planner sensitive to local ambient conditions. The overall objective includes:

- Consideration of global and local mapping and partitioning of the operation area.
- Design of a global and a local path planner, and the integration between them.
- Propose how to generate a feasible and smooth path online, efficiently.
- Design of a motion control system, including high-level control and control allocation.

1.3 Scope of Work

The scope of work in this thesis includes:

- Perform a background and literature review to provide information and relevant references on:
  - The concept vessel ReVolt by DNV GL.
  - Autonomy for marine control systems.
  - Maneuvering-based control methods, including the hybrid path parameterization method.
  - Curve theory, especially theory for generating Bézier curves.
  - Guidance models applicable for sampling-based path planning, e.g. cost maps.
  - Sampling-based path planning, especially Rapidly Exploring Random.
  - Waypoint-based path generation and iterative (stepwise) path generation.
- Establish a simplified dynamic vessel model for the ReVolt to use as a case when considering constraints and physical properties in the path planning and generation problem.
- Formulate the guidance and control problem, including the definition of a case study, description of the setup, vessel model/descriptions, operation workspace, and specific assumptions and delimitations.
- Define a relevant discrete topologically organized map of the workspace for the marine vessel by selecting an appropriate method. Investigate and propose an algorithm/method to generate waypoint candidates in the map, using sampling-based path planning according to the defined guidance and control problem.
- Investigate and propose an algorithm/method to generate a path in the horizontal plane for a marine vessel, using Bézier curves as basis functions. Assume a collision-free straight-line corridor between consecutive waypoints, as well as constraints on maximum curvature (in physical space).
- Design and implement a maneuvering-based control law for following a path. Implement and verify it in simulations.
- Integrate the guidance model, path planner, path generation, and maneuvering controller in a seamless manner so that autonomous guidance and control can be achieved. Perform simulation or experimentation studies to verify the resulting performance. Critically analyze and discuss the results.

1.4 Contributions

To the best of the author's knowledge, this thesis has contributed with a new way to generate a stepwise, smooth, and continuous path in the horizontal plane using the Bézier curve and quadratic optimization. A modified established method for global path planning is integrated with a proposed real-time RRT-algorithm serving as a local planner. The RRT-algorithm is optimized towards an ASV operating on occupancy grid maps. Together with existing methods, a complete and computationally efficient system for an autonomous guidance concept for an ASV has been proposed. The system has been implemented using the Robotic Operating System (ROS) and contributed to an additional control mode being added to the concept vessel ReVolt's control system. A GUI using a Qt-based framework, together with RViz, has been made to control the system easily.

1.5 Outline of Thesis

The thesis is divided into ten main chapters. In Chapter 2, the necessary theoretical background knowledge, as well as important concepts, are represented. In Chapter 3, the problem to be solved is formulated, including functional inputs/outputs and objectives of the different subsystems treated. Here we also state the simplifications and assumptions made when approaching the problem. In Chapters 4 to 6, the design and proposal answering the objectives are done. The experimental platform used and implementation details are given in Chapter 7. In Chapters 8 and 9, we represent the main results, and at last, a critical assessment and conclusion are given in Chapters 10 and 11.

Chapter 2

Background Knowledge and Literature Review

In this chapter, the necessary theoretical background knowledge, as well as important concepts for this thesis, are represented. Sections 2.1 to 2.3 is heavily based on the theory represented in the author's specialization project (Knædal, 2019).

2.1 Vessel Model and Description

2.1.1 Notation and Reference Frames

The notation in Table 2.1, defined by SNAME (1950) will be used. The geographical reference frame North-East-Down (NED) and the body-fixed reference frame are used for analysis. NED is chosen as a tangent plane to the surface of the earth, and positions within the frame are denoted $\{n\} = (x_n, y_n, z_n)$, where the x_n-axis points towards true north, the y_n-axis points east, and the z_n-axis points downwards. The body-fixed reference frame is denoted $\{b\} = (x_b, y_b, z_b)$, where the x_b-axis points in the longitudinal direction of the vessel, the y_b-axis in the transverse direction of the vessel, and the z_b-axis points in the direction normal to the x_b-y_b plane. See Fossen (2011d) for a closer explanation on kinematics and notation.

Table 2.1: Notation of SNAME (1950) for marine vessels.

Degree of freedom	Position in Euler coordinates	Linear and angular velocities	Forces and moments
Surge	x	u	X
Sway	y	v	Y
Yaw	ψ	r	N

2.1.2 Vessel Model

A simplified control design model, based on the ReVolt model scale ship, is used as a case when considering dynamic constraints and physical properties. A fully actuated three degrees-of-freedom (DOF) model operating in the horizontal plane will be used, thus neglecting motion in roll, pitch, and heave. According to Fossen (2011e), a 3 DOF equation of motion of a marine vessel can be represented as:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{R}(\psi)\boldsymbol{\nu} \\ \boldsymbol{M}\dot{\boldsymbol{\nu}} &= -\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{\tau} + \boldsymbol{R}(\psi)^\top \boldsymbol{b}, \end{aligned} \tag{2.1}$$

where $\boldsymbol{\nu} = [u, v, r]^\top$ is the generalized velocity of the vessel in $\{b\}$ and $\boldsymbol{\eta} = [\boldsymbol{p}, \psi]^\top = [x, y, \psi]^\top$ is the generalized position in $\{n\}$. Further, $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{C}(\boldsymbol{\nu})$ is the Coriolis and centripetal matrix, $\boldsymbol{D}(\boldsymbol{\nu})$ is the damping matrix and $\boldsymbol{\tau}$ is the forces acting on the vessel. The vector $\boldsymbol{b} = [b_1, b_1, b_3]^\top$, is an unknown constant (or slowly varying) bias expressed in $\{n\}$, accounting for model uncertainties.

$\boldsymbol{R}(\psi) \in SO(3)$ is the 3 DOF rotation matrix with the property that $\dot{\boldsymbol{R}}(\psi) = \boldsymbol{R}(\psi)\boldsymbol{S}(r)$ where $\boldsymbol{S}(r) \in SS(3)$ is skew-symmetric. Thus:

$$\boldsymbol{R}(\psi) \triangleq \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{S}(r) \triangleq \begin{bmatrix} 0 & -r & 0 \\ r & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{2.2}$$

The force vector $\boldsymbol{\tau}$ contains the thrust forces acting on the vessel:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_X \\ \tau_Y \\ \tau_N \end{bmatrix} = \begin{bmatrix} F_X \\ F_Y \\ l_x F_Y - l_y F_x \end{bmatrix}, \tag{2.3}$$

where F_X and F_Y are the forces in the respective indexed directions and l_x and l_y are the arms from which τ_N are acting on.

Further, we assume that the ship is symmetric and has homogeneous mass distribution about the x_b-z_b plane. This results in a decoupling of the surge motion from sway and yaw and implies that the products of inertia $I_{xy} = I_{yz} = 0$. By further assuming that the common origin (CO) of $\{b\}$ coincides with the ships' center of gravity (CG), results in $x_g = y_g = 0$, where x_g and y_g are the distance from CO to CG in respective directions. The system inertia matrix $\boldsymbol{M} = \boldsymbol{M}^\top > 0$ includes both rigid-body and added mass terms, and can then be expressed as:

$$\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & I_z - N_{\dot{r}} \end{bmatrix}, \tag{2.4}$$

where m is the mass of the vessel, I_z is the moment of inertia around the z_b-axis, and the rest of the terms comes from the added mass contributions caused by the acceleration indicated by the subscripts. The skew-symmetric Coriolis-centripetal matrix also includes both rigid-body and added mass terms, and can then be expressed as:

$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -mv + Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & mu - X_{\dot{u}}u \\ mv - Y_{\dot{v}}v - Y_{\dot{r}}r & -mu + X_{\dot{u}}u & 0 \end{bmatrix}. \quad (2.5)$$

The damping matrix $\boldsymbol{D}(\boldsymbol{\nu})$ is constructed by one linear and one nonlinear part. For a low-speed vessel such as ReVolt, the linear damping terms is dominating, such that:

$$\boldsymbol{D}(\boldsymbol{\nu}) = \boldsymbol{D}_L = -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} > 0, \quad (2.6)$$

where each term represents hydrodynamic damping forces caused by the velocity indicated by the subscripts.

2.1.3 Maneuverability and Vehicle Characteristics

Maneuverability is defined as the capability of the craft to carry out specific maneuvers (Fossen, 2011f). The maneuverability of the ship depends on several factors, such as water depth, environmental forces, hydrodynamic derivatives, and, most importantly, the dynamical constraints of the vessel. For maneuvering of a marine craft, the relationship between *heading*, *course*, and *sideslip*, depicted in Figure 2.1, is important. The velocity vector is given as:

$$U = \sqrt{u^2 + v^2}. \quad (2.7)$$

For a marine craft moving around, the velocity vector is not necessarily pointing in the same direction as the ship's heading. The course angle χ is the angle from the x_n-axis to the velocity vector of the craft. The heading ψ is defined to be the angle between the x_n-axis in $\{n\}$ to the x_b-axis in $\{b\}$. The difference between them is the sideslip angle β. Thus, we have the relation:

$$\chi = \psi + \beta. \quad (2.8)$$

When designing a path for the vessel to follow, one needs to take into account that the magnitude and orientation of the velocity vector cannot change arbitrarily fast. For a certain velocity U, there is a limitation on the maximum angular velocity the vessel can perform. The lower the speed, the sharper turns the vessel can perform. The maximum angular speed ω_{max} can be estimated as a function of the vehicle speed U: $\omega_{max}(U) \geq 0$. The angular speed of the velocity vector is given as $\dot{\chi} = \omega$. We know that the inequality:

$$|\dot{\chi}| \leq \omega_{max}(U), \tag{2.9}$$

must hold to not violate the angular speed constraint. Having the relation $U = \omega R$, where R is the *steady-state turning radius*, and the inequality in Equation (2.9), we obtain the relation:

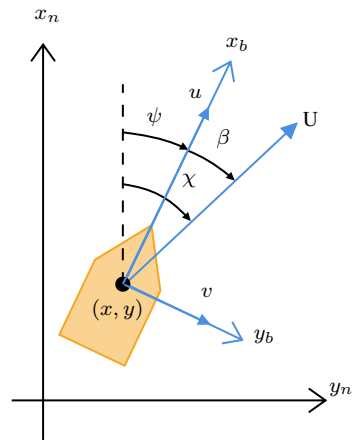


Figure 2.1: The relationship between ψ, χ, and β.

$$U \leq \omega_{max}(U)R. \tag{2.10}$$

This is an implicit relation between U and ω_{max}. By further using the relation from Equation (2.20), we get:

$$U \leq \frac{\omega_{max}(U)}{\kappa_{max}}. \tag{2.11}$$

The steady-state turning radius represents the smallest turn the vessel can perform at a constant surge speed. A maneuvering test such as the *Turning Circle Trial* can be used to obtain the steady-state turning radius. See Gertler and Hagen (1960) for a detailed explanation of the test. The turning radius then gives the curvature κ, by Equation (2.20).

2.2 Curve Theory

2.2.1 Path Parameterization

A curve is the image of a continuous function from an interval to a topological space. The function that defines the curve is called a *parameterization*, and the curve is a parametric curve. In this thesis, we look at planar curves and important properties for them concerning path planning and -generation. A planar parametrized curve is a path generated in a two-dimensional plane that is traced out by a point $(x(s), y(s))$ as the parameter $s \in \mathbb{R}$, ranges over an interval $I = [a, b]$. The position of a point on

the path can be represented as $\boldsymbol{p}_d(s) = [x_d(s), y_d(s)]^\top$, where subscript d denotes "desired". Thus, the path is a one dimensional manifold that can be defined as in Skjetne (2005) by the set:

$$\mathcal{P} \triangleq \{\boldsymbol{p} \in \mathbb{R}^2 \ : \ \exists s \in \mathbb{R} \text{ s.t. } \boldsymbol{p} = \boldsymbol{p}_d(s)\}. \tag{2.12}$$

The path parameter s is usually restricted to a specific interval $s \in [s_0, s_1]$.

For path planning, it is often desired to generate complex shapes due to, e.g., collision avoidance. To do this, it is common practice to implement $\boldsymbol{p}_d(s)$ as a piecewise parametric curve achieved by stitching together several curve segments. This is referred to as *hybrid path parameterization*. From Skjetne (2005), a planar path containing several curve segments in a one-dimensional manifold can be defined as:

$$\mathcal{P} \triangleq \{\boldsymbol{p} \in \mathbb{R}^2 \ : \ \exists i \in \mathcal{I} \text{ and } \theta \in [0, 1) \text{ s.t. } \boldsymbol{p} = \boldsymbol{p}_d(i, \theta)\}, \tag{2.13}$$

where $\mathcal{I}^m = \{1, 2 \ldots, m\}$ is the the set of m segment indices, and each point along the path is uniquely determined by a pair $(i, \theta) \in \mathcal{I}^m \times [0, 1)$. As stated in Lekkas (2014), this is convenient because it reduces the functional complexity of the curve, and, in turn, the computational effort for generating it. However, in return, one needs to consider the transition between the subpaths. In this thesis, the hybrid parameterization that is used to express a two-dimensional planar curve is expressed as:

$$\boldsymbol{p}_d(i, \theta) = \begin{bmatrix} x_d(i, \theta) \\ y_d(i, \theta) \end{bmatrix}, \quad \theta \in [0, 1], \ i \in \mathcal{I}^m, \tag{2.14}$$

with the path tangential angle defined as:

$$\gamma_d(i, \theta) = \text{atan2}(y_d^\theta(i, \theta), x_d^\theta(i, \theta)). \tag{2.15}$$

where atan2(y, x) is the four-quadrant version of arctan(y/x).

The pair $(i, \theta) \in \mathcal{I}^m \times [0, 1)$ does not conform to a continuous parameterization of a parameter $s \in \mathbb{R}$. However, this can be achieved by the following mapping:

$$\begin{aligned} i = g(s) &\triangleq \lfloor s \rfloor + 1, \\ \theta = h(s) &\triangleq s - \lfloor s \rfloor, \end{aligned} \tag{2.16}$$

where $\lfloor \cdot \rfloor$ is the floor operation. Then we achieve:

$$\boldsymbol{p}_d(s) = \boldsymbol{p}_d(g(s), h(s)) = \boldsymbol{p}_d(i, \theta). \tag{2.17}$$

Reparameterization

A curve can have indefinitely many reparameterizations. Let us consider a reparameterization. Let $I \subset \mathbb{R}$ and $J \subset \mathbb{R}$ be two intervals, and $\boldsymbol{p}$ be parametrized by $I \mapsto \mathbb{R}^2$.

Let h be a continuous function that maps I to J. Then $\bar{\boldsymbol{p}} \triangleq \boldsymbol{p} \circ h$ is a *reparameterization* of $\boldsymbol{p}$ by h. That is, for $s = h(\phi)$ we get the reparametrized curve (Skjetne, 2019):

$$\bar{\boldsymbol{p}}(\phi) = \boldsymbol{p}(h(\phi)). \tag{2.18}$$

Further assume that h is differentiable and monotonically increasing, such that $h^{\phi}(\phi) > 0$ for all $\phi \in J$, where $h(J) = I$. Then it follows that $\bar{\boldsymbol{p}}$ and $\boldsymbol{p}$ traces out the same path, but at different path speeds. Since h is bijective, the inverse mapping $s = h^{-1}(\phi) \triangleq g(s)$ will also exist.

An especially interesting reparameterization is done by using the arc length. A continuously differentiable curve $\boldsymbol{p}$ arbitrarily parametrized by a variable s, can be reparametrized by $s = h(l)$, where l is defined by Equation (2.19). Then we say that the curve is *arc length parametrized*. Arc length parameterization have the property of having unit speed in the direction of the tangent $\boldsymbol{p}^s(h(l))$.

2.2.2 Path Evaluation Criteria

To evaluate what constitutes a "better" path for a path-following motion control scenario, it is necessary to introduce a set of evaluation criteria. These criteria can help the designer in making a qualified decision when it comes to deciding what the better one is. These criteria are based on Lekkas (2014).

Arc Length

The arc length between two points a and b on a curve in the plane is the distance a parameter has to travel along the curve moving from one point to the other. Deducted from Pythagoras' theorem, the arc length is given by:

$$L = \int_a^b |\boldsymbol{p}_d^s(\tau)| \, d\tau = \int_a^b \sqrt{x_d^s(\tau)^2 + y_d^s(\tau)^2} \, d\tau. \tag{2.19}$$

For many path-following scenarios, the goal is to minimize the traveling distance between two points with respect to a set of constraints, e.g., clearance constraint for obstacles.

Path Curvature

Given a planar curve $\boldsymbol{p}_d$ and a value s, there exists a unique circle which approximates the curve near the point $\boldsymbol{p}_d(s)$. This circle is named the *osculating circle* at that point. See Figure 2.2 for a visualization. The radius $R(s)$ of the osculating circle at the given point $\boldsymbol{p}_d(s)$ is then defined to be the reciprocal of the curvature, such that:

$$R(s) \triangleq \frac{1}{\kappa(s)}, \tag{2.20}$$

where $\kappa(s)$ is the curvature at s along the path. Note that $\kappa(s)$ and $R(s)$ is defined with respect to arc length parameterization, such that the curvature has SI unit m^{-1}. See Section 2.2.1. In short, the curvature is a measure of how quickly the path changes direction at the point s. For an arbitrary general parameterization, the curvature can be found as (Goldman, 2005):

$$\kappa(s) = \frac{\left|\boldsymbol{p}_d^s(s) \times \boldsymbol{p}_d^{s^2}(s)\right|}{\left|\boldsymbol{p}_d^s(s)\right|^3} = \frac{\left|x_d^s(s) y_d^{s^2}(s) - x_d^{s^2}(s) y_d^s(s)\right|}{\left(x_d^s(s)^2 + y_d^s(s)^2\right)^{3/2}}. \tag{2.21}$$

It can be useful to know in which direction the path is turning/curving. A signed version of the curvature is given by:

$$\mathcal{H}(s) = \frac{\boldsymbol{p}_d^s(s) \times \boldsymbol{p}_d^{s^2}(s)}{\left|\boldsymbol{p}_d^s(s)\right|^3} = \frac{x_d^s(s) y_d^{s^2}(s) - x_d^{s^2}(s) y_d^s(s)}{\left(x_d^s(s)^2 + y_d^s(s)^2\right)^{3/2}}. \tag{2.22}$$

The sign will indicate the turning direction. It will rotate counter-clockwise if it is positive and rotate clockwise when it is negative.

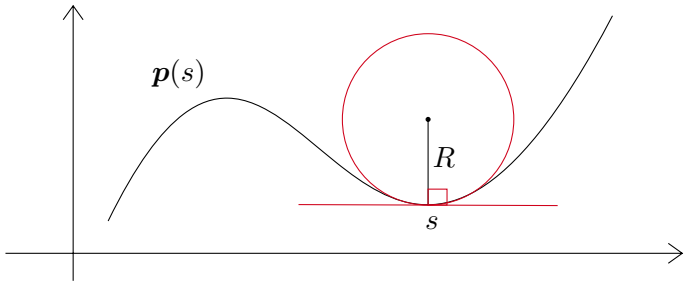


Figure 2.2: The osculating circle at a point s on the curve $\boldsymbol{p}(s)$.

Rate of Change in Curvature

If the third derivative exists at a point, we may also calculate the rate of change in curvature, $\tau(s)$[1]. This corresponds to the rate of change of the curve's osculating circle. The rate of change in curvature is a measure of how rapidly the curvature

[1]Note that τ is often used as symbol for "torsion" of a 3D space curve, sometimes also called the "second curvature" (Kreyszig et al., 2011). The rate of change in curvature for a 2D curve is not the same as torsion.

changes at a given point s. By differentiating Equation (2.22) one gets:

$$\tau(s) = \frac{x_d^s(s)y_d^{s^3}(s) - x_d^{s^3}(s)y_d^s(s)}{\left(x_d^s(s)^2 + y_d^s(s)^2\right)^{3/2}} - \frac{3\left(x_d^s(s)y_d^{s^2}(s) - x_d^{s^2}(s)y_d^s(s)\right)\left(2x_d^s(s)x_d^{s^2}(s) + y_d^{s^2}(s)y_d^s(s)\right)}{2\left(x_d^s(s)^2 + y_d^s(s)^2\right)^{5/2}}. \tag{2.23}$$

Both curvature and rate of change in curvature are independent of the parameterization of the curve and are Euclidean invariants, i.e., they do not change under rigid body motions of the curve.

Smoothness & Parametric Continuity

For a vehicle that needs to follow a path, two of the most fundamental requirements for the curve is continuity and smoothness. These criteria are directly related to the vehicle's dynamic constraints. A real function that can be represented by a curve in the Cartesian plane is continuous on an interval $[a, b]$ if, roughly speaking, the graph is a single unbroken curve and is defined at every point on that interval (Speck, 2014). In the context of smoothness for a planar parameterization, there are two different notions to describe the path smoothness, namely *geometric* and *parametric continuity*. In this thesis, only parametric continuity is exploited. For a review of geometric continuity, refer to Barsky and DeRose (1984).

Parametric continuity (PC) is a form of continuity that imposes restrictions on the derivatives of the parameterization. PC does not reflect the smoothness of the geometrical view of the curve, but rather the parameterization. This means that the curve may appear continuous, but does not necessarily have the property of being parametric continuous. For the topic studied here, there is often, if not always, restrictions on the level of PC for a path to be valid. PC is denoted $\mathcal{C}^n$, where n is the degree of PC smoothness.

Definition 2.1. Parametric continuity $\mathcal{C}^n$ and regularity (Barsky and DeRose, 1984). A parameterization $\boldsymbol{p}_d(s) = [x_d(s), y_d(s)]^\top$ is said to belong to the class $\mathcal{C}^n$ on the interval $[s_0, s_1]$ if the coordinate functions $x_d(s)$ and $y_d(s)$, are n times continuously differentiable on $[s_0, s_1]$. It is regular if:

$$\boldsymbol{p}_d^s(s) \neq 0, \quad \forall s \in [s_0, s_1]. \tag{2.24}$$

A regular parameterization means that the path never degenerate into a single point.

Definition 2.2. Parametric continuity up to $n = 2$.

- $\mathcal{C}^0$: requires all subpaths to be connected. That is:

$$\boldsymbol{p}_d(i, s_{ub}) = \boldsymbol{p}_d(i+1, s_{lb}), \quad i \in \mathcal{I}^m \backslash \{m\}, \tag{2.25}$$

  where lb and ub denotes lower and upper bound, respectively.

- $\mathcal{C}^1$: states that the velocity vector magnitude and orientation are continuous. Thus, $\mathcal{C}^1$ requires $\mathcal{C}^0$ and:

$$\boldsymbol{p}_d^s(i, s_{ub}) = \boldsymbol{p}_d^s(i+1, s_{lb}), \quad i \in \mathcal{I}^m \backslash \{m\}. \tag{2.26}$$

- $\mathcal{C}^2$: states that the acceleration vector magnitude and orientation are continuous. Thus, $\mathcal{C}^2$ requires $\mathcal{C}^1$ and:

$$\boldsymbol{p}_d^{s^2}(i, s_{ub}) = \boldsymbol{p}_d^{s^2}(i+1, s_{lb}), \quad i \in \mathcal{I}^m \backslash \{m\}. \tag{2.27}$$

When it comes to path-following motion control for a vehicle, it is desired with continuity of a certain degree to ensure bumpless transfers in the joints between segments of a piecewise parametric curve. $\mathcal{C}^1$ continuity entails continuity in course angle, $\mathcal{C}^2$ continuity entails continuity in curvature, and furthermore, $\mathcal{C}^3$ entails continuity in the rate of change in curvature.

2.3 The Bézier Curve

In the 1950s and '60s, there was a need in the automobile and aircraft industries to define free-form shapes accurately. The motivation behind was the ability to design cars in other shapes than basic lines, circles, and parabolas that could be brought from the drawing board with an accurate description, to the pattern shop. To solve this problem, the French mechanical engineer Pierre Étienne Bézier working at Renault car manufacturer, started a lead in transforming how design and manufacturing were done, through mathematical parametrized curves. He started using parametrically defined surfaces expressed with polynomials exhibiting special characteristics.

The primary outcome of Bézier's work is named after himself: The Bézier curve. Bézier took patent and popularized the Bézier curve, but unbeknownst to Bézier , another French engineer, Paul de Casteljau, working for Citroën, had just finished similar work. Since Casteljau's work was not formally published before Bézier had popularized his findings, Bézier's name is associated with the curves. As an acknowledgment to Casteljau's work, the numerically stable method to evaluate polynomials in Bézier curves is named after Casteljau.

The basis of the Bézier curve is called the Bernstein polynomial, named after the Russian mathematician Sergei Natanovich Bernstein. It was first used to prove the

Stone-Weierstrass theorem[2] in 1911. Because of the slow convergence rate of approximating continuous functions, the Bernstein polynomial was deemed not useful. It languished in obscurity in the advent of digital computers. Casteljau's and Bézier's initial formulation did not explicitly use the Bernstein polynomial, but their work is unmistakably linked to it. Today, the Bernstein Polynomial is seen as the foundation of the Bézier curve (Farouki, 2012).

Bézier and Casteljau's insight on how to mathematically use special polynomials to represent complex curves has had a significant impact on computer-aided design. It is used to represent complex shapes with smooth curves that can be scaled indefinitely. For more historical details on the Bézier curve, see Casselman (2008) and references therein.

With several useful properties, the Bézier curve shows great potential when it comes to path planning and generation, and will be used as a basis function in this thesis.

2.3.1 Definition

A Bézier curve is defined by a set of control points $\boldsymbol{P}_i = [x_i, y_i]$ where $i \in [0, 1, \ldots, n]$, and n is called the degree or order of the curve. This implies that a Bézier curve of degree n has $n+1$ control points. The first and the last control points are the end points of the curve. The intermediate control points, if any, generally do not lie on the curve itself. The Bézier is a planar parameterization, denoted $\boldsymbol{B}_{1\times 2}(\theta) = [x(\theta), y(\theta)]^\top$. An explicit formulation expressed as an affine[3] combination is given by:

$$\boldsymbol{B}_{1\times 2}(\theta) = \sum_{i=0}^{n} b_{n,i}(\theta)\boldsymbol{P}_i, \quad \theta \in [0, 1]. \tag{2.28}$$

The variable θ is the normalized time variable and $b_{n,i}(\theta)$ is the Bernstein basis polynomial of degree n, commonly referred to as the blending function and given by:

$$b_{n,i}(\theta) = \binom{n}{i}\theta^i(1-\theta)^{n-i}, \quad i = 0, 1, \ldots, n. \tag{2.29}$$

The first term is the binomial coefficient:

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}. \tag{2.30}$$

[2] In mathematical analysis, the Weierstrass approximation theorem states that every continuous function defined on a closed interval $[a, b]$ can be uniformly approximated as closely as desired by a polynomial function.

[3] An affine combination of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is a vector of linear combination of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, where the sum of the coefficients is 1.

By connecting the Bézier points with lines, starting with $\boldsymbol{P}_0$ and finishing with $\boldsymbol{P}_n$, the control polygon is obtained. See Figure 2.3 for a visualization. The Bézier curve is a set of convex combinations, which is important for the numerical stability of the Bézier curve (Lyche and Mørken, 2008).

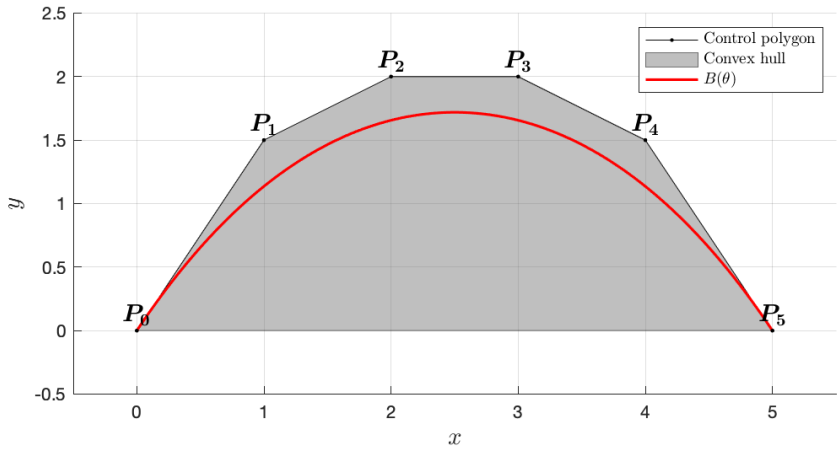


Figure 2.3: A fifth order Bézier curve drawn in red. The control polygon is obtained by connecting the control points $\boldsymbol{P}_0$ to $\boldsymbol{P}_5$ by straight lines. The convex hull, formed by the control polygon, is shaded in gray.

Matrix Formulation

For computational purposes, it is convenient to formulate the Bézier curve by a matrix representation. By recognizing that the Bézier curve is as a linear combination of the control points and the Bernstein basis functions:

$$\boldsymbol{B}_{1\times 2}(\theta) = b_{n,0}(\theta)\boldsymbol{P}_0 + b_{n,1}(\theta)\boldsymbol{P}_1 + \ldots + b_{n,n}(\theta)\boldsymbol{P}_n, \tag{2.31}$$

it is easy to see that this can be written as the dot product between two vectors, such that:

$$\boldsymbol{B}_{1\times 2}(\theta) = \begin{bmatrix} b_{n,0}(\theta) & b_{n,1}(\theta) & \ldots & b_{n,n}(\theta) \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_0 \\ \boldsymbol{P}_1 \\ \vdots \\ \boldsymbol{P}_n \end{bmatrix}_{n\times 2}, \tag{2.32}$$

which again can be converted to:

$$\boldsymbol{B}_{1\times 2}(\theta) = \begin{bmatrix} 1 & \theta & \dots & \theta^n \end{bmatrix} \begin{bmatrix} b_{0,0} & 0 & \dots & 0 \\ b_{1,0} & b_{1,1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,0} & b_{n,1} & \dots & b_{n,n} \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_0 \\ \boldsymbol{P}_1 \\ \vdots \\ \boldsymbol{P}_n \end{bmatrix}_{n\times 2}. \tag{2.33}$$

The coefficients in the matrix are given by (Joy, 2000b):

$$b_{i,j} = (-1)^{i-j} \binom{n}{i} \binom{i}{j}. \tag{2.34}$$

This is convenient when calculating derivatives since only the first vector is dependent on the parameter θ. Note that subscript denoting the order n in Equations (2.33) and (2.34) are left out for clarity. In a compact notation this can be written as:

$$\boldsymbol{B}_{1\times 2}(\theta) = \boldsymbol{a}^{\top}_{n\times 1}(\theta)\boldsymbol{M}_{n\times n}\boldsymbol{P}_{n\times 2}, \tag{2.35}$$

where $\boldsymbol{a}^{\top}_{n\times 1}$ is the *monomial basis vector*, $\boldsymbol{M}_{n\times n}$ is the *spline matrix*, and the product of them is known as the *blending function*.

Example 2.1. *For a septic Bézier curve consisting of eight control points, the monomial basis vector is given as:*

$$\boldsymbol{a}^{\top}_{8\times 1}(\theta) = \begin{bmatrix} 1 & \theta & \theta^2 & \theta^3 & \theta^4 & \theta^5 & \theta^6 & \theta^7 \end{bmatrix},$$

and the spline matrix as:

$$\boldsymbol{M}_{8\times 8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -7 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 21 & -42 & 21 & 0 & 0 & 0 & 0 & 0 \\ -35 & 105 & -105 & 35 & 0 & 0 & 0 & 0 \\ 35 & -140 & 120 & -140 & 35 & 0 & 0 & 0 \\ -21 & 105 & -210 & 210 & -105 & 21 & 0 & 0 \\ 7 & -42 & 105 & -140 & 105 & -42 & 7 & 0 \\ -1 & 7 & -21 & 35 & -35 & 21 & -7 & 1 \end{bmatrix}.$$

2.3.2 Derivatives

The derivative of an n-th order Bézier curve is a Bézier curve of order $n-1$. Since the control points in a Bézier curve are independent of the parametrized variable θ, the derivative of a Bézier curve is found by computing the derivative of the Bernstein basis function. Using the definition of the Bernstein basis polynomial in Equation (2.29) and taking the derivative, it can be shown that (Joy, 2000a):

$$b^{\theta}_{n,i}(\theta) = n(b_{n-1,i-1}(\theta) - b_{n-1,i}(\theta)), \quad 0 \leq i \leq n. \tag{2.36}$$

This shows that the derivative of a Bernstein polynomial can be given as a linear combination of Bernstein polynomials. Taking the derivative of Equation (2.28) and plugging in the result of Equation (2.36), the derivative of a Bézier curve can be written as:

$$\boldsymbol{B}^{\theta}_{1\times 2}(\theta) = n\sum_{i=0}^{n-1} b_{n-1,i}(\theta)(\boldsymbol{P}_{i+1} - \boldsymbol{P}_i), \quad \theta \in [0,1]. \tag{2.37}$$

The first derivative of the Bézier curve is known as a *hodograph*. The second derivative becomes the second hodograph, etc. From Equation (2.37), one can see that the derivative is defined by the difference between the control points in the original curve. By defining $\boldsymbol{Q}_i = \boldsymbol{P}_{i+1} - \boldsymbol{P}_i$, Equation (2.37) is simplified to:

$$\boldsymbol{B}^{\theta}_{1\times 2}(\theta) = n\sum_{i=0}^{n-1} b_{n-1,i}(\theta)\boldsymbol{Q}_i, \quad \theta \in [0,1]. \tag{2.38}$$

By using Equations (2.28) and (2.38), one can find any higher-order degree of derivatives. However, to find a given higher-order derivative, one has to find the control points of every lower degree derivative.

Example 2.2. *Given an n-th order Bézier curve. The Bézier curve itself and its derivatives up to order three, evaluated at the endpoints ($\theta = 0$ and $\theta = 1$), is given as:*

$$\begin{aligned}
\boldsymbol{B}_{1\times 2}(0) &= \boldsymbol{P}_0 \\
\boldsymbol{B}_{1\times 2}(1) &= \boldsymbol{P}_n \\
\boldsymbol{B}^{\theta}_{1\times 2}(0) &= n(\boldsymbol{P}_1 - \boldsymbol{P}_0) \\
\boldsymbol{B}^{\theta}_{1\times 2}(1) &= n(\boldsymbol{P}_n - \boldsymbol{P}_{n-1}) \\
\boldsymbol{B}^{\theta^2}_{1\times 2}(0) &= n(n-1)(\boldsymbol{P}_2 - 2\boldsymbol{P}_1 + \boldsymbol{P}_0) \\
\boldsymbol{B}^{\theta^2}_{1\times 2}(1) &= n(n-1)(\boldsymbol{P}_n - 2\boldsymbol{P}_{n-1} + \boldsymbol{P}_{n-2}) \\
\boldsymbol{B}^{\theta^3}_{1\times 2}(0) &= n(n-1)(n-2)(\boldsymbol{P}_3 - 3\boldsymbol{P}_2 + 3\boldsymbol{P}_1 - \boldsymbol{P}_0) \\
\boldsymbol{B}^{\theta^3}_{1\times 2}(1) &= n(n-1)(n-2)(\boldsymbol{P}_n - 3\boldsymbol{P}_{n-1} + 3\boldsymbol{P}_{n-2} - \boldsymbol{P}_{n-3})
\end{aligned}$$

2.3.3 Properties

The Bézier curve possesses several interesting properties valuable for different fields of study. An overview of the most important properties when it comes to path generation is stated in the following section. See MiT (2009) for a more comprehensive list of useful properties the Bézier curve exhibits.

Variation Diminishing Property

The Bézier curve exhibit the *variation diminishing property*: A straight line will intersect the "legs" (the lines going between the control points) of the control polygon at least as many times as it crosses the Bézier curve itself. More intuitively, this means that the Bézier curve oscillates less than its control polygon. This property is essential in the context of path generation because it implies that the Bézier curve is always smoother than the polygon formed by its control points. Furthermore, it tends to reduce the shape variations in the control polygon rather than enlarge it. See Sarfraz et al. (2018) for a visualization.

The Convex Hull Property

Closely related to the variation diminishing property is the *convex hull property*. It states that the Bézier curve is always completely contained inside the *convex hull*, formed by the control polygon. The convex hull of a set of control points is the smallest convex set that contains all control points. See Figure 2.3 for an example. In the case of path generation, this property is important because it guarantees that if the control points are placed within a small region, the curve will not "blow up" arbitrarily. See Runge's phenomenon (Runge, 1901). Closely related to this property is the fact that adjusting one control point changes the curve in a "predictable manner". The curve, practically speaking, follows the adjusted control point.

The Endpoint Interpolation Property

The Bézier curve inhibits the property that it always passes through the first and the last control point, meaning that the first and the last control point are the endpoints:

$$\boldsymbol{B}_{1\times 2}(0) = \boldsymbol{P}_0 \text{ and } \boldsymbol{B}_{1\times 2}(1) = \boldsymbol{P}_n. \tag{2.39}$$

This can easily be derived from the explicit formulation of the Bézier curve in Equation (2.28). Further, it is possible to freely specify higher-order derivatives in the endpoints by the placement of the control points. This property is known as the *endpoint interpolation property* and is very convenient when studying the smoothness of the concatenation point between two Bézier curves.

Definition 2.3. A planar curve will belong to the class $\mathcal{D}^n$ if it is possible to freely specify the derivatives of order n in both endpoints[4] .

[4]Note that Definition 2.3 will entail that $\mathcal{C}^n$-continuity is possible in the joints.

Shape Control

Shape control of a path investigates what happens when there is a change in the position of a waypoint, or another one is added. There are three possible scenarios for what can happen (Lekkas, 2014):

1) *Global control*: changes will be made on the entire path.

2) *Local control*: possible to add or change a waypoint without affecting the rest of the path.

3) *Partial control*: local changes possible in some cases.

If a path is constructed with a single Bézier curve, where the waypoints are considered to be control points, it exhibits the property of global control. Thus, moving a control point alters the shape of the whole curve. It means that moving or adding a control point affects what happens with the vehicle's current position and heading. This is undesired behavior, but will be handled by the stepwise approach, explained in Chapter 3. This is because we are only facing one segment at a time, with a predefined number of control points to be used.

2.4 The Path Planning Problem

Some basic definitions and concepts are needed to introduce our 2D path planning problem. See Figure 2.4 for a visualization.

- **Configuration space:** The ASV can be considered as a 2D shape that can translate and rotate, and thus be represented using three parameters (x, y, ψ). The configuration space $\mathcal{X} \in SE(2)$, represents the workspace where planning is done.

- **Obstacle space:** The space that the agent cannot move to is known as the obstacle space $\mathcal{X}_{obs} \subsetneq \mathcal{X}$. The obstacles may be static or dynamic. We assume that the obstacle space is known.

- **Free space:** The free space is denoted $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$.

- **Goal region:** The space that we want the agent to move to is known as the goal region $\mathcal{X}_{goal} \subset \mathcal{X}$.

We define the feasible path planning problem as: "Find a collision-free motion of connecting waypoints between initial state x_0 and a goal state x_{goal} within a specified workspace". An algorithm that addresses the problem is said to be *complete* if it terminates in finite time and returns a solution if one exists or failure otherwise

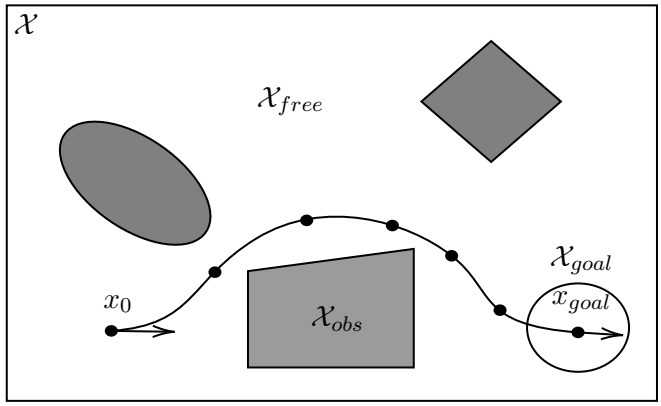


Figure 2.4: A configuration space $\mathcal{X}$ where the white area is $\mathcal{X}_{free}$, the gray area is $\mathcal{X}_{obs}$, and the transparent circle containing x_{goal} is the goal region $\mathcal{X}_{goal}$.

(Karaman and Frazzoli, 2011). Usually, in path planning, we are not only interested in a feasible path, but an optimal one as well. Given a *cost function*, which assigns a non-negative cost to all nontrivial collision-free paths, the optimality problem of path planning is to find a feasible path with minimal cost (Karaman and Frazzoli, 2010).

Problem 2.1. *The Optimal Path Planning Problem (Karaman and Frazzoli, 2010). Let $\sigma : [0, s] \mapsto \mathcal{X}_{free}$ be a sequence of states (a path) and Σ be the set of all non-trivial paths. Given a bounded connected configuration space $\mathcal{X}$, an obstacle space $\mathcal{X}_{obs}$, an initial state x_0, a goal state $x_{goal} \in \mathcal{X}_{goal}$, and a specified cost function $c : \Sigma \mapsto \mathbb{R}_{\geq 0}$. Find a path σ^*, that minimizes the cost function, while connecting x_0 to x_{goal} through $\mathcal{X}_{free}$:*

$$\sigma^* = \arg\min_{\sigma \in \Sigma} \{c(\sigma) \mid \sigma(0) = x_0,\ \sigma(1) = x_{goal},\ \forall s \in [0, 1],\ \sigma(s) \in \mathcal{X}_{free}\}, \quad (2.40)$$

where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers.

Problem 2.1 is the fundamental problem we want to solve when designing our path planner. Difficulties arise when the structure of the agent and the environment gets more sophisticated (Russell and Norvig, 2009). The problem plays a vital role in several applications, such as robotics, transportation, information systems (message routing), and video games. Much research has been devoted to the problem, and the first complete algorithms were introduced in the 1970s (Reif, 1979).

As of today, path planning algorithms are widely divided into three main categories, according to the methodologies used to generate the path, namely:

1) Combinatorial algorithms.

2) Sampling-based algorithms.

3) Potential field algorithms.

Combinatorial algorithms characterize $\mathcal{X}_{free}$ by capturing the connectivity of $\mathcal{X}_{free}$ into a graph/roadmap and finds a path using a typical shortest-path algorithm, such as the A* algorithm. Well-known techniques include cell decomposition, visibility graphs, and Voronoi roadmaps. Several combinatorial methods are proven to be optimal, but only up to the resolution of the partitioned configuration space. Combinatorial algorithms can effectively solve low-dimensional problems. Nevertheless, for high-dimensional configuration spaces, the category of algorithms becomes computationally intractable (Yu, 2016), because their running time exponentially depends on the dimension of $\mathcal{X}$.

Instead of characterizing the whole configuration space, sampling-based algorithms take random samples from $\mathcal{X}$, declare them as vertices if in $\mathcal{X}_{free}$, and try to connect nearby vertices with local planners. Sampling-based algorithms avoid local minima and solve several path planning problems quickly. A drawback with sampling-based algorithms is that they are unable to determine if no path exists at all, but as time goes, the probability of failure decreases to zero. For a detailed overview see e.g. Gasparetto et al. (2015) and Souissi et al. (2013).

Both combinatorial and sampling-based algorithms intend to capture the connectivity of $\mathcal{X}_{free}$ into a graph/roadmap. In contrast, potential field methods represent the agent as a particle in $\mathcal{X}_{free}$ influenced by an artificial potential field, where obstacles induce repulsive forces and the goal attractive ones. Potential field methods are efficient, but besides harmonic potential field methods (Panati et al., 2015), the category is prone to get stuck in local minima. See Elia Nadira et al. (2016) for a reference on potential field algorithms used for path planning. Note that there are several hybrid variants between these three approaches.

Today, sampling-based methods are, in many ways, considered state-of-the-art in high-dimensional path planning problems, because they have been proven to work well in practice and possess theoretical guarantees such as *probabilistic completeness* (Karaman and Frazzoli, 2011). Arguably, the most influential sampling-based algorithms today, includes Rapidly-exploring Random Tree (RRT) (LaValle, 1998; LaValle and Kuffner, 2000) and Probabilistic RoadMap (PRM) (Kavraki et al., 1996, 1998).

For a comparative study of different path planning and collision avoidance algorithms applied on ASVs, refer to recently published surveys by Vagale et al. (2020a,b). In this thesis, we will investigate and design one global planner using a combinatorial algorithm, the A* algorithm, where $\mathcal{X}_{free}$ is partitioned using a Voronoi diagram. Working together with the global planner, a local planner using the sampling-based RRT method will be used.

2.5 Map Representation and Partitioning

Obtaining and representing information of the environment, is formally known as *mapping* in the robotics and AI community. Mapping is one of the core features required to make mobile robots truly autonomous, and has been a highly active research area since the 1980s. It has since been divided into two main approaches; metric and topological. Topological maps typically represent environments through graphs where vertices represent significant places and arcs contain information on how to navigate between them. See Thrun (2003) for an overview on relevant research. These maps usually lack precise details on scaling and distances and direction is subject to change and variation. Thus they are not suited for maintaining precise details needed for accurate maneuvering of an ASV, as studied in this thesis. Metric maps, on the other hand, capture detailed geometric properties of the environment, and is therefore more suited for the objectives stated in this thesis. One of the most popular algorithms and representations for obtaining metric maps is *occupancy grid map algorithms*, first introduced by Elfes (1989). Occupancy grid maps (OGMs) represents the environment by fine-grained binary grids representing occupied and free space. See Figure 2.5. Of course, high resolution comes with a computational cost, but it gives a more precise representation of the environment.

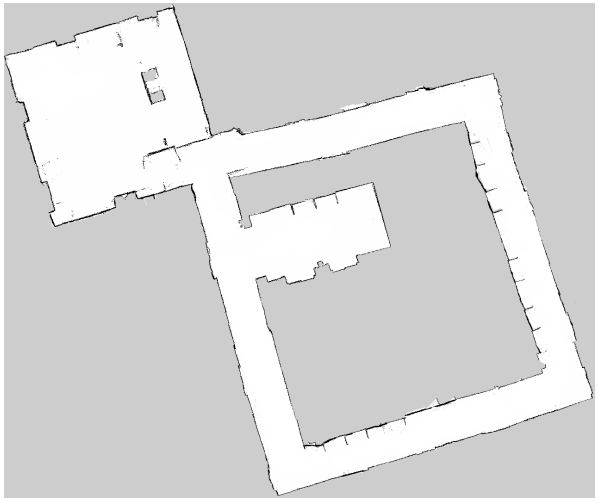


Figure 2.5: An occupancy grid map of a floor obtained from raw odometry and laser range data using a Pioneer 3DX robot and a Hokuyo URG-04LX laser range finder. The grayscale indicates the posterior probability of each grid cell: white corresponds to free with high certainty, black to occupied with high certainty. The gray area represents the prior (unknown area). The map has been generated by Knædal et al. (2018).

Due to ambiguities in perception and actuation, the field of localization and map-

ping has been dominated by probabilistic techniques. See Thrun (2003) for references on statistical framework used. Acquiring maps is known as a "chicken-and-egg" problem commonly referred to has *simultaneous localization and mapping* (SLAM) (Thrun et al., 2005). In this thesis, OGMs will serve as a foundation when designing our navigation system. We will further assume that the map is known in advance, which omits the SLAM problem.

2.5.1 Cost Maps

As mentioned in Section 2.4, a path planner should find a collision-free motion of connecting waypoints between an initial point x_0 and a goal point x_{goal} within the specified workspace. Combined with the right guidance system, they can navigate agents around in the configuration space with great skill. However, these algorithms optimize based on an oversimplified binary constraint (occupied space or not) of finding efficient collision-free paths. For example, an ASV driving a few meters away from known piers is perfectly acceptable in most cases. However, driving too close to other vessels is undesirable. The shortest collision-free path may not always be optimal if a longer one avoiding hazardous areas can be chosen. By introducing cost maps addressing this problem, more sophisticated path planners can be designed.

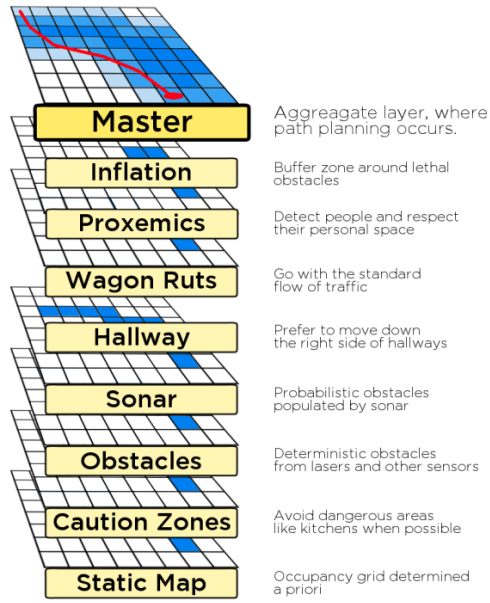
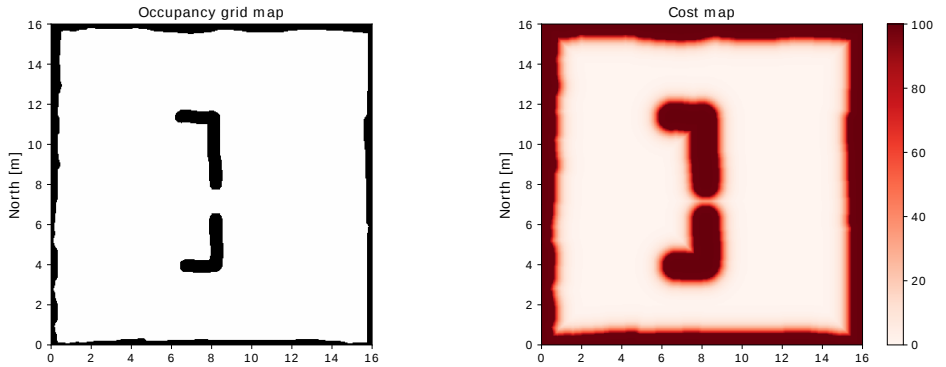


Figure 2.6: A stack of cost map layers, showcasing the different contextual behaviors achievable with the layered cost map approach. Courtesy: Lu et al. (2014).

Cost maps can incorporate information about different obstacles and constraints. Lu et al. (2014), introduces the notion of *layered cost maps* that work by separating the processing of cost map data into semantically separated layers. Each layer tracks one type of obstacle or constraint and then modifies a master cost map, which is used for path planning. Examples of different obstacles and constraints and their respective layer are given in Figure 2.6. In Figure 2.7b a cost map is visualized.



(a) **(b)**

Figure 2.7: A cost map with static obstacles and their inflation radius set to 1 meter. Figure **(a)** depicts an occupancy grid map. Figure **(b)** shows a cost map generated from the occupancy grid map in **(a)**. The cost map is created using the ROS package `costmap_2d`[5].

2.5.2 Voronoi Partitioning

A Voronoi diagram (VD) is a tool to divide a configuration space into regions determined by, in our case, the obstacles presented in the space. The VD produces a roadmap that can be traversed by a combinatorial algorithm. They have been widely applied in different applications since the beginning of the 20th century. See Aurenhammer (1991) for a survey. Especially within the field of path planning, VDs has received much attention. This is true mainly due the to following three reasons (Candeloro et al., 2013):

1) The VD divides the configuration space in a way such that the borders of the regions have maximum distance from all the obstacles in the cluttered environment.

2) Construction of VDs has $O(n)$ complexity, while the majority of other mathematical tools solve the same problem with $O(n^2)$ complexity.

3) Kinematics of mobile robots allow them to change heading without affecting the other degrees of freedom, so they can easily run a path composed by a sequence of straight lines.

Mathematical Formulation and Metric Definition

Given a set of *generator points* $\mathcal{P} = \{p_1, \ldots, p_n\}$, contained in the configuration space $\mathcal{X}$ (here $\mathcal{X} \in SE(2)$), where a metric function $d(\cdot)$ is defined. The method

[5] `http://wiki.ros.org/costmap_2d`

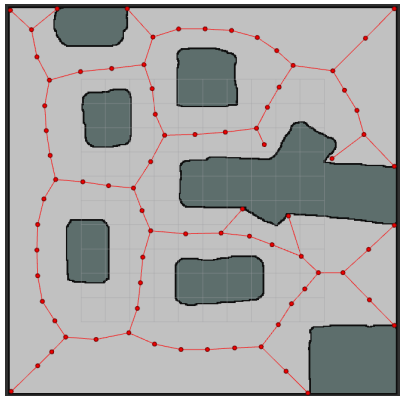


Figure 2.8: A VD constructed from a occupancy grid map. Map created by Binder (2017).

associates what is called a *Voronoi region* $\mathcal{R}_i$ defined by the set of points $x_i \in \mathcal{X}$ such that the distance to p_i is lower than the distance from x_i to any other point in $\mathcal{P}$. In mathematical terms, if:

$$d(x, p) = \inf\{d(x, p) | p \in \mathcal{P}\}, \tag{2.41}$$

the Voronoi region is defined as (Candeloro et al., 2013):

$$\mathcal{R}_k \triangleq \{x \in \mathcal{X} | d(x, \mathcal{P}_k) \leq d(x, \mathcal{P}_j) \; \forall j \neq k\}. \tag{2.42}$$

The VD $\mathcal{V}(p_i)$ will then be the set of the Voronoi region borders, that is the intersection of the tuple of cells $(\mathcal{R}_k)_{k \in \mathcal{K}} \; : \; \mathcal{V}(p_i) = \sum_i \mathcal{R}_i$, where $\mathcal{K}$ is a set of indices. From the definition it is clear that the shape $\mathcal{V}(p_i)$ will depend on the definition of the metric function $d(\cdot)$. In this thesis we will use the Euclidean distance as the metric function:

$$d\Big((x_1, y_1), (x_2, y_2)\Big) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{2.43}$$

If the obstacles are polygon-shaped and the edges are considered to be generator points, the metric will generate Voronoi regions with straight-line borders.

By using the obstacles in the configuration space as generator points, it is evident that the borders of the Voronoi regions maximize the distance to the obstacles. Thus, we achieve a raw obstacle-free roadmap consisting of waypoints connected by straight-line paths. See Figure 2.8 for a visualization of a VD constructed from an OGM.

2.6 The Maneuvering Problem

In several control applications, the main goal is to steer an object to achieve a certain *control objective*. The different control objectives are typically classified into setpoint

regulation, trajectory tracking, path following, or maneuvering. In this thesis, we will address the last two. *Path following* is following a predefined path independent of time. No restrictions are placed on the temporal propagation along the path (Fossen, 2011a). In *maneuvering*, the first and most crucial task is path following. The second and less important task is to satisfy a desired dynamic behavior along the path, for example, a specified desired speed. For the ASV treated in this thesis, this means that we want to follow the predefined path at the desired speed. However, if we encounter a place in time where it is not achievable to satisfy both tasks (e.g., a sharp turn), we should sacrifice the speed to achieve a more accurate path following.

The following is based on Chapter 2 from Skjetne (2005). The system output from the navigation system depicted in Figure 1.1, will be the ship pose $\boldsymbol{\eta} = [\boldsymbol{p}, \psi]^\top \in \mathbb{R}^2 \times [-\pi, \pi)$, and the body-fixed linear and angular velocity vector $\boldsymbol{\nu} = [u, v, r]^\top \in \mathbb{R}^3$. The desired pose for an ASV is represented by:

$$\mathcal{P} \triangleq \{\boldsymbol{\eta} \in \mathbb{R}^2 \times [-\pi, \pi) \ : \ \exists s \in \mathbb{R} \text{ s.t. } \boldsymbol{\eta} = \boldsymbol{\eta}_d(s)\}, \tag{2.44}$$

where $\boldsymbol{\eta}_d$ is parametrized by the continuous variables s.

Problem 2.2. *The Maneuvering Problem for an ASV.*
The Maneuvering Problem for an ASV is comprised of two tasks:

1. ***Geometric task:*** *Given a desired pose $\boldsymbol{\eta}_d(s(t))$, force the vessels pose $\boldsymbol{\eta}(t)$ to converge to the desired path:*

$$\lim_{t\to\infty} [\boldsymbol{\eta}(t) - \boldsymbol{\eta}_d(s(t))] = 0. \tag{2.45}$$

2. ***Dynamic task:*** *Satisfy one or more of the following assignments:*

   2.1. ***Speed assignment:*** *Force the path speed $\dot{s}(t)$ to converge to the desired speed $v_s(t, s(t))$:*

$$\lim_{t\to\infty} [\dot{s}(t) - v_s(t, s(t))] = 0. \tag{2.46}$$

   2.2. ***Time assignment:*** *Force the path variable s to converge to a desired time signal $v_t(t)$:*

$$\lim_{t\to\infty} [s(t) - v_t(t, s(t))] = 0. \tag{2.47}$$

The maneuvering problem, as stated above, only highlights the superior goal of convergence to the path and to fulfill the dynamic assignment. For a path-following motion control scenario for a marine craft, the speed assignment is considered the most suitable dynamic assignment and is the one considered in this thesis. However, note that a time scheduling along the path could also be important, especially for

short sea shipping. Other dynamic tasks are also possible. See Skjetne (2005) for details.

Combining the generated path with a dynamic assignment along the path ensures that dynamic and temporal requirements are satisfied. The maneuvering problem states that the geometry of the path and the dynamical behavior along the path can be defined and controlled separately. This means that a path can be generated, and the speed can be controlled without having to regenerate a new path. This approach becomes handy when analyzing the vessel feasibility, considering the combination of the generated path and speed assignment along the path.

2.7 The Concept Vessel ReVolt by DNV GL

DNV GL is one of the leading providers of quality assurance services and risk management in the maritime industry. Due to the increasing integration of autonomy in the sector, a set of standards concerning safety, quality, and integrity of these systems must be made. The concept vessel ReVolt serves as a research and development platform for this purpose.

ReVolt is a crewless and fully battery-powered ship, intended to serve as cargo ship along the Norwegian coast. It is a low-speed vessel operating at 6 knots, with a range of 100 nautical miles and a cargo capacity of 100 twenty-foot containers. As well as contributing to increased profit margin in short sea shipping, ReVolt is set to reduce the number of accidents in the industry, contributing to a safer maritime industry (Tvete, 2020).

A 1:20 scale model of the ReVolt concept vessel has been built and is shown in Figure 2.9. It is 3 m long and fully battery-powered, with a maximum speed of approximately 1.5 $\mathrm{m/s}$ during transit operation. Contributions through masters thesis have been made to ReVolt already. This thesis depends on several of them. Alfheim and Muggerud (2017) made several sensor improvements and implemented a DP system for the real vessel. Abrahamsen (2019) improved the DP system and made it work with a pseudoinverse thrust allocation algorithm developed by DNV GL. In May 2020, Andreas Bell Martinsen implemented the current nonlinear observer, an Extended Kalman Filter (EKF). Kamsvåg (2018) mounted and worked on sensor fusion between the Ladybug Camera and the Velodyne lidar. Further work on sensor fusion has been done by Norbye (2019).

Figure 2.9: The ReVolt model scale ship. Photo: Simen Sem Øvereng (2019).

Chapter 3

Problem Formulation

The process of planning and executing a collision-free, feasible path for an agent moving around in an environment has been extensively studied in different fields of study, e.g., the AI and robotics community. The AI community distinguishes between different task environments based on a set of different properties. For a detailed explanation of different types of environments, refer to Russell and Norvig (2009).

For the case of an ASV studied in this thesis, we are facing a partially observable, multi-agent, stochastic, continuous, and dynamic environment (in some cases, an unknown environment as well). Each property makes it a more complex problem to solve. A common thread for the GNMC system in these types of environments is the necessity of *replanning* to handle real-time constraints. For example, somewhere along the way from the start to the end location, the agent has to replan its route and desired speed in the presence of new obstacles discovered along the way. For this reason, we face the path planning problem in a *stepwise manner*. That is, for each step in time where new relevant information is available, the agent can replan its path and speed according to the new information. Intuitively, this coincides with how we humans and various animals solve similar problems. We put up a global plan of how to reach the destination, which we divide into smaller tasks and then conquer. If something happens during the execution, we replan "on the go" to solve the problem.

The stepwise approach sets up a framework and will serve as a foundation for this thesis. Keeping Figure 1.1 in mind, the following sections formulate the objective and functional inputs and outputs for each subsystem investigated in this thesis. Finally, an overall problem statement for the thesis is given.

3.1 Control System for a Stepwise Maneuvering Problem

The control system of an ASV consists of three main submodules: the *high-level control*, the *control allocation*, and the *low-level control*. See Figure 3.1. Given the reference signals from the guidance system, the task of the high-level controller is to calculate the desired generalized thrust forces $\boldsymbol{\tau}_d \in \mathbb{R}^n$ to be applied to the vessel. The marine craft treated in this thesis is a 3 DOF system such that $n = 3$.

The generalized forces are then fed into the control allocation, which distributes the control forces to the actuators in terms of control input $\boldsymbol{u} \in \mathbb{R}^r$, where r denotes the number of control inputs. The low-level controller relates the desired thrust given by the thrust allocation to the commanded motor torque. In this thesis, we will focus on high-level control and thrust allocation.

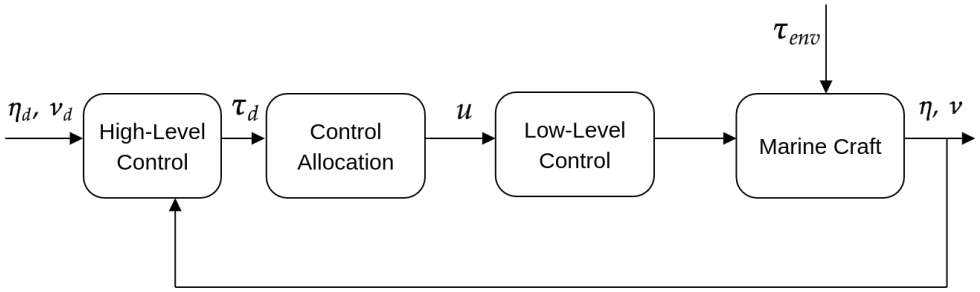


Figure 3.1: A control system block diagram. The control law produces the generalized forces $\boldsymbol{\tau} \in \mathbb{R}^n$. The control allocation distributes the control forces to the actuators in terms of control input $\boldsymbol{u} \in \mathbb{R}^r$.

The high-level controller, together with the thurst allocation, is a dynamic algorithm that continuously calculates the forces that must be applied to achieve a certain control objective. The high-level controller will usually consist of a feedforward term provided by the guidance system and a feedback term from the system states provided by the measurement system. As stated in Section 2.6, we will focus on path following and maneuvering. The maneuvering objective for the ASV is to follow a desired pose $\boldsymbol{\eta}_d(s(t))$ given by Equation (2.44) with a speed assignment $v_s(t, s(t))$ along the path. The control objective is then to design a controller $\boldsymbol{\tau}$ for our 3 DOF equation of motion given by Equation (2.1), such that:

$$\left.\begin{aligned} \boldsymbol{\eta}(t) &\to \boldsymbol{\eta}_d(s(t)) \\ \dot{s}(t) &\to v_s(t, s(t)) \end{aligned}\right\} \text{ as } t \to \infty. \tag{3.1}$$

Since the vessel is assumed to be fully actuated, both tasks are feasible.

3.2 Navigation system for a Stepwise Maneuvering Problem

A global low-resolution path planner is combined with a local dynamic high-resolution path planner. See Figure 1.1 for a visualization. The task of a stepwise path planner for a marine vessel is (Skjetne, 2019):

1. The global planner is to determine the initial, intermediate, and destination waypoint with corresponding reference speed in the regions $u_d(t)$ (and its derivative $\dot{u}_d(t)$ if needed).

2. The local planner is to determine the current waypoint $\boldsymbol{x}_0$ and next target waypoint $\boldsymbol{x}_t$, which brings the vessel closer to the next intermediate/destination waypoint, following the global plan as far as possible.

It may be possible to assume knowledge of a few waypoints coming up, but in this thesis, it is assumed that the guidance system will at most be provided with one waypoint ahead, as well as all previous ones.

Note that a scheduler that determines the time scheduling along the path can also be an essential task of the path planner. This could be especially important for short-sea shipping, but is left out in this thesis.

3.3 Guidance System for a Stepwise Maneuvering Problem

For a stepwise maneuvering problem, the guidance system in Figure 1.1 is to generate the desired path and speed profile along the curve, which together constitutes the reference signals into the control system. These signals must be generated in a format corresponding to the maneuvering controller in use. Given the stepwise approach, the path needs to be parametrized in a hybrid fashion, described in Section 2.2.1. We start with the heading since it introduces an essential requirement for the curve to be generated. Given the waypoints and reference speed $u_d(t)$ from the path planner, the hybrid path generator is to for each path segment i (Skjetne, 2019):

1) Generate the desired heading curve $\psi_d(i,\theta)$ defined in Equation (2.15), and its derivatives $\psi_d^{\theta^n}(i,\theta)$ up to order n. For our case $n = 2$, because of requirements imposed by the maneuvering controller later designed. This is explained in details in Chapter 6. Thus we have:

$$\psi_d^{\theta}(i,\theta) = \frac{x_d^{\theta}(i,\theta)y_d^{\theta^2}(i,\theta) - x_{d,i}^{\theta^2}(\theta)y_d^{\theta}(i,\theta)}{x_d^{\theta}(i,\theta)^2 + y_d^{\theta}(i,\theta)^2}, \tag{3.2}$$

$$\psi_d^{\theta^2}(i,\theta) = \frac{x_d^{\theta}(i,\theta)y_d^{\theta^3}(i,\theta) - x_d^{\theta^3}(i,\theta)y_d^{\theta}(i,\theta)}{x_d^{\theta}(i,\theta)^2 + y_d^{\theta}(i,\theta)^2} \tag{3.3}$$

$$-2\frac{\left(x_d^{\theta}(i,\theta)y_d^{\theta^2}(i,\theta) - x_d^{\theta^2}(i,\theta)y_d^{\theta}(i,\theta)\right)\left(x_d^{\theta}(i,\theta)x_d^{\theta^2}(i,\theta) - y_d^{\theta^2}(i,\theta)y_d^{\theta}(i,\theta)\right)}{\left(x_d^{\theta}(i,\theta)^2 + y_d^{\theta}(i,\theta)^2\right)^2}.$$

2) Generate a desired curve:

$$\boldsymbol{p}_d(i,\theta) = \begin{bmatrix} x_d(i,\theta) & y_d(i,\theta) \end{bmatrix}^\top, \quad \theta \in [0,1], \tag{3.4}$$

and its necessary derivatives $\boldsymbol{p}_d^{\theta^n}(i,\theta)$ up to order n. Note the third derivatives appearing in Equation (3.3). For the heading to be continuous, we must generate the derivatives up to order $n = 3$.

3) Generate the speed profile $v_s(t, s(t))$, and its respective derivatives:

$$v_s^t(t, s(t)) \text{ and } v_s^\theta(t, s(t)). \tag{3.5}$$

The derivatives is needed for the maneuvering controller design.

Given the requirement of generating $\psi_d^{\theta^2}(i,\theta)$ from the maneuvering controller, it is clear that the path must be at least $\mathcal{C}^3$ continuous because of the third derivatives appearing in Equation (3.3).

3.4 Problem Statement

The main objective is to develop an intelligent guidance concept for an ASV from the initial to the target point. The problem will be faced in a stepwise manner to facilitate real-time system performance. This sets some superior requirements for the complete GNMC system: It must be computationally efficient, reliable, and robust. Further, it must incorporate the dynamical constraints of the vessel, stated in Section 2.1.

The navigation system will consist of a global low-resolution path planning method combined with a dynamic high-resolution method such that the planner can react to local ambient conditions. This includes consideration of global and local mapping and partitioning of the operation area and how to integrate the global and local methods seamlessly. The guidance system will provide the reference signal, which constitutes a dynamic assignment together with a path generator. The Bézier curve will serve as a basis function for the proposed path generator. For the control system, a maneuvering control law will be proposed together with a control allocation algorithm. The measurement system will not be investigated in this thesis, but an observer providing the necessary state estimates is assumed.

The proposed block diagram of a GNMC system in Figure 1.1 will serve as a key reference. We will investigate and propose block by block and finally put the subsystems together.

3.5 Assumptions and Delimitations

A set of assumptions and delimitations are made when approaching the problem:

- The work is limited to the horizontal plane.
- Environmental loads are not considered.

- We will focus on low-speed (DP) transit operations where the heading is equal to the direction of the path.
- A simplified 3 DOF control design model with the assumptions and simplifications done in Section 2.1 will be considered.
- The vessel is assumed to be fully actuated.
- Necessary state estimates from the measurement system are provided by an observer.
- It is assumed that the operation area is known prior to operation.
- It is assumed that information about obstacles are provided by necessary onboard sensors.
- It is assumed that at most one waypoint ahead at a time is known, as well as all previous ones.
- We may assume that a straight-line corridor, within some path-transversal distance bound from current to next waypoint, is collision-free for obstacles. Thus, the vessel is freely allowed to move inside the given corridor.
- It is assumed that there is a maximum restriction of $\pm$ 90° turn between previous, current, and next waypoint. This is a reasonable assumption since the vessel will hopefully not need to move backward.
- The international regulations for preventing collisions at sea (COLREG) are not considered.

Chapter 4

Guidance System

This chapter will focus on designing the guidance system. A path generator is proposed using the Bézier curve as a basis function. The requirements and objectives of the path generator are stated in Section 3.3. First, an analysis of the Bézier curve is carried out. Then, a proposition for what constitutes a reasonable path will be given along with the desired behavior, properties, and constraints. Two methods are proposed: first, a pragmatic solution, second, a solution involving optimization.

At last, we design the speed assignment. Some parts of the following chapter are based on the author's specialization project (Knædal, 2019). It is included to make the master thesis self-contained.

4.1 Analysis of the Bézier Curve

As mentioned in Section 2.3.3, the Bézier curve exhibits several desired properties when it comes to path generation. To fully utilize the versatility of the Bézier curve, an analysis of the necessary degree of the curve has to be carried out. A Bézier curve constructed by a large set of control points is numerically unstable (Wang et al., 2017). Thus, it is desired to keep the degree as low as possible. It is also convenient to keep the degree low since higher-order Bézier curves are computationally more expensive.

From requirements and objectives stated in Section 3.3, the path is required to be at least $\mathcal{C}^3$ to ensure bumpless transfer in the concatenation points. From Definition 2.1, a single Bézier curve of order n will belong to the class $\mathcal{C}^n$. This implies that for a single interval, a Bézier curve of order three fulfills the property of being $\mathcal{C}^3$. Thus, the following analysis is concerned with the discontinuity that may occur in the concatenation points of the spline. Table 4.1 summarizes the necessary properties.

Using Definition 2.2 and the derivatives found in Example 2.2, the constraints for $\mathcal{C}^3$ continuity in the joints can be formulated. For the path to only be continuous, that is $\mathcal{C}^0$, we have that:

$$\boldsymbol{P}_{0,i+1} = \boldsymbol{P}_{n,i}, \quad i \in \mathcal{I}^m, \tag{4.1}$$

where the first subscript denotes which control point and the second which curve

segment i in the set of m segments, $\mathcal{I}^m = \{1, 2 \ldots, m\}$. Further $\mathcal{C}^1$ can be stated as:

$$\boldsymbol{P}_{1,i+1} = 2\boldsymbol{P}_{n,i} - \boldsymbol{P}_{n-1,i}, \quad i \in \mathcal{I}^m, \tag{4.2}$$

where the result is simplified with the relation found in Equation (4.1). Further, we can state the requirements for $\mathcal{C}^2$ and $\mathcal{C}^3$ continuity in the same manner as:

$$-2\boldsymbol{P}_{1,i+1} + \boldsymbol{P}_{2,i+1} = \boldsymbol{P}_{n-2,i} - 2\boldsymbol{P}_{n-1,i}, \quad i \in \mathcal{I}^m, \tag{4.3}$$

$$\begin{aligned} 3\boldsymbol{P}_{1,i+1} - 3\boldsymbol{P}_{2,i+1} + \boldsymbol{P}_{3,i+1} = 2\boldsymbol{P}_{n,i} - 3\boldsymbol{P}_{n-1,i} \\ + 3\boldsymbol{P}_{n-2,i} - \boldsymbol{P}_{n-3,i}, \quad i \in \mathcal{I}^m \end{aligned} \tag{4.4}$$

We can conclude that if Equations (4.1) to (4.4) is fulfilled, $\mathcal{C}^3$ continuity in the joints is achieved. For computational purposes, it is convenient to formulate these constraints as a system of linear equations on the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, given as:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_{1,i+1} \\ \boldsymbol{P}_{2,i+1} \\ \boldsymbol{P}_{3,i+1} \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{P}_{n,i} - \boldsymbol{P}_{n-1,i} \\ -2\boldsymbol{P}_{n-1,i} + \boldsymbol{P}_{n-2,i} \\ 2\boldsymbol{P}_{n,i} - 3\boldsymbol{P}_{n-1,i} + 3\boldsymbol{P}_{n-2,i} - \boldsymbol{P}_{n-3,i} \end{bmatrix} \tag{4.5}$$

It is easy to verify that $\text{rank}(\boldsymbol{A}) = 3$, is equal to the number of columns in $\boldsymbol{A}$ and thus have a unique solution.

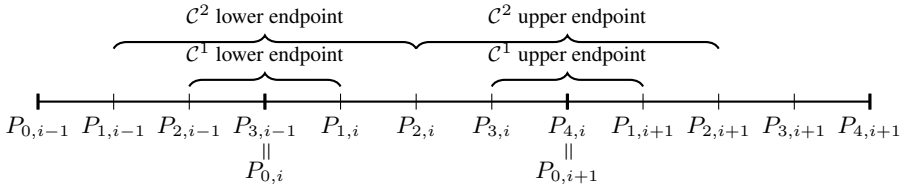


Figure 4.1: A cubic Bézier spline of three segments and its control points are shown. The joints are visualized with a thick mark. One can see that for $\mathcal{C}^1$ continuity, there are only constraints on the neighboring points, and we are thus able to specify the derivative freely. But for $\mathcal{C}^2$ continuity, $P_{2,i}$ is a part of the constraint for both the lower and upper endpoints. This means we are no longer able to freely set the second derivative without the influence of the constraints from the other endpoint, and thus not a part of $\mathcal{D}^2$ class.

4.1.1 Cubic and Quintic Bézier Spline

A Bézier spline stitched together by cubic curves is $\mathcal{C}^3$ and can freely set the first derivative in each endpoint, thus in $\mathcal{D}^1$ class. However, it is not able to set the second and third-order derivative freely in the endpoints; thus, we are not able to represent complex shapes that have the restriction of being $\mathcal{C}^3$. See Figure 4.1 for a visualization. The same reasoning applies to the quintic Bézier curve, where we are not able to set the third derivative freely.

4.1.2 Septic Bézier Spline

From Definition 2.1, a septic Bézier curve is in class $\mathcal{C}^7$. Also, we can set the third derivatives in the endpoints freely. See Example 4.1 for how this can be done. This means that it is a part of $\mathcal{D}^3$ class, and we can have $\mathcal{C}^3$ continuity in the joints. Thus, it is concluded that a septic Bézier curve is the curve of the lowest degree that fulfills our requirements.

Table 4.1: Bézier curve of different orders and desired properties for stepwise path generation. $\mathcal{C}^n$ is fulfilled according to Definition 2.1, while $\mathcal{D}^n$ is fulfilled according to Definition 2.3.

Property \ Degree	**Cubic** ($n = 3$)	**Quintic** ($n = 5$)	**Septic** ($n = 7$)
$\mathcal{C}^3$	✔	✔	✔
$\mathcal{D}^1$	✔	✔	✔
$\mathcal{D}^2$	✗	✔	✔
$\mathcal{D}^3$	✗	✗	✔

4.2 The Path Generator

For a path-following motion control scenario, it is essential to decide upon what constitutes a "better" path before designing a path generation algorithm. For a surface vessel treated here, this can vary a lot depending on the overall task environment and what properties are deemed more valuable. For some missions, the vessel must converge and stay on the exact path, whereas for other missions, it is more important to find a path that minimizes length, energy consumed, or time spent. There is a vast amount of literature that has proposed different solutions, each of them satisfying different priorities.

For example, the famous result of Dubins (1957) showed that the shortest path (minimum time) between two configurations (x, y, ψ) of a craft moving at constant speed U is a path formed by straight lines and circular arc segments. However, Dubins path is not the shortest in case of Zermelo's navigation problem[1], since straight lines are not optimal anymore. Secondly, Dubin's path is not $\mathcal{C}^2$ continuous, since it is constructed by straight lines which by definition has curvature $\kappa = 0$, and circular arcs with curvature $\kappa = 1/R$, defined in Section 2.2.2. Thus, there is a discontinuity in the concatenation points, which causes Dubins path to be unfeasible in a maneuvering problem design for a 3 DOF vessel. From this, one can conclude that there

[1]Zermelo's navigation problem is the task of finding the optimal trajectory that minimizes the travel time in case of environmental forces acting on the vessel.

does not exist one clear definition of what constitutes the optimal path between two waypoints.

First and foremost, a reasonable path must be feasible with respect to the dynamical constraints imposed by the vessel. Furthermore, for a low-speed vessel such as ReVolt, a reasonable path between two waypoints could be a path where the traveled distance/energy consumption is minimized. At the same time, it is desired to keep the cruising speed throughout the curve as good as possible. Further, this implies that it is desired to avoid sharp turns. These properties seem reasonable for a large container vessel such as ReVolt since it would be energy-consuming to increase/decrease the speed very often, as well as undesired to introduce more lateral acceleration than necessary due to fragile cargo, for instance.

Proposition 4.1. *A reasonable path for a low-speed vessel such as ReVolt, is one where the traveled distance/energy consumed is minimized, and at all time is feasible with respect to the dynamical constraints of the vessel.*

4.2.1 Strategy on the Next Waypoint

By assuming we only know a certain amount of waypoints ahead, a strategy has to be defined for which heading the vessel should have when approaching the next waypoint. If we only know one waypoint ahead, a reasonable strategy would be to use the angle given by the current and the next waypoint. Then the heading at the next waypoint is given as:

$$\psi_{k+1} = \text{atan2}(y_{k+1} - y_k, x_{k+1} - x_k), \quad (4.6)$$

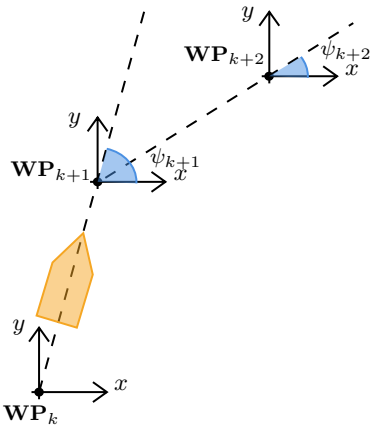


Figure 4.2: Three waypoints with the given heading strategy at the following waypoints. ψ_k denotes the heading at waypoint number k.

where atan2(y, x) is the four-quadrant version of arctan(y/x). See Figure 4.2 for a visualization. Further, it is convenient to have zero curvature and rate of change in curvature such that the vessel is equally ready for turning in both directions. By placing the control points on a straight line with an angle defined by Equation (4.6) through the given waypoint, one can specify the given heading, as well as achieve zero curvature and rate of change in curvature through the waypoint. This was indicated by Marley (2019), and we can indeed demonstrate that this is correct by the following example.

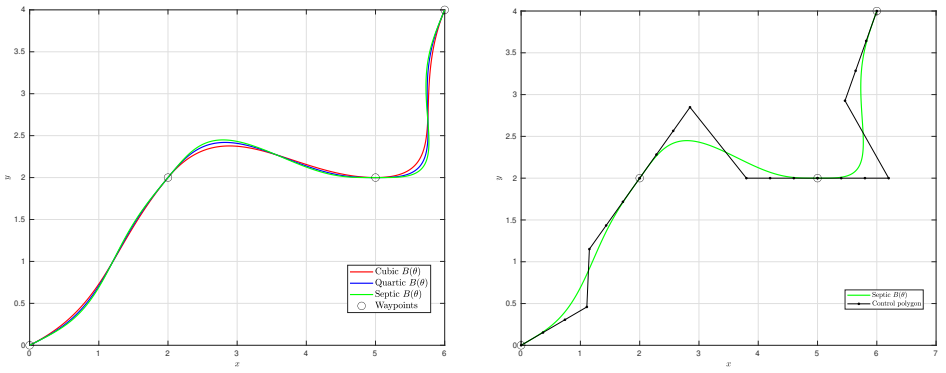
Example 4.1. *Given a septic Bézier curve, initial heading $\psi_0 = \pi/8$, the following waypoints: $\{(0,0),(2,2),(5,2),(6,4)\}$, and the assumption that we only know one*

waypoint ahead. We want zero curvature and rate of change in curvature, as well as specify the heading through the waypoints.

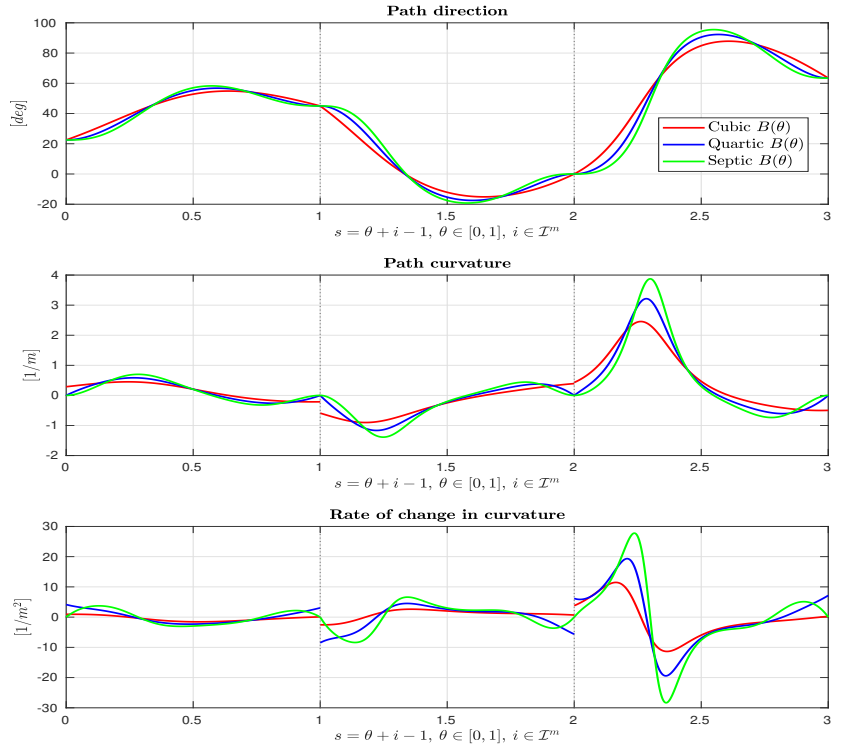
We start off by placing $\boldsymbol{P}_{4,i}$*,* $\boldsymbol{P}_{5,i}$*, and* $\boldsymbol{P}_{6,i}$ *on a straight line with an angle defined by Equation* (4.6)*. For simplicity we can place the control points at equal distance.* $\boldsymbol{P}_{1,i+1}$*,* $\boldsymbol{P}_{2,i+1}$*, and* $\boldsymbol{P}_{3,i+1}$ *is then given by Equation* (4.5)*. See Figure 4.3b for a visualization. The control points obtained for each segment is given in Appendix B. Using the constraints defined in Equations* (4.1) *to* (4.4)*, we can verify if* C^3 *continuity in the joints are achieved. Considering e.g. waypoint* $(5, 2)$*:*

$$
\begin{aligned}
\textit{eq. } (4.1): \quad (5,2) &= (5,2).\\
\textit{eq. } (4.2): \quad (5.4,2) &= -2\cdot(5,2)+(4.6,2),\\
(10,2) &= (10,2).\\
\textit{eq. } (4.3): \quad (5.8,2)-2\cdot(5.4,2) &= (4.2,2)-2\cdot(4.6,2),\\
(-5,-2) &= (-5,-2).\\
\textit{eq. } (4.4): \quad 3\cdot(5.4,2)-3\cdot(5.8,2)+(6.2,2) &= 2\cdot(5,2)-3\cdot(4.6,2)\\
&\quad +3\cdot(4.2,2)-(3.8,2),\\
(10,2) &= (10,2).
\end{aligned}
$$

All constraints are satisfied. By further calculating the Bézier curve and its derivatives defined by the control points, one can find the direction, curvature, and rate of change in curvature from Equations (2.21)*,* (2.23)*, and* (4.6)*. Checking the concatenation points, one will indeed find that* $\kappa = \tau = 0$*, and the path tangential* $\gamma_d = \psi_{k+1}$*, Q.E.D. This is depicted in Figure 4.3c. A plot of the derivatives is found in Appendix B.*



(a) **(b)**



(c)

Figure 4.3: In **(a)** a cubic, quartic, and a septic Bézier spline consisting of three segments defined by the control points in Appendix B is visualized. The septic Bézier spline's control points are visualized in **(b)**. In **(c)** the direction, curvature, and rate of change in curvature is plotted. The strategy for heading, curvature, and rate of change in curvature in the waypoints is described in Section 4.2.1.

4.2.2 Corridor

For safety reasons, a restriction on the area between the current and next waypoint where the curve is allowed to be should be imposed. We may assume that a straight-line corridor, within some path-transversal distance bound from current to next waypoint, is collision-free for static obstacles. The vessel is then freely allowed to move inside the given corridor, and the path can be constructed within. A visualization is given in Figure 4.4. Here, each path segment is constructed inside its associated corridor G_i.

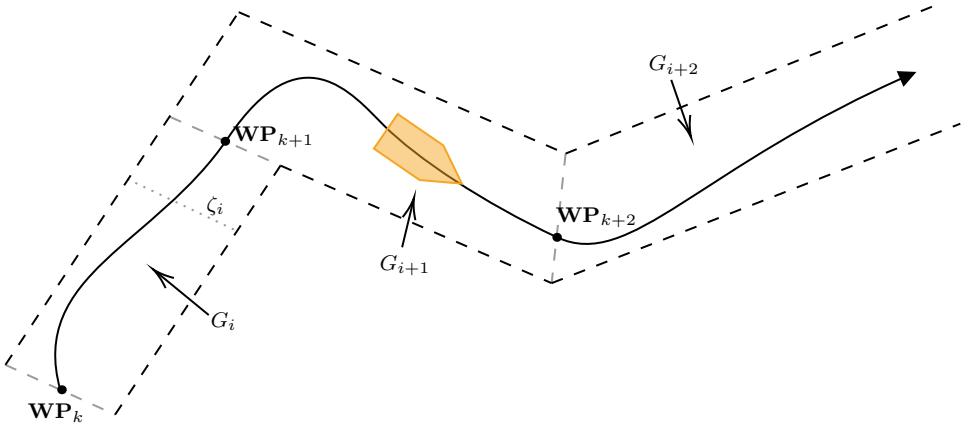


Figure 4.4: Three waypoints are visualized with a straight-line corridor G_i with width ζ_i defined between each waypoint. The path must be contained inside the corridor.

Corridor Constraints

To ensure that the path is always inside a given corridor with width ζ_i, we take advantage of the convex hull property in Section 2.3.3. Based on the strategy for the next waypoint in Section 4.2.1, the placement of the control points $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$ for the next path segment will lie on a straight line defined by Equation (4.6). By assuming the "worst-case" scenario, a $\pm 90^\circ$ turn between previous, current, and next waypoint, these control points are placed perpendicular to the wall. See Figure 4.5 for a visualization.

By placing the control points inside the corridor, the path will by definition be inside the corridor, by the convex hull property. From Section 4.1, we know that $\boldsymbol{P}_{4,i}$, $\boldsymbol{P}_{5,i}$, $\boldsymbol{P}_{6,i}$, and $\boldsymbol{P}_{7,i}$ uniquely determines the placement of $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$. Thus, we can set constraints on the former once, to ensure that the latter once never exceeds half the corridor width $\zeta_{i+1}/2$.

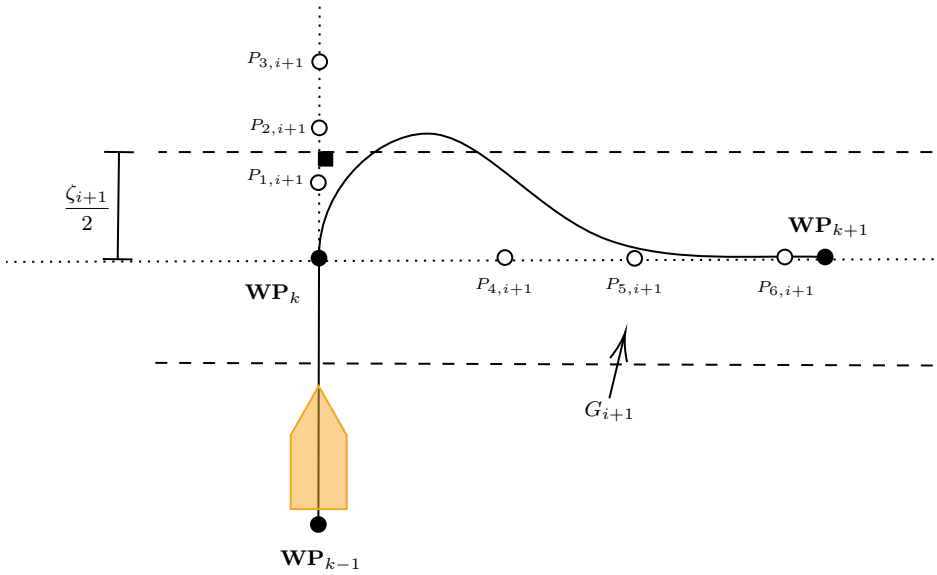


Figure 4.5: Three waypoints forming a 90° turn are visualized. A straight-line corridor G_i with width ζ_i are shown between $\mathbf{WP}_k$ and $\mathbf{WP}_{k+1}$. By placing the control points inside the corridor, the path will by definition be inside the corridor, from the convex hull property.

4.2.3 Replanning

If a better path is found by the path planner, or in an emergency where the ship is about to collide with a newly discovered object, it is critical to be able to replan immediately. See Figure 4.6. A new path needs to be generated from where the vessel is located right now to a new waypoint. By taking advantage of that $\boldsymbol{B}_{1\times 2}(\theta)$ and its derivatives is known for all θ, one can solve the "inverse problem" by finding the control points $\boldsymbol{P}_{0,i+1}$, $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$, which makes the path $\mathcal{C}^3$ for a given θ. Let us say that we want to replan at $\theta = \theta_x$. By taking advantage of the properties of the Bézier curve and its derivatives, we can find the control points that makes the new joint at θ_x $\mathcal{C}^3$ continuous, by solving the following system of equations:

$$\boldsymbol{B}_{1\times 2}(\theta_x) = \boldsymbol{P}_{0,i+1}, \tag{4.7}$$

$$\boldsymbol{B}^{\theta}_{1\times 2}(\theta_x) = n(\boldsymbol{P}_{1,i+1} - \boldsymbol{P}_{0,i+1}), \tag{4.8}$$

$$\boldsymbol{B}^{\theta^2}_{1\times 2}(\theta_x) = n(n-1)(\boldsymbol{P}_{2,i+1} - 2\boldsymbol{P}_{1,i+1} + \boldsymbol{P}_{0,i+1}), \tag{4.9}$$

$$\boldsymbol{B}^{\theta^3}_{1\times 2}(\theta_x) = n(n-1)(n-2)(\boldsymbol{P}_{3,i+1} - 3\boldsymbol{P}_{2,i+1} + 3\boldsymbol{P}_{1,i+1} + \boldsymbol{P}_{0,i+1}). \tag{4.10}$$

This can be formulated as system of linear equations on the form $\boldsymbol{Ax} = \boldsymbol{b}$. For a septic Bézier curve, it becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -7 & 7 & 0 & 0 \\ 42 & -84 & 0 & 0 \\ -210 & 630 & -630 & 210 \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_{0,i+1} \\ \boldsymbol{P}_{1,i+1} \\ \boldsymbol{P}_{2,i+1} \\ \boldsymbol{P}_{3,i+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_{1\times 2}(\theta_x) \\ \boldsymbol{B}^{\theta}_{1\times 2}(\theta_x) \\ \boldsymbol{B}^{\theta^2}_{1\times 2}(\theta_x) \\ \boldsymbol{B}^{\theta^3}_{1\times 2}(\theta_x) \end{bmatrix}. \tag{4.11}$$

Note that one must solve two systems of linear equations; one for the x- and one for the y-coordinates, or one could rotate into a "path-fixed" reference frame using Equation (4.17), solve one system of equations, and then rotate back. An important thing to note is that control point $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$ of the new segment will not necessarily be on a straight line since the curvature and rate of change in curvature may not be zero at θ_x.

For the replanner to work in conjunction with a maneuvering controller a mapping function $f : s \to (\theta, i, \theta_x)$ must be defined.

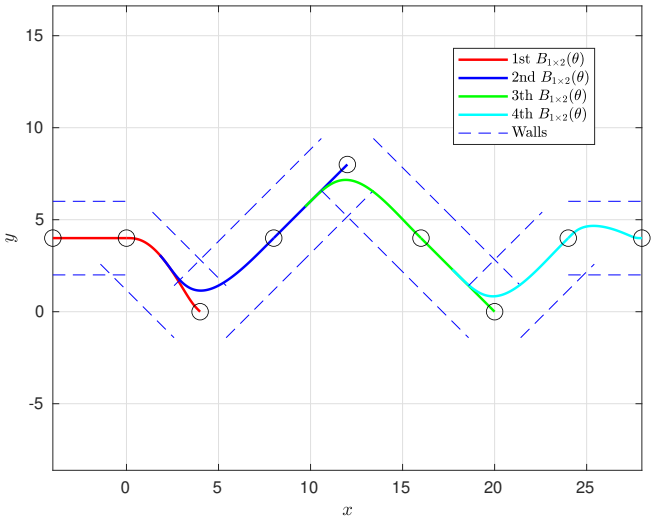


Figure 4.6: A Bézier curve which has been replanned three times. Each color indicates a new generated curve.

4.2.4 Pragmatic Approach

A pragmatic solution where the control points of the Bézier curve are placed "by hand" in a proper manner, has the advantage of being obviously explicit and computationally efficient. Based on Proposition 4.1 and the strategy presented in Section 4.2.1, a pragmatic solution is established.

Placement of Control Points

Based on the constraints for $\mathcal{C}^3$ continuity in Section 4.1, the placement of control points $\boldsymbol{P}_{0,i+1}$, $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$ of next segment, is uniquely determined by the placement of the control points $\boldsymbol{P}_{4,i}$, $\boldsymbol{P}_{5,i}$, $\boldsymbol{P}_{6,i}$, and $\boldsymbol{P}_{7,i}$ from the previous segment. $\boldsymbol{P}_{7,i+1}$ is determined by the next waypoint. Only control point $\boldsymbol{P}_{4,i}$, $\boldsymbol{P}_{5,i}$, and $\boldsymbol{P}_{6,i}$ need to be placed "by hand".

From practical observations, a suitable combination based on Proposition 4.1, is obtained by placing $\boldsymbol{P}_{4,i}$ a distance δ away from $\mathbf{WP}_{k+1}$ ($= \boldsymbol{P}_{7,i}$), while $\boldsymbol{P}_{5,i}$ and $\boldsymbol{P}_{6,i}$ close to $\mathbf{WP}_{k+1}$. This keeps the arc length as well as the curvature low. A trade-off must be done; placing the waypoints too close causes high curvature while placing them too far away results in big turns. The distance can be tuned by a scaling factor μ, which is coupled to δ. See Figure 4.7a for a visualization.

The control points are thus placed as:

$$\boldsymbol{P}_{4,i} = \mathbf{WP}_{k+1} - \delta \left[\cos(\psi_{k+1}), \sin(\psi_{k+1})\right], \tag{4.12}$$

$$\boldsymbol{P}_{5,i} = \mathbf{WP}_{k+1} - \frac{\delta}{\mu} \left[\cos(\psi_{k+1}), \sin(\psi_{k+1})\right], \tag{4.13}$$

$$\boldsymbol{P}_{6,i} = \mathbf{WP}_{k+1} - \frac{\delta}{2\mu} \left[\cos(\psi_{k+1}), \sin(\psi_{k+1})\right], \tag{4.14}$$

where ψ_{i+1} is given by Equation (4.6). This corresponds to the function **place-cp** in Algorithm 1. A rule for calculating δ needs to be established. Using Equation (4.5), one must eventually calculate and evaluate the placement of $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$.

Avoiding Impractical Behavior

It is never reasonable to perform maneuvers leading the ship further away from the next waypoint. See figure Figure 4.7b for a visualization. This can occur if the control points of the current segment are placed beyond the next waypoint. A mathematical derivation for when this will happen is hard to establish, but it can always be avoided by placing the control points such that "overlapping" never happens. A simple rule for avoiding this is to never place the control points further away than half the distance from next to the current waypoint. The maximum distance is found as:

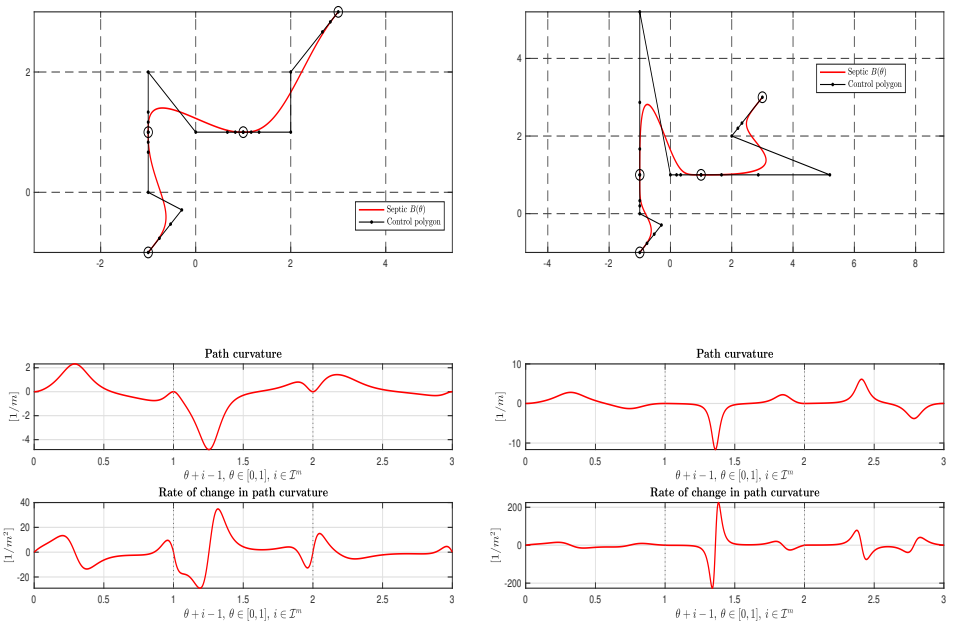
$$\delta_{max} = \frac{|\mathbf{WP}_{k+1} - \mathbf{WP}_k|}{2}. \tag{4.15}$$

Comply to Dynamical Constraints

From the convex hull and variation diminishing property, we know that placing the control points close to the waypoints results in a short path. Actually, by placing

the control points infinitely close, the resulting Bézier curve becomes a straight line between the waypoints. However, in return, one gets infinitely high curvature. A distance $\delta_{min}\left(\kappa_{max}, u_d(t)\right)$ can be established as a function of the curvature constraint imposed by the ship dynamics and the chosen speed $u_d(t)$. δ_{min} represetns then the minimum distance from $\boldsymbol{P}_{4,i}$ to $\boldsymbol{P}_{7,i}$, which assures that the dynamical constraints are not violated.

A minimum and a maximum distance for δ is now established. As long as δ_{min} is lower than δ_{max}, $\boldsymbol{P}_{4,i}$ is placed δ_{min} away. If δ_{min} is higher than δ_{max}, $\boldsymbol{P}_{4,i}$ is placed δ_{max} away. The latter results in a higher path curvature than allowed at the given speed, thus the speed must be reduced. A pseudo-algorithm is given in Algorithm 1.



(a) **(b)**

Figure 4.7: Two Bézier curves with same waypoints $(0,-1),(0,1),(1,1),(3,3)$, but different placement of control points is shown. In (**a**) the control points $\boldsymbol{P}_{5,i}$ $\boldsymbol{P}_{6,i}$ is placed sufficiently close to $\boldsymbol{P}_{7,i}$ (next waypoint). In (**b**) the control points $\boldsymbol{P}_{5,i}$ $\boldsymbol{P}_{6,i}$ is placed close to $\boldsymbol{P}_{4,i}$. As seen from the plots of path curvature and rate of change in path curvature, the design in (**a**) achieves much better results compared to (**b**), based on Proposition 4.1.

Algorithm 1: Pragmatic Placement of Control Points

input: $\mathbf{WP}_{current}$, $\mathbf{WP}_{next}$, δ_{min}, δ_{max}, μ
if $\delta_{min} < \delta_{max}$ **then**
  $\delta \leftarrow \delta_{min}$;
else
  `// Reduce Speed`
  $\delta \leftarrow \delta_{max}$;
$\psi_{next} \leftarrow$ `CalculateHeading`$(\mathbf{WP}_{current}, \mathbf{WP}_{next})$;
$\boldsymbol{x} \leftarrow$ `PlaceControlPoints`$(\mathbf{WP}_{current}, \mathbf{WP}_{next}, \delta, \mu, \psi_{next})$;
return $\boldsymbol{x}$;

Discussion

As mentioned, the pragmatic solution does not suffer from computational complexity and is very efficient. It is not optimal based on Proposition 4.1, but can give good results if tuned properly. The drawback of the approach is that the control points need to be placed "by hand" to meet the imposed constraints. The constraints are first and foremost the corridor constraints, and the maximum curvature allowed for a given speed, but one must also take into account the distance between the waypoints. The tuning is not necessarily straight forward, but a stepwise procedure would be:

1. Find κ_{max} for the given speed $u_d(t)$.
2. Find δ_{min} $(\kappa_{max}, u_d(t))$.
3. Tune μ based on δ_{min}.

4.2.5 Optimization Approach

A common solution to path generation is to apply optimization techniques. This method allows an objective function to be specified where, e.g., minimum time, energy, and distance are design goals. The idea is: For each hybrid path segment, an optimization routine can be performed to find the optimal path. A general optimization problem for path planning and generation can be formulated as in Fossen (2011a):

$$
\begin{aligned}
J = \min_{\boldsymbol{\chi}} \{ f(\boldsymbol{\chi}) \} & \\
\text{subject to} \qquad g_k(\boldsymbol{\chi}) &\preceq 0, \quad (k = 1, \ldots, n_g)\,, \\
h_j(\boldsymbol{\chi}) &= 0, \quad (j = 1, \ldots, n_h)\,, \\
\chi_{i,\min} \leq \chi_i &\leq \chi_{i,\max}, \quad (i = 1, \ldots, n_x)\,,
\end{aligned}
\tag{4.16}
$$

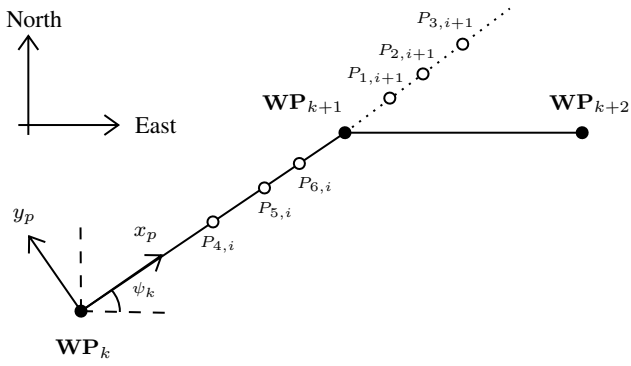


Figure 4.8: A path-fixed reference frame with origin in $\mathbf{WP}_k^n$ rotated by a an angle ψ_k relative to the NED-frame.

where $f(\boldsymbol{\chi})$ should be minimized with respect to the decision variables in the parameter vector $\boldsymbol{\chi}$. $g_k(\boldsymbol{\chi})$ and $h_j(\boldsymbol{\chi})$ are linear or nonlinear inequality and equality constraints, respectively. In the following section, an optimization problem based on the Bézier curve will be formulated for optimally placing the control points for each segment.

Path-Fixed Reference Frame

Consider a straight-line path defined by two waypoints $\mathbf{WP}_k^n = [x_k, y_k]^\top$ and $\mathbf{WP}_{k+1}^n = [x_{k+1}, y_{k+1}]^\top$, respectively. Here, subscript n refers to the NED-frame. Also, consider a path-fixed reference frame with origin in $\mathbf{WP}_k^n$, whose x-axis has been rotated by an angle ψ_k, defined in Equation (4.6), relative to the x_n-axis. Hence, the coordinates of a point $\boldsymbol{P}^n$ in the path-fixed reference frame can be computed as:

$$\boldsymbol{P}^p = \boldsymbol{R}(\psi_k)^\top (\boldsymbol{P}^n - \mathbf{WP}_k^n), \tag{4.17}$$

where subscript p denotes a point relative to the path-fixed reference frame, and:

$$\boldsymbol{R}(\psi_k) \triangleq \begin{bmatrix} \cos\psi_k & -\sin\psi_k \\ \sin\psi_k & \cos\psi_k \end{bmatrix} \in SO(2). \tag{4.18}$$

See Figure 4.8. By transforming the problem into a path-fixed reference frame and treating $\mathbf{WP}_k^n$ as origin, the problem of placing the control points simplifies to a one-dimensional problem along the x_p-axis in the "path-frame".

Decision Variables

The shape of the Bézier curve is uniquely determined by the placement of the control points and will thus serve as our decision variables. The first and the last control

point of the Bézier curve is defined by current and next waypoint, thus not part of the decision variables. As for the pragmatic approach, the constraints for $\mathcal{C}^3$ continuity in the joints, control point $\boldsymbol{P}_{0,i+1}$, $\boldsymbol{P}_{1,i+1}$, $\boldsymbol{P}_{2,i+1}$, and $\boldsymbol{P}_{3,i+1}$ of the next segment, is uniquely determined by the system of linear equations in Equation (4.5). $\boldsymbol{P}_{7,i+1}$ is determined by the next waypoint. Thus we are left with $\boldsymbol{P}_{4,i+1}$, $\boldsymbol{P}_{5,i+1}$, and $\boldsymbol{P}_{6,i+1}$.

As mentioned, by formulating the problem in the path-frame we are facing a one-dimensional problem since the y_p-coordinates will be zero. Thus, the decision variables are only the x_p-coordinates of the control points. For convenience we call the decision variables χ_1, χ_2, and χ_3 such that:

$$\boldsymbol{\chi}_{3\times 1}^{\top} = \begin{bmatrix} x_{4,i} & x_{5,i} & x_{6,i} \end{bmatrix} = \begin{bmatrix} \chi_1 & \chi_2 & \chi_3 \end{bmatrix}. \tag{4.19}$$

The Bézier curve becomes:

$$\boldsymbol{B}_{1\times 1}(\theta) = \boldsymbol{a}_{1\times 8}^{\top}(\theta)\boldsymbol{M}_{8\times 8}\boldsymbol{x}_{8\times 1}, \tag{4.20}$$

where the monomial basis vector, $\boldsymbol{a}_{1\times 8}^{\top}(\theta)$, and the spline matrix, $\boldsymbol{M}_{8\times 8}$, is the same as in Example 2.1, and:

$$\boldsymbol{x}_{8\times 1}^{\top} = \begin{bmatrix} x_{0,i} & x_{1,i} & x_{2,i} & x_{3,i} & \boldsymbol{\chi}_{3\times 1}^{\top} & x_{7,i} \end{bmatrix}. \tag{4.21}$$

Objective Function

Following Proposition 4.1, we want to minimize the traveled distance/energy consumed of the path. For our purpose, the traveled distance corresponds to the arc length of the path, defined in Equation (2.19). The objective function for one path segment is then to minimize the arc length. For convenience, we minimize the square:

$$J = \min_{\boldsymbol{\chi}_{3\times 1}} \int_0^1 \left| \boldsymbol{B}_{1\times 1}^{\theta}(\theta) \right|^2 d\theta. \tag{4.22}$$

Noting that the Bézier curve becomes as in Equation (4.20), it can be expressed as the inner product of the vector and itself:

$$J = \min_{\boldsymbol{\chi}_{3\times 1}} \int_0^1 \boldsymbol{B}_{1\times 1}^{\theta}(\theta)^{\top} \boldsymbol{B}_{1\times 1}^{\theta}(\theta)\, d\theta. \tag{4.23}$$

Plugging in Equation (4.20), the objective function can be written as:

$$J = \min_{\boldsymbol{\chi}_{3\times 1}} \int_0^1 (\boldsymbol{a}_{1\times 8}^{\theta}(\theta)^{\top}\boldsymbol{M}_{8\times 8}\boldsymbol{x}_{8\times 1})^{\top} \boldsymbol{a}_{1\times 8}^{\theta}(\theta)^{\top}\boldsymbol{M}_{8\times 8}\boldsymbol{x}_{8\times 1}\, d\theta \tag{4.24}$$

$$= \min_{\boldsymbol{\chi}_{3\times 1}} \int_0^1 \boldsymbol{x}_{8\times 1}^{\top}\boldsymbol{M}_{8\times 8}^{\top}\boldsymbol{a}_{1\times 8}(\theta)\boldsymbol{a}_{1\times 8}(\theta)^{\top}\boldsymbol{M}_{8\times 8}\boldsymbol{x}_{8\times 1}\, d\theta \tag{4.25}$$

$$= \min_{\boldsymbol{\chi}_{3\times 1}} \boldsymbol{x}_{8\times 1}^{\top} \underbrace{\boldsymbol{M}_{8\times 8}^{\top} \int_0^1 \boldsymbol{a}_{1\times 8}^{\theta}(\theta)\boldsymbol{a}_{1\times 8}^{\theta}(\theta)^{\top}\, d\theta\, \boldsymbol{M}_{8\times 8}}_{\boldsymbol{W}_{8\times 8}} \boldsymbol{x}_{8\times 1}. \tag{4.26}$$

We define the integral of the the product of the derivative monomial basis vector as:

$$c \triangleq \int_0^1 \boldsymbol{a}^{\theta}_{1\times 8}(\theta)\boldsymbol{a}^{\theta}_{1\times 8}(\theta)^{\top}\, d\theta. \tag{4.27}$$

Since c is a constant, it will not contribute to the objective function[2]. $\boldsymbol{W}_{8\times 8}$ is then found by calculating the product of the lower triangular matrix $\boldsymbol{M}_{8\times 8}$ and its conjugate transpose:

$$\boldsymbol{W}_{8\times 8} = \boldsymbol{M}^{\top}_{8\times 8}\, c\, \boldsymbol{M}_{8\times 8}. \tag{4.28}$$

$\boldsymbol{M}_{8\times 8}$ will in fact be the *Cholesky decomposition* of $\boldsymbol{W}_{8\times 8}$[3]. This means that $\boldsymbol{W}_{8\times 8}$ is a symmetric positive-definite matrix, and we are left with a strictly convex objective function in quadratic form:

$$J = \min_{\boldsymbol{\chi}_{3\times 1}} \boldsymbol{x}^{\top}_{8\times 1}\, \boldsymbol{W}_{8\times 8}\, \boldsymbol{x}_{8\times 1}. \tag{4.29}$$

Note that the objective function now contains the vector of all x_p-coordinates, but only χ_1, χ_2, and χ_3 ($= x_{4,i}, x_{5,i}$, and $x_{6,i}$) are decision variables. The rest of the x_p-coordinates are known constants. One can further rearrange and simplify to obtain the expression of a general objective function for a quadratic programming problem:

$$J = \min_{\boldsymbol{\chi}_{3\times 1}} \boldsymbol{\chi}^{\top}_{3\times 1}\, \boldsymbol{Q}_{3\times 3}\, \boldsymbol{\chi}_{3\times 1} + \boldsymbol{q}^{\top}_{3\times 1}\, \boldsymbol{\chi}_{3\times 1}. \tag{4.30}$$

Here, $\boldsymbol{\chi}_{3\times 1}$ is given by Equation (4.19), $\boldsymbol{Q}_{3\times 3} = \boldsymbol{W}_{8\times 8}[5,6,7;5,6,7]$ (submatrix of $\boldsymbol{W}_{8\times 8}$), and $\boldsymbol{q}_{3\times 1}$ contains all linear terms found by expanding Equation (4.29). All constant terms are left out, since they do not contribute to the solution.

Constraints

The constraints must be formulated such that they are dependent on the decision variables in $\boldsymbol{\chi}_{3\times 1}$. Since the origin is located in $\mathbf{WP}_k$, the decision variables must be greater than zero and less than the distance to the next waypoint. These constraints will be lower and upper bound for the optimization problem. One must also assure that the decision variables are arranged, such that, e.g., χ_3 does not appear before χ_1. See Figure 4.9. These are linear inequality constraints and can be expressed as:

$$0 \leq \chi_1 \leq \chi_2 \leq \chi_3 \leq x_{7,i}. \tag{4.31}$$

[2] $\int_0^1 \boldsymbol{a}^{\theta}_{1\times 8}(\theta)\boldsymbol{a}^{\theta}_{1\times 8}(\theta)^{\top}\, d\theta = \int_0^1 1 + 4\theta^2 + 9\theta^4 + 16\theta^6 + 25\theta^8 + 36\theta^{10} + 49\theta^{12}\, d\theta = 16.239.$

[3] The Cholesky decomposition of a Hermitian positive definite matrix $\boldsymbol{A}$ is a decomposition of the form $\boldsymbol{A} = \boldsymbol{L}^{\top}\boldsymbol{L}$, where $\boldsymbol{L}$ is a lower triangular matrix with real and positive diagonal entries, and $\boldsymbol{L}^{\top}$ denotes the conjugate transpose of $\boldsymbol{L}$. Every Hermitian positive definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition (Zhang et al., 2017).

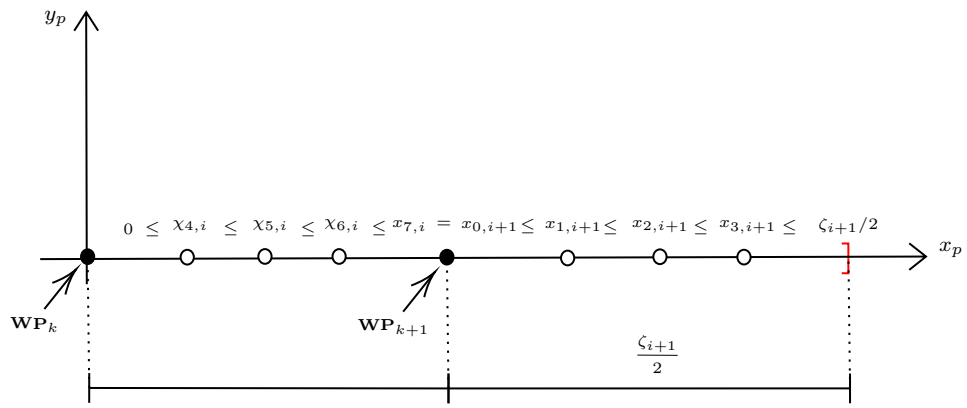


Figure 4.9: One-dimensional constraints for optimization problem in the path-fixed reference frame with origin in $\mathbf{WP}_k$. The red mark indicates the maximum allowed distance of placing the control points for the next segment.

To ensure that the path is always inside a given corridor with width ζ_{i+1}, we take advantage of the convex hull property of the Bézier curve and the relation between the control points given by Equation (4.5). For convenience, we assume that the corridor width is constant. It can be expressed as:

$$x_{1,i+1} \leq x_{7,i} + \frac{\zeta_{i+1}}{2}, \tag{4.32}$$

$$x_{2,i+1} \leq x_{7,i} + \frac{\zeta_{i+1}}{2}, \tag{4.33}$$

$$x_{3,i+1} \leq x_{7,i} + \frac{\zeta_{i+1}}{2}, \tag{4.34}$$

where $x_{7,i}$ is the coordinate of the next waypoint. See Figure 4.9. By using Equation (4.5) and manipulating the constraints, they can be expressed as:

$$-\chi_3 \leq -x_{7,i} + \frac{\zeta_{i+1}}{2}, \tag{4.35}$$

$$\chi_2 - 4\chi_3 \leq -3x_{7,i} + \frac{\zeta_{i+1}}{2}, \tag{4.36}$$

$$-\chi_1 + 6\chi_2 - 12\chi_3 \leq -7x_{7,i} + \frac{\zeta_{i+1}}{2}. \tag{4.37}$$

One must also assure that these control points are arranged and greater than $x_{7,i}$. See Figure 4.9. It can be expressed as:

$$x_{7,i} \leq x_{1,i+1} \leq x_{2,i+1} \leq x_{3,i+1}. \tag{4.38}$$

Again, we can use Equation (4.5) and manipulate the constraints. To express that all control points should be greater than $x_{7,i}$, we can use the relations found in Equations (4.35) to (4.37) by flipping the inequality sign and setting the corridor term equal to zero. Lastly, we add the following constraints to ensure that these control points are arranged:

$$-\chi_2 + 3\chi_3 \leq 2x_{7,i}, \tag{4.39}$$
$$\chi_1 - 5\chi_2 + 8\chi_3 \leq 4x_{7,i}. \tag{4.40}$$

Keeping the Curvature Low

In order to respect the dynamical constraints imposed by the vessel, it is vital to keep the curvature low. The curvature for a Bézier curve is:

$$\kappa(\theta) = \frac{\left|\boldsymbol{B}^{\theta}_{1\times 2}(\theta) \times \boldsymbol{B}^{\theta^2}_{1\times 2}(\theta)\right|}{\left|\boldsymbol{B}^{\theta}_{1\times 2}(\theta)\right|^3} = \frac{\left|x^{\theta}_d(\theta) y^{\theta^2}_d(\theta) - x^{\theta^2}_d(\theta) y^{\theta}_d(\theta)\right|}{\left(x^{\theta}_d(\theta)^2 + y^{\theta}_d(\theta)^2\right)^{3/2}}. \tag{4.41}$$

Imposing that $\kappa(\theta) \leq \kappa_{max}$, where κ_{max} is specified by the user, results in a highly nonlinear constraint and a hard optimization problem to solve efficiently. Noting that the curvature has its peak right after a given waypoint, the constraints given by Equations (4.31) and (4.38) can be tuned, to smooth out sharp turns:

- Increasing the lower bound given in Equation (4.31) will help smooth out the sharp turn that can arise after a waypoint. For instance, setting the lower bound to $x_{7,i}/2$ is a decent choice.

- The constraint that the control points must be greater than $x_{7,i}$ expressed in Equation (4.38), can be tuned to avoid sharp turns. We introduce three tuning variables $\varepsilon_1, \varepsilon_2$, and ε_3, and forces $x_{1,i+1}$, $x_{2,i+1}$ and $x_{3,i+1}$ to be greater than $x_{7,i} + \varepsilon_i$:

$$\chi_3 \leq x_{7,i} - \varepsilon_1, \tag{4.42}$$
$$-\chi_2 + 4\chi_3 \leq 3x_{7,i} - \varepsilon_2, \tag{4.43}$$
$$\chi_1 - 6\chi_2 + 12\chi_3 \leq 7x_{7,i} - \varepsilon_3. \tag{4.44}$$

An idea can also be to use the replanner in Section 4.2.3 to construct smoother paths if the waypoint after the next waypoint is assumed to be known a certain distance before the next waypoint. See Figure 4.6.

Optimization Problem

Based on the decision variables, objective function, and constraints formulated, we can state our optimization problem for one path segment as a quadratic optimization problem on the form:

$$
\begin{aligned}
J = \min_{\boldsymbol{\chi}} \quad & \boldsymbol{\chi}_{3\times1}^{\top}\, \boldsymbol{Q}_{3\times3}\, \boldsymbol{\chi}_{3\times1} + \boldsymbol{q}_{3\times1}^{\top}\, \boldsymbol{\chi}_{3\times1} && \text{(4.45a)}\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{\chi}_{3\times1} \preceq \boldsymbol{b}, && \text{(4.45b)}\\
& \boldsymbol{\chi}_{3\times1} \succeq \boldsymbol{0}_{3\times1}, && \text{(4.45c)}\\
& \boldsymbol{\chi}_{3\times1} \preceq \boldsymbol{x}_{7,i}. && \text{(4.45d)}
\end{aligned}
$$

For our problem, the set of linear inequality constraints $\boldsymbol{A}\boldsymbol{\chi}_{3\times1} \preceq \boldsymbol{b}$ becomes:

$$
\begin{bmatrix}
1 & -1 & 0 \\
0 & 1 & -1 \\
0 & 0 & -1 \\
0 & 1 & -4 \\
-1 & 6 & -12 \\
0 & 0 & 1 \\
0 & -1 & 4 \\
1 & -6 & 12 \\
0 & -1 & 3 \\
1 & -5 & 8
\end{bmatrix}
\boldsymbol{\chi}_{3\times1} \preceq
\begin{bmatrix}
0 \\
0 \\
-x_{7,i} + \dfrac{\zeta}{2} \\
-3x_{7,i} + \dfrac{\zeta}{2} \\
-7x_{7,i} + \dfrac{\zeta}{2} \\
x_{7,i} - \varepsilon_1 \\
3x_{7,i} - \varepsilon_2 \\
7x_{7,i} - \varepsilon_3 \\
2x_{7,i} \\
4x_{7,i}
\end{bmatrix}. \tag{4.46}
$$

Discussion

Compared to the pragmatic solution, the control points are placed optimally according to the objective function. This is a huge advantage over the pragmatic solution because one can then specify the corridor width ζ directly, rather than going the opposite way.

4.3 Speed Assignment

The speed assignment is based on Skjetne (2005). Following Section 3.3, the guidance system should generate the speed profile $v_s(t, s(t))$ and its respective derivatives. We let the desired path speed $u_d(t)$ (in $\mathrm{m/s}$) from the path planner be a com-

manded input speed. We have that:

$$\begin{aligned} |\dot{\boldsymbol{p}}_d(s(t))| =& \sqrt{x_d^s(s(t))^2 \dot{s}(t)^2 + y_d^s(s(t))^2 \dot{s}(t)^2} \\ =& \sqrt{x_d^s(s(t))^2 + y_d^s(s(t))^2}\, |v_s(s(t), t)| = |u_d(t)|\,, \end{aligned} \tag{4.47}$$

which must hold along the path. The speed assignment is thus, by definition:

$$v_s(t, s(t)) \triangleq \frac{u_d(t)}{\sqrt{x_d^s(s(t))^2 + y_d^s(s(t))^2}}. \tag{4.48}$$

Setting the commanded input $u_d(t) = 0\ \mathrm{m/s}$ will stop the vessel on the path, while setting $u_d(t) > 0\ \mathrm{m/s}$ will move the vessel in positive direction along the path and $u_d(t) < 0\ \mathrm{m/s}$ will move it in negative direction. This speed assignment can assure that feasibility along the path always is satisfied.

If we have a constant speed assignment along the path where u_d is the desired constant speed, a maximum constraint on curvature κ_{max} is imposed. It is found by the implicit relation established in Equation (2.11). Given this type of speed assignment, the path must be designed according to the constraint imposed by κ_{max} to assure feasibility along the path. On the other hand, if we have a path-varying speed assignment, such that the speed is automatically reduced according to the curvature, one can assure that feasibility is always satisfied along the path.

Chapter 5

Navigation System

This chapter addresses the problem of designing a navigation system comprised of a global low-resolution path planner together with a local dynamic high-resolution path planner, satisfying the objectives stated in Section 3.2. OGMs, introduced in Section 2.5, will serve as a foundation for both methods, and we will assume that the map is known prior to operation.

We separate local and global waypoints. Local waypoints, also referred to as nodes and states, are represented by x's: $(x_0, x_1, \ldots, x_k)$. Global waypoints are represented by $\mathbf{WP}$'s: $(\mathbf{WP}_0, \mathbf{WP}_1, \ldots, \mathbf{WP}_k)$. The Euclidean distance between two waypoints i and j is denoted $|x_i - x_j|$.

5.1 Local Planner - Rapidly Exploring Random Trees

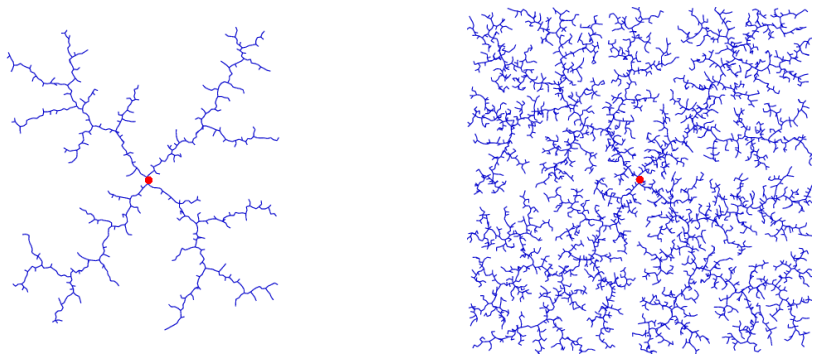
As our local planner, the Rapidly Exploring Random Trees (RRT), first introduced by LaValle (1998) will be considered. RRT aims to rapidly explore nonconvex high-dimensional configuration spaces by generating a tree of feasible paths subject to specified constraints. In its most basic version, the algorithm incrementally builds a directed tree $\mathcal{T} = (V, E)$[1] of feasible connected vertices V and edges E, rooted at the initial node x_0. In each iteration, a node $x_{rand} \in \mathcal{X}_{free}$ is sampled. An effort is made to connect the nearest vertex, $x_{nearest} \in V$, to x_{rand}. If a connection is made, $x_{nearest}$ is expanded such that x_{rand} is added to V, and the edge $(x_{nearest}, x_{rand})$ is added to E.

RRTs are constructed in a way that reduces the expected distance of a randomly chosen point to the tree. With a uniform sampling of $\mathcal{X}_{free}$, the probability of expanding a vertex is proportional to the size of its Voronoi region, introduced in Section 2.5.2. Since the largest Voronoi regions are in the frontier of the search, the three preferentially expands towards the largest unsearched regions of $\mathcal{X}_{free}$. Because of its expanding nature, the number of samples does not need to be specified a priori, and a solution is returned as soon as the tree built is dense enough. This enables the

[1] See e.g. Cormen et al. (2009), pp.246-253.

algorithm to handle real-time constraints.

Another great feature of RRT is the simplicity of adding constraints on how the tree expands. This makes RRT particularly suited for path planning problems that involve obstacles and differential constraints. As mentioned, the method is *probabilistically complete*, guaranteeing that the probability of finding a solution, if one exists, approaches one as the number of iterations approaches infinity. On the other hand, the method is not optimal because the existing state graph biases future expansion (Karaman and Frazzoli, 2011). This is solved by introducing *incremental rewiring* of the tree, a heuristic extension known as RRT*. New vertices are then not only added to the tree if the connection is feasible, but also considered to replace parents of existing nearby vertices in the tree by optimizing a specified cost function.



(a) **(b)**

Figure 5.1: A visualization of RRT on a square with dimensions $\mathcal{X} = [0, 100] \times [0, 100]$ and $x_{init} = (50, 50)$. In (**a**) the RRT quickly expands towards the largest Voronoi regions of the square. In (**b**) the RRT becomes more dense and breaks down the largest Voronoi regions. Adopted from LaValle (2005).

RRT* is *asymptotically optimal*, guaranteeing that an optimal path from the initial state to every state in the configuration space is found as the number of iterations approaches infinity. For a detailed explanation on how RRT* works refer to Karaman and Frazzoli (2010, 2011). RRT*, given in Algorithm 2, will serve as the foundation for the local path planner designed in this thesis. The primitive procedures used in RRT* is explained in Appendix C. In the following sections, we will combine useful aspects of different RRT variants proposed in the literature as well as modifying them to our purpose.

5.1.1 Constraints

The RRT* algorithm needs to be customized according to which vehicle type is considered. Several vehicles with different dynamics and DOF, such as automobiles,

Algorithm 2: RRT*

input: $x_0, x_{goal}, \mathcal{X}$
Initialize $\mathcal{T} = (V \leftarrow \{x_0\}, E = \emptyset)$;
for $i = 1, \ldots, n$ **do**
  $x_{rand} \leftarrow \texttt{Sample}_i$;
  $x_{nearest} \leftarrow \texttt{Nearest}(\mathcal{T}, x_{rand})$;
  $x_{new} \leftarrow \texttt{Steer}(x_{nearest}, x_{rand})$;
  if $\texttt{ObstacleFree}(x_{nearest}, x_{new})$ **then**
    $X_{near} \leftarrow \texttt{Near}(\mathcal{T}, x_{new}, r)$;
    $V \leftarrow V \cup \{x_{new}\}$;
    $x_{min} \leftarrow x_{nearest}$, $c_{min} \leftarrow \texttt{Cost}(x_{nearest})$;
    // Connect along minimum-cost path
    foreach $x_{near} \in X_{near}$ **do**
      if $\texttt{CollisionFree}(x_{near}, x_{new}) \wedge \texttt{Cost}(x_{near}) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$, $c_{min} \leftarrow \texttt{Cost}(x_{near})$;
    $E \leftarrow E \cup \{x_{min}\}$;
    // Rewire the tree
    foreach $x_{near} \in X_{near}$ **do**
      if $\texttt{CollisionFree}(x_{near}, x_{new}) \wedge \texttt{Cost}(x_{near}) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$, $c_{min} \leftarrow \texttt{Cost}(x_{near})$;
      $E \leftarrow E \setminus \{x_{parent}, x_{near}\} \cup \{x_{new}, x_{near}\}$;
return $\mathcal{T}$;

car-like robots, and spacecrafts, have been investigated in the literature. See for example Cheng et al. (2001). We design the constraints to fit our ASV.

Obstacles

In order to navigate safely from initial to target waypoint, the ASV must avoid both static and dynamic obstacles. Let d_{min} be the minimum distance to an obstacle allowed. Then, a constraint on minimum distance allowed is given as:

$$|\boldsymbol{p}(t) - \boldsymbol{o}_i(t)| \geq d_{min}, \quad \forall \boldsymbol{o}_i \in \mathcal{O}, \tag{5.1}$$

where $\boldsymbol{p}(t)$ is the vessel position and $\boldsymbol{o}_i(t)$ is obstacle i in the set of obstacles $\mathcal{O}$. For the local path planner, this entails that both vertices and edges in our tree built by RRT* must be examined for a collision. For convenience, we use d_{min} both for static and dynamical obstacles here. Note that d_{min} should account for the uncertainties

introduced by the measurement system as well as the corridor width introduced in Section 4.2.2.

We separate static and dynamic obstacles since the static ones can easily be incorporated into the OGM. To ensure that d_{min} is fullfilled for static obstacles, we introduce a new cost map layer, namely the *inflation layer*. Each occupied grid cell is inflated by d_{min}. This inflation increases the size of the obstacle space $\mathcal{X}_{obs}$ on the map. By checking that the vertices and the line segments[2] between them are not inside the inflated area, static obstacles are avoided.

Dynamic obstacles can be checked by ensuring that the vertices are not inside a circle with radius d_{min} and origin at $\boldsymbol{o}_i(t)$. The line segments between them can be easily checked by finding the projection (if it exists) of $\boldsymbol{o}_i(t)$ onto the line segment and ensure that the distance between the projection and $\boldsymbol{o}_i(t)$ is greater than d_{min}. Dynamic obstacles must, of course, be continuously updated. A proposal on how this can be done is given in Section 5.1.4. See Figure 5.2 for a visualization.

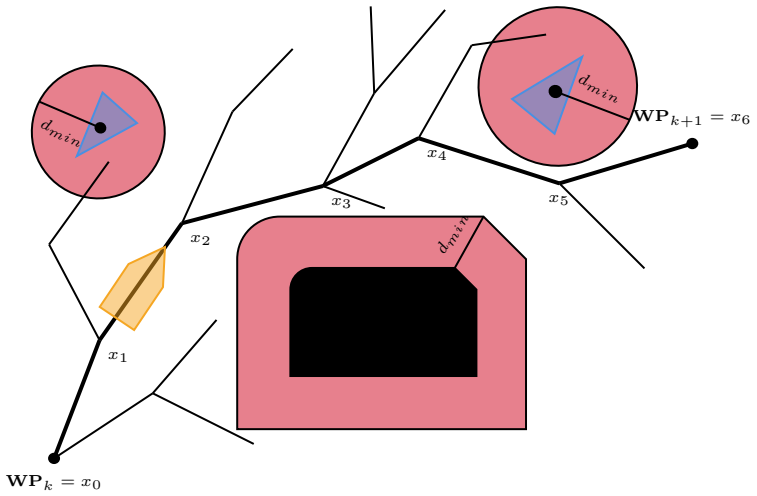


Figure 5.2: An expanded RRT tree between two global waypoints $\mathbf{WP}_k$ and $\mathbf{WP}_{k+1}$. The chosen path of local waypoints $x_0, \ldots, x_6$ is drawn with a thick marker. One static obstacle in black, two dynamic obstacles in blue, and their respective obstacle regions in red are shown.

[2]The line segment between two points in an OGMs can be checked using Bresenham's line algorithm (Joy, 1999).

Course Variation

To comply with the assumption from Section 3.5 on a maximum restriction on the angle between previous, current, and next waypoint we impose that:

$$|\psi_k - \psi_{k+1}| < \psi_{max}, \quad k \in \mathcal{K}^m. \tag{5.2}$$

Here ψ_k is the heading at waypoint k defined in Equation (4.6), ψ_{max} is the maximum allowed course variation, and $\mathcal{K}^m$ is the set of waypoint indices.

Curvature

It is in our interest to keep the curvature low to respect the dynamic constraints of the vessel described in Section 2.1.3. The path planner cannot control the curvature of the path itself. However, one can constraint the curvature of the circumference connecting the midpoints of the respective segments. This will cause that the waypoints are connected such that the path is kept straight with less course variation between them. Let l_k be the Euclidean distance between two consecutive waypoints, x_{k-1} and x_k, and ψ_k its heading. The constraint then takes the form (Martelli and Zaccone, 2018):

$$2\frac{\tan\left(|\psi_k - \psi_{k+1}|\right)}{\min(l_k, l_{k+1})} < \rho_{max}, \quad k \in \mathcal{K}^m \setminus m, \tag{5.3}$$

where ρ_{max} is the maximum curvature allowed of the circumference connecting the midpoints, and $\mathcal{K}^m$ is the set of waypoint indices.

5.1.2 Cost Function

A weakness of the standard RRT algorithm is that it does not take into account the path cost. This can lead to path solutions that are far from optimal. The rewiring procedure of RRT* performs an optimizing action and seeks to minimize a specified cost function as given in Problem 2.1. The cost function will serve as a heuristic that biases the growth of the tree towards those regions that result in low-cost solutions. Thus, the cost function controls the shape of the solution. It is essential to decide upon what constitutes a better path in terms of waypoints, as for the path generator discussed in Section 4.2. Since the path planner, more or less, decides the shape of the overall path, a reasonable path in terms of waypoints could be one that minimizes the path elongation and the number of maneuvering actions. Furthermore, it is desired to keep as much distance as possible from the obstacles.

Proposition 5.1. *A reasonable path in terms of waypoints for a low-speed vessel such as the ReVolt is one where the traveled distance/energy consumed and the number of maneuvering actions is minimized, and the distance to obstacles is maximized.*

The features given in Proposition 5.1 can be conflicting in many scenarios. If so, a trade-off solution should be chosen.

Minimum Distance

Let l_k be the Euclidean distance between two consecutive waypoints, x_{k-1} and x_k. The distance cost of waypoint number k is then:

$$g_l(x_k) = \sum_{k \in \mathcal{K}^m} l_k, \tag{5.4}$$

where $\mathcal{K}^m$ is the set of waypoint indices.

Maneuvering Actions

We want to penalize maneuvering actions. Let ρ_k be equal to the left side of the inequality sign in Equation (5.3):

$$\rho_k = 2\frac{\tan\Big(|\psi_k - \psi_{k+1}|\Big)}{\min(l_k, l_{k+1})}. \tag{5.5}$$

The maneuvering cost of waypoint number k is then, as proposed by Martelli and Zaccone (2018):

$$g_\rho(x_k) = \max_{k \in \mathcal{K}^m} \rho_k + \frac{1}{m} \sum_{k \in \mathcal{K}^m} \rho_k, \tag{5.6}$$

where $\mathcal{K}^m$ is the set of waypoint indices, and m is the number of waypoints. g_ρ will contribute to keeping the path straight with smooth turns. Also, it will distribute the waypoints at regular distances.

Obstacles

To maximize the distance to obstacles, we mimic the repulsive forces of obstacles used primarily by potential field algorithms discussed in Section 2.4. Let $d(x_k, \boldsymbol{o}_i)$ be the Euclidean distance between waypoint number k and obstacle i. The obstacle cost of waypoint number k is then:

$$g_o(x_k) = \frac{1}{2} \sum_{k \in \mathcal{K}^m} \frac{1}{\min\limits_{o_i \in \mathcal{O}} d(x_k, \boldsymbol{o}_i)^2}, \tag{5.7}$$

where $\mathcal{K}^m$ is the set of waypoint indices, and $\mathcal{O}$ is the set of obstacles. Note that the repulsive potential as defined here will influence the cost function at any distance $d(x_k, \boldsymbol{o}_i)$ from obstacle i. It is common to set a certain limit on how far away from

the object, the repulsive potential should affect the cost, but for simplicity, this is not treated here.

By combining the proposed cost functions, the total cost of waypoint number k can be stated as:

$$g_{tot}(x_k) = \delta \cdot g_l(x_k) + \epsilon \cdot g_\rho(x_k) + \zeta \cdot g_o(x_k), \tag{5.8}$$

where δ, ϵ, and ζ are tuning variables for the respective cost functions. Each cost function will produce trees with significantly different topologies and path shapes. See Simulation 1 in Section 8.4.1 for a visualization. Thus, the cost function should be tuned properly to obtain the desired behavior.

5.1.3 Informed RRT*

RRT* will asymptotically find the optimal path from the initial node to every other node in the configuration space. For our problem, this is both inefficient and inconsistent with the single-query problem treated in this thesis. Gammell et al. (2014) showed that for problems that want to minimize the path elongation, the subset of states that can improve a solution could be described by an ellipse for an $\mathbb{R}^2$ problem. For large configuration spaces, they showed that unless the subset given by the ellipse was sampled directly, the probability of improving the solution becomes arbitrarily small. The algorithm is named *informed* RRT* because it utilizes the knowledge obtained from the informed sampling procedure.

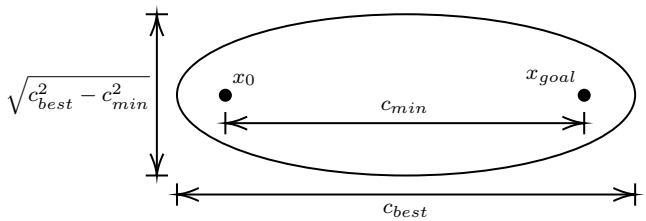


Figure 5.3: The heuristic sampling domain $\mathcal{X}_{ellip}$ for an $\mathbb{R}^2$ problem seeking to minimize path length, is an ellipse with the initial state x_0 and the goal state x_{goal} as focal points. c_{best} is the cost of the best solution found so far and c_{min} is the theoretically minimum cost between x_0 and x_{goal}. The shape of the ellipsoid will depend on x_0, x_{goal}, c_{best}, and c_{min}.

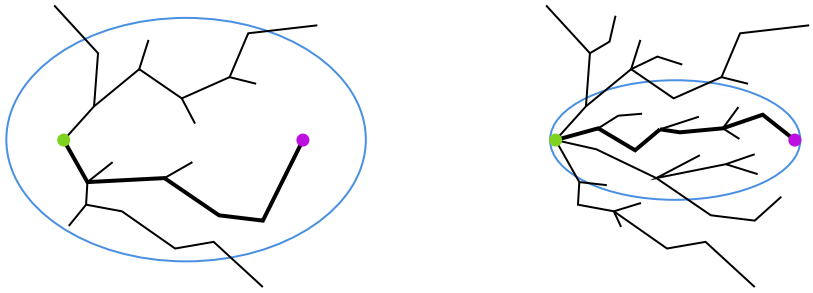
Informed RRT* is an extension to RRT* that introduces a clear improvement in terms of convergence rate and final solution quality. It retains the same probabilistic guarantees on completeness and optimality as RRT*. Informed RRT* works as RRT* until a first feasible solution is found. Then, for the solution to be improved at any iterations, Gammell et al. (2014) proved that states must be sampled from a heuristic

domain $\mathcal{X}_{ellip} \subset \mathcal{X}$:

$$\mathcal{X}_{ellip} = \{x \in \mathcal{X} \mid |x_0 - x| + |x_{goal} - x| \leq c_{best}\}, \tag{5.9}$$

which is the general equation for an n-dimensional hyperspheroid. Here we treat only $n = 2$. The transverse diameter of the ellipsoid is c_{best} and the conjugate one is $\sqrt{c_{best}^2 - c_{min}^2}$. See Figure 5.3.

The algorithm limits the search to the subproblem formed by the ellipsoid containing all possible better solutions. As time goes and better solutions are found, the heuristic domain will decrease, and the ellipsoid will shrink. See Figure 5.4. Uniform sampling from the ellipsoid can then be achieved by transforming uniformly distributed samples from a unit circle. For a detailed description refer to Gammell et al. (2014).



(a) **(b)**

Figure 5.4: Informed RRT* expanding its tree and converging in the absence of obstacles. The start and goal node are shown in green and purple, respectively. The current solution is shown with a thick marker, and the borders of $\mathcal{X}_{ellip}$ in blue. In (**a**) the initial solution is found. In (**b**) the solution is improved, and the area of the ellipse decreases. Asymptotically the ellipse will degenerate to a line between the start and goal.

5.1.4 Real-Time RRT*

Until now, our design of the local planner has only addressed a static environment. In order to react to dynamic obstacles, a real-time extension of RRT* must be addressed. Different real-time extensions have been proposed in the literature. For example, see ERRT and CL-RRT proposed by Bruce and Veloso (2002) and Kuwata et al. (2009). The expanded tree of these algorithms are only used as a look-ahead in the environment, and thus only covers small portions of the environment. Naderi et al. (2015) presented the Real-Time RRT* (RT-RRT*) algorithm that retains the whole tree in the environment and rewires the nodes online to be able to handle dynamic

obstacles. Our real-time extension of RRT* is heavily based on the work of Naderi et al. (2015).

The algorithm is given in Algorithm 3. RT-RRT* interleaves two main tasks: expansion-and-rewiring of the tree and path planning. For RT-RRT* to manage real-time execution, the algorithm has a limited amount of time for both tasks. The algorithm is initialized with a root node x_0, equal to the start location. In each loop iteration, we expand and rewire the tree as long as possible. Then, we plan a path towards the goal. If x_{goal} is found, we follow the path leading to it, just like RRT*. If not, we plan a K-step path $(x_0, x_1, \ldots, x_k)$ from the current tree root, according to our cost function. Now in Lines 8 and 9 of Algorithm 3, we check if the vessel is sufficiently close to x_0. If so, we update x_0 to the next node, x_1, in our planned path. Hence, we enable the ASV to move on to the planned path towards the goal in a stepwise manner. By updating and keeping the ASV near the root, we can retain the already expanded tree.

Algorithm 3: RT-RRT* (Naderi et al., 2015).

```
1  input: x_0, x_goal, X
2  Initialize T = (V ← {x_0}, E = ∅), Q_r, Q_s;
3  for i = 1, ..., n do
4      Update x_0, x_goal, X_free, X_obst;
5      while time is left for Expansion and Rewiring do
6          Expand and Rewire T using Algorithm 4;
7      Plan (x_0, x_1, ..., x_k) using Algorithm 5 ;
8      if p(t) is close to x_0 then
9          x_0 ← x_1;
10     Move the vessel towards x_0 for a limited time;
```

Tree Expansion and Rewiring

The tree expansion-and-rewiring algorithm is given in Algorithm 4. The method will be described briefly. For a detailed description of all primitive procedures used, see Naderi et al. (2015).

The sampling of random nodes is done in the same way as for Informed RRT*, but we only focus part of the sampling inside the ellipse in case of changes in the

environment. The sampling procedure in Line 2 of Algorithm 4 is performed as:

$$x_{rand} = \begin{cases} x_{goal}, & \text{if } P_r > 1 - \alpha, \\ \mathcal{U}(\mathcal{X}_{free}), & \text{if } \begin{cases} \text{if } P_r > \dfrac{1-\alpha}{\beta}, \\ \text{or goal is not found,} \end{cases} \\ \text{Sample ellipse}, & \text{otherwise,} \end{cases} \tag{5.10}$$

where $P_r \in [0, 1]$ is a random drawn number, $\alpha \in [0, 1]$ is a tuning variable for sampling the goal state, and $\beta \in \mathbb{R}_{>0}$ is for dividing the sampling procedure between uniform sampling and inside the ellipse. Also, the rotation of the ellipse is updated every iteration the root x_0 is changed.

To control the density of the tree, we check in Line 7 of Algorithm 4 that the maximum number of neighbors around a node, k_{max}, and the minimum Euclidean distance, r_{min}, between nodes in the tree is not violated before adding the sampled node to the tree. The new sampled node, x_{new}, is then used for rewiring random parts of the tree around itself or its closest node $x_{nearest}$.

Gradually, as more iterations are done, the tree becomes too large to handle real-time path planning. To find nearby nodes, we use grid-based spatial indexing of nodes. We divide the configuration space into square grid cells. The set of nodes $\mathcal{X}_{SI} \subset \mathcal{T}$ for a node x_{new} is then found by only considering the neighboring grid cells and the grid that x_{new} is contained inside. A grid cell g_i is considered a neighbor to g_j if it is the closest non-empty grid (at least one node inside). See Figure 5.5. Note that the size of each grid cell strongly influences the processing time and the size of $\mathcal{X}_{SI}$. Also, the size of each cell should be big enough to contain the dynamic obstacle regions.

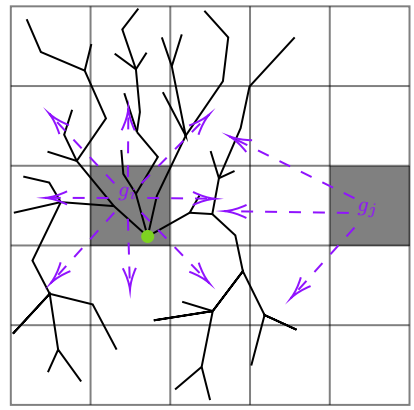


Figure 5.5: Adjacent grid cells for g_i and g_j.

Same as for RRT*, rewiring is done when a new sampled node is added to the tree, but in addition, rewiring is done when the tree root x_0 or dynamic obstacles are changing. Thus, rewiring is done in two different ways:

1. Rewiring starting from a random node in the tree (Line 12 in Algorithm 4).
2. Rewiring starting from the tree root, x_0 (Line 13 in Algorithm 4).

This is done by using two queues, $\mathcal{Q}_s$ and $\mathcal{Q}_r$. Nodes added to the queues are nodes that should be rewired. Except for the start point, the two procedures do the same:

Algorithm 4: Tree Expansion-and-Rewiring (Naderi et al., 2015).

input: $\mathcal{T}$, $\mathcal{Q}_r$, $\mathcal{Q}_s$, k_{max}, r_{min}
$x_{rand} \leftarrow \texttt{Sample}_i$;
$x_{nearest} \leftarrow \texttt{Nearest}(\mathcal{T}, x_{rand})$;
$x_{new} \leftarrow \texttt{Steer}(x_{nearest}, x_{rand})$;
if $\texttt{ObstacleFree}(x_{nearest}, x_{new})$ **then**
    $X_{near} \leftarrow \texttt{Near}(\mathcal{T}, x_{new}, \mathcal{X}_{SI})$;
    if cardinality$(X_{near}) < k_{max}$ or $|x_{nearest} - x_{new}| > r_s$ **then**
        $\texttt{AddNodeToTree}(\mathcal{T}, x_{new}, x_{nearest}, X_{near})$;
        Push x_{new} to $\mathcal{Q}_r$;
    else
        Push $x_{nearest}$ to $\mathcal{Q}_s$;
    $\texttt{RewireRandomNode}(\mathcal{Q}_r, \mathcal{T})$
$\texttt{RewireFromRoot}(\mathcal{Q}_s, \mathcal{T})$

We start by rewiring a node. Then, we continue to push the neighbors of the node to the respective queue as well as popping new nodes to be rewired from it. The procedures then iteratively rewire larger portions of the tree until a time condition is met. See Naderi et al. (2015) for algorithm details.

Blocking Nodes by Dynamic Obstacles

By the assumption that the position of all dynamic obstacles is known, we can block branches of the tree contained inside their obstacle region. A node is blocked by setting its cost-to-reach, c_{tot}, to infinity. As a result, branches going out from the blocked node will get infinite cost-to-reach. Gradually, as rewiring is performed, the children of the blocked node will be connected to other nodes with lower cost-to-reach values, thus creating new paths around the dynamic obstacle. With this strategy, dynamic obstacles are avoided. See Figure 5.6.

Planning

In Line 7 of Algorithm 3, RT-RRT* plans a K-step path from x_0. Two scenarios must be handled; when the tree has reached x_{goal}, and when it has not. In the first case, we follow the path leading to the x_{goal}, just like RRT*, except when the path is rewired. Then, we update the path accordingly. In the second case, we plan a subpath that ideally brings us closer to x_{goal} in the growing tree. We use a cost function to guide us towards the goal:

$$f(x_k) = g_{tot}(x_k) + h(x_k). \tag{5.11}$$

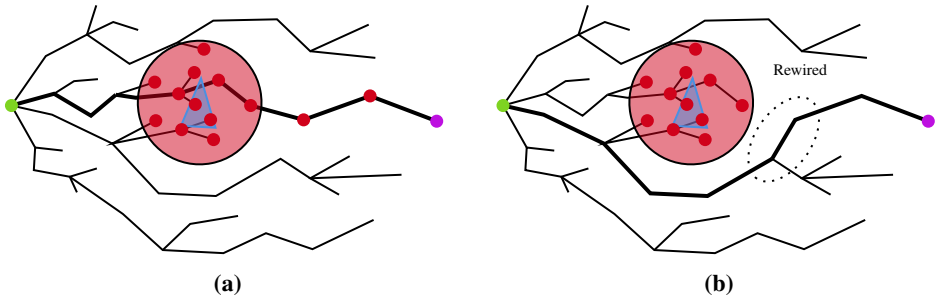


Figure 5.6: RT-RRT* blocking nodes and outgoing branches from an obstacle region. The start and goal nodes are shown in green and purple, respectively. The obstacle region is shown by a transparent red region, blocked nodes with red marks, and the current solution with a thick marker. In (**a**) the initial solution is blocked, and the cost-to-reach is set to infinity. In (**b**) the tree is rewired to find a new solution going around the obstacle region.

Here, $g_{tot}(x_k)$ is defined in Equation (5.8) and $h(x_k)$ is an admissible heuristic[3] given as the Euclidean distance to x_{goal}:

$$h(x_k) = |x_k - x_{goal}|. \tag{5.12}$$

By using the cost function $f(x_k)$, one risk being caught in local minima. This is solved by planning a K-step path at each iteration and block already "seen" nodes by setting their heuristic to infinity. Thus the planner can visit other branches. When the planner reaches a node that is a leaf node or blocked, the algorithm returns the planned path and block the node. Then, the "best-already-found" path is updated if the planned path leads us to a better location (closer to x_{goal}). However, the vessel only follows the path if it leads to a better location than the current placement of the vessel. All nodes have the chance to be revisited when another unblocked node is added as its children by rewiring or adding a new node. Then all ancestors are unblocked. Note that if x_{goal} is reached, and the path is blocked by an obstacle, the path is followed up to the obstacle until rewiring finds another one, or the path gets cleared. The algorithm is given in Algorithm 5.

[3] An *admissible heuristic* is one that never overestimates the cost to reach the goal (Russell and Norvig, 2009).

Algorithm 5: Plan a K-step Path (Naderi et al., 2015).

input: $\mathcal{T}$, x_{goal}
if Tree has reached x_{goal} **then**
    Update `path` from x_0 to x_{goal} if `path` is rewired;
    $(x_0, \ldots, x_k) \leftarrow (x_0, \ldots, x_{goal})$;
else
    for $x_i \in (x_0, \ldots, x_k)$ **do**
        x_i = child of x_{i-1} with minimum $f(x_c) = g(x_c) + h(x_c)$;
        if x_i is leaf node or its children are blocked **then**
            $(x'_0, \ldots, x'_k) \leftarrow (x_0, \ldots, x_i)$;
            `block`(x_i);
            break loop;
        Update best `path` with $(x'_0, \ldots, x'_k)$ if necessary;
        $(x'_0, \ldots, x'_k) \leftarrow$choose to stay in x_0 or follow best `path`;
return $(x_0, \ldots, x_k)$;

5.2 Global Planner - A* Algorithm on a Voronoi Roadmap

The global planner should solve the superior problem of navigating the vessel from start to goal using the information known before the operation. More specifically, it should determine the initial, intermediate, and destination waypoint with corresponding reference speeds in the regions. For efficiency reasons, the global planner will be of low resolution. This means that we do not consider all grids in a configuration space, but partition it in order to be suited for global planning. In this thesis, we will partition the configuration space and construct a roadmap using Voronoi roadmaps, represented in Section 2.5.2. The metric function will be the Euclidean distance given in Equation (2.43). Static obstacles and map borders are used as generator points.

By using the Voronoi roadmap, one can efficiently use an optimal combinatorial algorithm. The design of the global planner is similar to the ones represented by Candeloro et al. (2013) and Lekkas (2014), and can be summarized by the following steps:

1. Create a raw obstacle-free roadmap using Voronoi partitioning of the configuration space.
2. Waypoints and paths going out of the configuration space border are removed.
3. Include the start and goal location of the vessel in the roadmap.
4. The A* algorithm is used to find the optimal path σ^* in the roadmap while

directly verifying if clearance constraints are fulfilled.

5. Given a feasible path, its collinear and almost-collinear waypoints are removed.

6. Finally, the pruning of unnecessary waypoints is done.

The difference between the algorithm represented by Candeloro et al. (2013) is that we verify if the clearance constraints are fulfilled directly in step 4, which ensures that A* only needs to search for a feasible path once. This can be done since we know for a fact that our path generator designed in Chapter 4 will generate the path inside a certain corridor width (which should be smaller or equal to our clearance constraint). If the clearance constraint is not fulfilled, we set the heuristic cost of the following node to infinity, ensuring that the path is never chosen.

The creation of a Voronoi roadmap is explained in Section 2.5.2. In the following subsections step 4, 5, and 6 will be described.

5.2.1 The A* Algorithm

For our global path planner, we will use the well-known A* algorithm, first introduced by Hart et al. (1968). A* is a graph-traversal algorithm that has been widely used in the robotics and AI community because of its optimality, optimal efficiency, and completeness (see Russell and Norvig (2009)).

The algorithm is an informed search that leverages the information of the start and goal location. It evaluates waypoints in the roadmap using a cost function $F(\mathbf{WP}_k)$, combining the cost to reach the waypoint, $G(\mathbf{WP}_k)$, and the heuristic cost to reach the goal from the waypoint $H(\mathbf{WP}_k)$:

$$F(\mathbf{WP}_k) = G(\mathbf{WP}_k) + H(\mathbf{WP}_k). \tag{5.13}$$

$F(\mathbf{WP}_k)$ can then be seen as the estimated cost of the cheapest solution through node $\mathbf{WP}_k$. To find the optimal solution, we explore in a greedy manner, the neighbors of the current node with the lowest estimated cost. This strategy is proven to be both complete and optimal, given that the heuristic $H(\mathbf{WP}_k)$ is admissible (Russell and Norvig, 2009). For our global planner, the Euclidean distance will be used to calculate the cost. Assuming that the distance between the waypoints given by the Voronoi roadmap is big enough not to violate the dynamic constraints of the vessel, it is reasonable to not bother about maneuvering actions. The cost to reach node $\mathbf{WP}_k$ is thus defined to be equal to the path length cost of the local planner in Equation (5.4):

$$G(\mathbf{WP}_k) = \sum_{k \in \mathcal{K}^m} l_k, \tag{5.14}$$

where l_k is the Euclidean distance between two consecutive waypoints $\mathbf{WP}_{k-1}$ and $\mathbf{WP}_k$, and $\mathcal{K}^m$ is the set of waypoint indices. The admissible heuristic cost is further given as:

$$H(\mathbf{WP}_k) = |\mathbf{WP}_k - \mathbf{WP}_{goal}|. \tag{5.15}$$

A detailed description with pseudo-code of the algorithm can be found in Russell and Norvig (2009).

5.2.2 Clearance Constraints

In step 6, one needs to verify that the piecewise linear path between the global waypoints respects a certain clearance constraint d_{min}. By assuming that the static obstacles are greater in size than d_{min}, this can be done by verifying if the grid cells in the OGM a distance d_{min} on both sides between two consecutive waypoints are part of the free space. The line segments to be verified bear similarity to the corridor walls introduced in Section 4.2.2. Four points must be calculated, two endpoints for each line segment (or wall). The grid cells between the points can then be found using Bresenham's line algorithm (Joy, 1999). See Algorithm 6.

Algorithm 6: Check Clearance Constraint.

```
input: WP_k, WP_{k+2}, d_min
p_1, ..., p_4 ← Get endpoints of line segments to be verified;
Cells = Bresenham(p_1, ..., p_4);
for cell ∈ Cells do
    if cell ∈ X_obs then
        // Clearance constraint violated
        return False;
return True;
```

5.2.3 Pruning of Waypoints

Both step 5 and 7 in our global planner, similarly prune waypoints to achieve a more practical path. The Voronoi diagram produces a high number of waypoints, proportional to the number of generator points. Including all the waypoints found from the A* search can result in unpractical paths.

First, as stated in Proposition 5.1, it is desired to limit the number of maneuvering actions. By removing collinear and almost-collinear waypoints in step 5, one reduces heading changes as well as shorten the path length.

We treat three consecutive waypoints $\mathbf{WP}_1$, $\mathbf{WP}_2$, and $\mathbf{WP}_3$, and calculate the heading change between the former, and the latter once as:

$$|d\psi_{1-2} - d\psi_{2-3}| < \psi_{thres}, \tag{5.16}$$

where $d\psi_{1-2}$ and $d\psi_{2-3}$ is the heading between $\mathbf{WP}_1$ and $\mathbf{WP}_2$, and $\mathbf{WP}_2$ and $\mathbf{WP}_3$, respectively. ψ_{thres} is the threshold value for pruning waypoints. If the heading change is lower than ψ_{thres}, we remove the waypoint. In step 7, we prune excessive waypoints to achieve a more practical path in terms of fewer waypoints. There is no reason to assign a path A-B-C if the path A-C is feasible and respects the clearance constraint, verified by Algorithm 6.

Both step 5 and 7 in our global planner can be done by iterating through the list of waypoints found by the A* and verify if the angle/clearance constraint is met in Line 7 of Algorithm 7 (using Algorithm 6 or Equation (5.16)). See Figure 5.7 for a visualization.

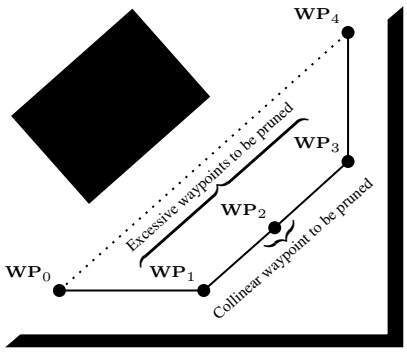


Figure 5.7: Five waypoints and the connecting straight-line path between them drawn with a thick marker are shown. Black areas are obstacles. $\mathbf{WP}_1$, $\mathbf{WP}_2$, and $\mathbf{WP}_3$ are collinear, thus $\mathbf{WP}_2$ can be pruned. Further, by assuming that the straight-line path between $\mathbf{WP}_0$ and $\mathbf{WP}_4$ satisfy the clearance constraint, all intermediate waypoints can be pruned.

Algorithm 7: Pruning of Waypoints.

1 **input:** $\sigma^* = \{\mathbf{WP}_0, \ldots, \mathbf{WP}_{goal}\}$, d_{min} or ψ_{thres}
2 $i = 0$;
3 **while** $i <$ size$(\sigma*) - 2$ **do**
4 $\quad \mathbf{WP}_1 = \sigma^*[i]$;
5 $\quad \mathbf{WP}_2 = \sigma^*[i+1]$;
6 $\quad \mathbf{WP}_3 = \sigma^*[i+2]$;
7 $\quad$ **if not** `Angle/Clearance Constraint` **then**
8 $\quad\quad$ `Remove` $\mathbf{WP}_2$ from σ^*;
9 $\quad$ **else**
10 $\quad\quad$ $i = i + 1$;
11 **return** σ^*;

Chapter 6

Control System

This chapter addresses the problem of designing a high-level control law together with a thrust allocation algorithm that satisfies the control objective given in Section 3.1. The fully actuated low-speed control design model in Equation (2.1) will be used to designing a maneuvering controller. We assume that the system states $\{\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{b}\}$ are provided by an observer from the measurement system. To design the maneuvering controller, we will use nonlinear stability theory. See Khalil (2002) for a reference.

6.1 Maneuvering Control Design

To design the maneuvering controller, we will use the nonlinear *adaptive backstepping* technique. The method is described in detail by Krstic et al. (1995) and includes methods for tuning functions, parameter adaptation, and modular designs for both full-state feedback and output feedback (observer backstepping). In short, backstepping is a recursive technique breaking the design problem of the full system down to a sequence of sub-problems on lower-order systems, and by recursively use some states as *virtual control* inputs to obtain the intermediate control laws using the appropriate Control Lyapunov Function (CLF) (Zhang and Qian, 2017). The design is based on Skjetne (2019), and is done in two steps. After the first step, we define the *dynamic update law* for the parametric value $\dot{s}(t)$ to fulfill the control objective in Equation (3.1), and to ensure that the update law only acts in the output space. Bear in mind the system matrices and their properties given in Section 2.1.

6.1.1 Adaptive Backstepping - Step 1

We define the error state variables:

$$\boldsymbol{z}_1 \triangleq \boldsymbol{R}(\psi)^\top(\boldsymbol{\eta} - \boldsymbol{\eta}_d(s)), \tag{6.1}$$

$$\boldsymbol{z}_2 \triangleq \boldsymbol{\nu} - \boldsymbol{\alpha}_1, \tag{6.2}$$

$$\omega \triangleq \dot{s} - v_s(t, s), \tag{6.3}$$

where $\boldsymbol{\alpha}_1$ is the virtual control to be specified later. The total derivative of $\boldsymbol{z}_1$ is:

$$\dot{\boldsymbol{z}}_1 = \dot{\boldsymbol{R}}(\psi)^\top(\boldsymbol{\eta} - \boldsymbol{\eta}_d(s)) + \boldsymbol{R}(\psi)^\top(\dot{\boldsymbol{\eta}} - \boldsymbol{\eta}_d^s(s)\dot{s}) \tag{6.4}$$

$$= -\boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top(\boldsymbol{\eta} - \boldsymbol{\eta}_d(s)) + \boldsymbol{\nu} - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)\dot{s} \tag{6.5}$$

$$= -\boldsymbol{S}(r)\boldsymbol{z}_1 + \boldsymbol{z}_2 + \boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s)). \tag{6.6}$$

The first CLF is defined as:

$$V_1 \triangleq \frac{1}{2}\boldsymbol{z}_1^\top\boldsymbol{z}_1, \tag{6.7}$$

and its total derivative is:

$$\dot{V}_1 = \frac{1}{2}\dot{\boldsymbol{z}}_1^\top\boldsymbol{z}_1 + \frac{1}{2}\boldsymbol{z}_1^\top\dot{\boldsymbol{z}}_1 \tag{6.8}$$

$$= \frac{1}{2}\Big(- \boldsymbol{S}(r)\boldsymbol{z}_1 + \boldsymbol{z}_2 + \boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big)^\top\boldsymbol{z}_1 \tag{6.9}$$

$$+ \frac{1}{2}\boldsymbol{z}_1^\top\Big(- \boldsymbol{S}(r)\boldsymbol{z}_1 + \boldsymbol{z}_2 + \boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big)$$

$$= \frac{1}{2}\Big(- \boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1 + \boldsymbol{z}_1^\top\boldsymbol{z}_2 + \boldsymbol{z}_1^\top\boldsymbol{\alpha}_1 \tag{6.10}$$

$$- \boldsymbol{z}_1^\top\Big[\boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big] - \boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1 + \boldsymbol{z}_1^\top\boldsymbol{z}_2 + \boldsymbol{z}_1^\top\boldsymbol{\alpha}_1$$

$$- \boldsymbol{z}_1^\top\Big[\boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big]\Big)$$

$$= \underbrace{-\boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1}_{=0^1} + \boldsymbol{z}_1^\top\boldsymbol{z}_2 + \boldsymbol{z}_1^\top\Big[\boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big], \tag{6.11}$$

and furthermore, its derivative with respect to s is:

$$V_1^s = \frac{1}{2}\boldsymbol{z}_1^{s\top}\boldsymbol{z}_1 + \frac{1}{2}\boldsymbol{z}_1^\top\boldsymbol{z}_1^s \tag{6.12}$$

$$= \frac{1}{2}\Big(- \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)\Big)^\top\boldsymbol{z}_1 + \frac{1}{2}\boldsymbol{z}_1^\top\Big(- \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)\Big) \tag{6.13}$$

$$= -\boldsymbol{z}_1^\top\boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s). \tag{6.14}$$

Note now the use of Young's inequality [2], such that:

$$\dot{V}_1 = \boldsymbol{z}_1^\top\boldsymbol{z}_2 + \boldsymbol{z}_1^\top\Big[\boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big] \tag{6.15}$$

$$\leq \frac{1}{4\gamma}\boldsymbol{z}_2^\top\boldsymbol{z}_2 + \boldsymbol{z}_1^\top\Big[\gamma\boldsymbol{z}_1 + \boldsymbol{\alpha}_1 - \boldsymbol{R}(\psi)^\top\boldsymbol{\eta}_d^s(s)(\omega + v_s(t,s))\Big]. \tag{6.16}$$

[1] $\boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1 = \boldsymbol{z}_1^\top(-\boldsymbol{S}(r))\boldsymbol{z}_1 \implies 2\boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1 = 0 \implies \boldsymbol{z}_1^\top\boldsymbol{S}(r)\boldsymbol{z}_1 = 0$

[2] Young's inequality: $\boldsymbol{a}^\top\boldsymbol{b} \leq \frac{1}{4\gamma}\boldsymbol{b}^\top\boldsymbol{b} + \gamma\boldsymbol{a}^\top\boldsymbol{a}$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n,\ \gamma > 0$.

We choose our first virtual control $\boldsymbol{\alpha}_1$ and tuning function ρ_1 as:

$$\boldsymbol{\alpha}_1 = -\boldsymbol{K}_1 \boldsymbol{z}_1 + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) - \gamma \boldsymbol{z}_1, \quad \boldsymbol{K}_1 = \boldsymbol{K}_1^\top > 0,\ \gamma > 0, \tag{6.17}$$

$$\rho_1 = -\boldsymbol{z}_1^\top \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) = V_1^s. \tag{6.18}$$

Plugging $\boldsymbol{\alpha}_1$ and ρ_1 into Equation (6.16) gives:

$$\dot{V}_1 \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 + \rho_1 \omega + \frac{1}{4\gamma} \boldsymbol{z}_2^\top \boldsymbol{z}_2, \tag{6.19}$$

where we postpone dealing with the coupling term involving $\boldsymbol{z}_2$ until the next step.

6.1.2 Dynamic Update Law Acting in Output Space:

The dynamic update law is constructed to bridge the path following objective with the speed assignment. We examine two candidates so that the hybrid path signal being sent from the guidance system can be controlled. Remember that:

$$\omega \triangleq \dot{s} - v_s(t,s), \tag{6.20}$$

$$\dot{V}_1 \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 + \rho_1 \omega + \frac{1}{4\gamma} \boldsymbol{z}_2^\top \boldsymbol{z}_2, \tag{6.21}$$

$$\rho_1 = -\boldsymbol{z}_1^\top \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) = V_1^s. \tag{6.22}$$

- **Tracking Update Law:** Choosing:

$$\omega = 0 \implies \dot{s} = v_s(t,s), \tag{6.23}$$

  satisfies the dynamic task in Equation (3.1). Furthermore, leaving the coupling term for the next step and inserting ω into Equation (6.21) we obtain:

$$\dot{V}_1 \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1. \tag{6.24}$$

  Hence the tracking update law renders $\dot{V}_1$ negative definite and thereby the equilibrium $\boldsymbol{z}_1 = 0$ uniformly globally exponentially stable (UGES)[1]. We call it a "tracking update law" since $\dot{s}$ just becomes a time signal following the speed assignment $v_s(t,s)$.

- **Unit-Tangent Gradient Update Law:** Choosing:

$$\omega = -\mu \rho_1 = -\mu V_1^s, \quad \mu \geq 0, \tag{6.25}$$

$$\implies \dot{s} = v_s(t,s) - \mu V_1^s = v_s(t,s) + \mu \boldsymbol{z}_1^\top \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) \tag{6.26}$$

$$= v_s(t,s) + \mu \boldsymbol{\eta}_d^s(s)^\top \boldsymbol{R}(\psi) \boldsymbol{z}_1. \tag{6.27}$$

[1] See Lemma 4.5 in Khalil (2002).

This is known as the gradient update law because of V_1^s. μ is a tuning variable. This update law is a smooth dynamic optimization algorithm that selects the point $\boldsymbol{\eta}_d(s)$ that minimizes the weighted distance between $\boldsymbol{\eta}$ and $\boldsymbol{\eta}_d(s)$. See Chapter 3.4 in Skjetne (2005) for details. Note that $\boldsymbol{\eta}_d^s(s)$ may for a given s have varying length, depending on the parameterization. Since it appears in Equation (6.27), the gradient update law will have a varying gain along the path. To avoid this, we divide by $|\boldsymbol{\eta}_d(s)|$ and use the unit tangent vector, such that:

$$\dot{s} = v_s(t,s) + \mu \frac{\boldsymbol{\eta}_d^s(s)^\top}{\left|\boldsymbol{\eta}_d^s(s)\right|} \boldsymbol{R}(\psi)\boldsymbol{z}_1. \tag{6.28}$$

Same as for the tracking update law, we leave the coupling term for the next step, and insert ω into Equation (6.21) to obtain:

$$\dot{V}_1 \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 - \mu \frac{V_1^{s\,2}}{\left|\boldsymbol{\eta}_d^s(s)\right|}. \tag{6.29}$$

Hence the unit tangent gradient update law renders $\dot{V}_1$ negative definite and thereby the equilibrium $\boldsymbol{z}_1 = 0$ is UGES.

For each of these choices we get that the term $\rho_1 \omega \leq 0$ in Equation (6.21), thus we can leave it out for the next step. Concluding the first step yields:

$$\text{Step 1:} \begin{cases} \tilde{\boldsymbol{K}}_1 & = \boldsymbol{K}_1 + \gamma I > 0 \\ \dot{\boldsymbol{z}}_1 & = -\Big(\tilde{\boldsymbol{K}}_1 + \boldsymbol{S}(r)\Big)\boldsymbol{z}_1 + \boldsymbol{z}_2 - \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s)\omega \\ \boldsymbol{\alpha}_1 & = -\tilde{\boldsymbol{K}}_1 \boldsymbol{z}_1 + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) \\ \dot{V}_1 & \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 + \dfrac{1}{4\gamma}\boldsymbol{z}_2^\top \boldsymbol{z}_2 \\ \dot{s} & = v_s(t,s) + \omega \end{cases} \tag{6.30}$$

6.1.3 Adaptive Backstepping - Step 2

The second CLF is defined as:

$$V_2 \triangleq V_1 + \frac{1}{2}\boldsymbol{z}_2^\top \boldsymbol{M} \boldsymbol{z}_2, \tag{6.31}$$

and its total derivative is:

$$\dot{V}_2 = \dot{V}_1 + \boldsymbol{z}_2^\top \boldsymbol{M} \dot{\boldsymbol{z}}_2 \tag{6.32}$$

$$= -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 + \frac{1}{4\gamma}\boldsymbol{z}_2^\top \boldsymbol{z}_2 + \boldsymbol{z}_2^\top \Big(\boldsymbol{M}\dot{\boldsymbol{\nu}} - \boldsymbol{M}\dot{\boldsymbol{\alpha}}_1\Big). \tag{6.33}$$

Plugging in our 3 DOF model in Equation (2.1) for $\boldsymbol{M}\dot{\boldsymbol{\nu}}$ we obtain:

$$\begin{aligned}\dot{V}_2 &= -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 + \frac{1}{4\gamma}\boldsymbol{z}_2^\top \boldsymbol{z}_2 \\ &\quad + \boldsymbol{z}_2^\top \Big(-\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{\tau} + \boldsymbol{R}(\psi)^\top \boldsymbol{b} - \boldsymbol{M}\dot{\boldsymbol{\alpha}}_1 \Big).\end{aligned} \tag{6.34}$$

We now choose $\boldsymbol{\tau}$ to stabilize our second CLF:

$$\boldsymbol{\tau} = -\boldsymbol{K}_2 \boldsymbol{z}_2 + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{R}(\psi)^\top \boldsymbol{b} + \boldsymbol{M}\dot{\boldsymbol{\alpha}}_1, \quad \boldsymbol{K}_2 = \boldsymbol{K}_2^\top > 0. \tag{6.35}$$

Plugging Equation (6.35) into Equation (6.34) we obtain:

$$\dot{V}_2 \leq -\boldsymbol{z}_1^\top \boldsymbol{K}_1 \boldsymbol{z}_1 - \boldsymbol{z}_2^\top \Big(\boldsymbol{K}_2 - \frac{1}{4\gamma} \Big) \boldsymbol{z}_2 \leq 0. \tag{6.36}$$

Thus $\boldsymbol{\tau}$ renders $\dot{V}_2$ negative definite and thereby the equilibrium point $(\boldsymbol{z}_1, \boldsymbol{z}_2) = (0, 0)$ UGES. Note that we need to find an expression for $\dot{\boldsymbol{\alpha}}_1$, which directly appears in our controller. We get:

$$\begin{aligned}\dot{\boldsymbol{\alpha}}_1 &= -\tilde{\boldsymbol{K}}_1 \dot{\boldsymbol{z}}_1 + \dot{\boldsymbol{R}}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \dot{\boldsymbol{\eta}}_d^s(s) v_s(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) \dot{v}_s(t,s)\end{aligned} \tag{6.37}$$

$$\begin{aligned}&= -\tilde{\boldsymbol{K}}_1 \dot{\boldsymbol{z}}_1 - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^{s^2}(s) \dot{s} v_s(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) \Big(v_s^t(t,s) + v_s^s(t,s)\dot{s} \Big).\end{aligned} \tag{6.38}$$

By plugging in for $\dot{\boldsymbol{z}}_1$ the expression can be further simplified:

$$\begin{aligned}\dot{\boldsymbol{\alpha}}_1 &= -\tilde{\boldsymbol{K}}_1 \Big(-\Big(\tilde{\boldsymbol{K}}_1 + \boldsymbol{S}(r) \Big) \boldsymbol{z}_1 + \boldsymbol{z}_2 - \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) \omega \Big) \\ &\quad - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^t(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \Big(\boldsymbol{\eta}_d^{s^2}(s) v_s(t,s) + v_s^s(t,s) \Big) \dot{s}\end{aligned} \tag{6.39}$$

$$\begin{aligned}&= -\tilde{\boldsymbol{K}}_1 \Big(-\Big(\tilde{\boldsymbol{K}}_1 + \boldsymbol{S}(r) \Big) \boldsymbol{z}_1 + \boldsymbol{\nu} + \tilde{\boldsymbol{K}}_1 \boldsymbol{z}_1 - \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(\dot{s} - \omega) \\ &\quad - \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) \omega \Big) - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^t(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \Big[\boldsymbol{\eta}_d^{s^2}(s) v_s(t,s) + v_s^s(t,s) \Big] \dot{s}\end{aligned} \tag{6.40}$$

$$\begin{aligned}&= \tilde{\boldsymbol{K}}_1 \boldsymbol{S}(r) \boldsymbol{z}_1 - \tilde{\boldsymbol{K}}_1 \boldsymbol{\nu} - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^t(t,s) + \Big[\tilde{\boldsymbol{K}}_1 \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^{s^2}(s) v_s(t,s) \\ &\quad + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^s(t,s) \Big] \dot{s}.\end{aligned} \tag{6.41}$$

The terms inside the square brackets in Equation (6.41) constitutes $\boldsymbol{\alpha}_1^s$. If we now define:

$$\boldsymbol{\sigma}_1 \triangleq \tilde{\boldsymbol{K}}_1 \boldsymbol{S}(r)\boldsymbol{z}_1 - \tilde{\boldsymbol{K}}_1 \boldsymbol{\nu} - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^t(t,s), \quad (6.42)$$

we can write:

$$\dot{\boldsymbol{\alpha}}_1 = \boldsymbol{\sigma}_1 + \boldsymbol{\alpha}_1^s \dot{s}. \quad (6.43)$$

6.1.4 Maneuvering Control Law

In summary, we need these signals for implementation:

$$\begin{aligned}
\boldsymbol{z}_1 &= \boldsymbol{R}(\psi)^\top (\boldsymbol{\eta} - \boldsymbol{\eta}_d(s)) \\
\boldsymbol{z}_2 &= \boldsymbol{\nu} - \boldsymbol{\alpha}_1 \\
\dot{s} &= v_s(t,s) + \omega \\
\boldsymbol{\alpha}_1 &= -\tilde{\boldsymbol{K}}_1 \boldsymbol{z}_1 + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) \\
\boldsymbol{\alpha}_1^s &= \tilde{\boldsymbol{K}}_1 \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^{s^2}(s) v_s(t,s) + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^s(t,s) \\
\boldsymbol{\sigma}_1 &= \tilde{\boldsymbol{K}}_1 \boldsymbol{S}(r)\boldsymbol{z}_1 - \tilde{\boldsymbol{K}}_1 \boldsymbol{\nu} - \boldsymbol{S}(r)\boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s(t,s) \\
&\quad + \boldsymbol{R}(\psi)^\top \boldsymbol{\eta}_d^s(s) v_s^t(t,s) \\
\boldsymbol{\tau} &= -\boldsymbol{K}_2 \boldsymbol{z}_2 + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{R}(\psi)^\top \boldsymbol{b} + \boldsymbol{M}\boldsymbol{\sigma}_1 + \boldsymbol{M}\boldsymbol{\alpha}_1^s \dot{s}
\end{aligned} \quad (6.44)$$

with the associated tuning variables:

$$\begin{aligned}
&\gamma > 0 \\
&\boldsymbol{K}_1 = \boldsymbol{K}_1^\top > 0 \\
&\tilde{\boldsymbol{K}}_1 = \tilde{\boldsymbol{K}}_1^\top = \boldsymbol{K}_1 + \gamma I > 0 \\
&\boldsymbol{K}_2 = \boldsymbol{K}_2^\top > 0
\end{aligned} \quad (6.45)$$

Note that the bias in the control law is assumed to be provided by a DP observer from the measurement system. If this is not the case, the bias can be compensated by including an integral action state on $\boldsymbol{z}_2$, but this will not be treated here.

6.2 Thrust Allocation

The generalized forces calculated by the high-level controller has to be distributed to the actuators available in terms of control input $\boldsymbol{u} \in \mathbb{R}^r$, where r denotes the number of control inputs. By assuming linearity, the control force due to a propeller can be written as:

$$T = ku, \quad (6.46)$$

where k is the force coefficient, and u is the control input depending on the actuator. For the azimuth thrusters treated in this thesis, the input is typically shaft speed, torque, or power. See Smogeli (2006) for an in-depth description of different actuator models and thruster parameters.

A mapping between the actuator forces $\boldsymbol{T} = \boldsymbol{K}\boldsymbol{u}$, where $\boldsymbol{K} \in \mathbb{R}^{r \times r}$ is the diagonal force coefficient matrix, to the desired virtual control $\boldsymbol{\tau}_d \in \mathbb{R}^n$, is usually expressed through a *linear actuator model*:

$$\boldsymbol{\tau}_d = \boldsymbol{B}(\boldsymbol{x}_{thr}, t)\boldsymbol{K}\boldsymbol{u} = \boldsymbol{B}(\boldsymbol{x}_{thr}, t)\boldsymbol{T}. \tag{6.47}$$

Here $\boldsymbol{B}(\boldsymbol{x}_{thr}, t) \in \mathbb{R}^{n \times r}$ represents the *thrust configuration matrix* representing the geometric relationship which transforms the individual thruster forces into the commanded forces and moments. $\boldsymbol{x}_{thr}$ includes both position and orientation of each thruster, and t denotes time. The ReVolt concept vessel treated in this thesis, represented in Section 2.7, has three azimuth thrusters: two in the stern and one in the bow. Thus, the thrust configuration matrix depends on the orientation of each thruster, and can be modeled as:

$$\boldsymbol{B}(\boldsymbol{x}_{thr}) = \begin{bmatrix} \cos\alpha_1 & \cos\alpha_2 & \\ \sin\alpha_1 & \sin\alpha_2 & \\ l_{x,1}\sin\alpha_1 - l_{y,1}\cos\alpha_1 & l_{x,2}\sin\alpha_2 - l_{y,2}\cos\alpha_2 & \\ & & \cos\alpha_3 \\ & & \sin\alpha_3 \\ & & l_{x,2}\sin\alpha_3 - l_{y,2}\cos\alpha_3 \end{bmatrix}, \tag{6.48}$$

where α_i is angle of thruster number i, and $l_{x,i}$ and $l_{y,i}$ denotes the longitudinal and transverse distance from thruster number i to the center of gravity (CG), respectively.

The primary objective of the control allocation is to compute the actuator forces $\boldsymbol{T}$ that ensures that the commanded virtual control $\boldsymbol{\tau}_d$ is produced jointly by the efforts at all time t (Johansen and Fossen, 2013). In this thesis, we are faced with an *overactuated* problem ($r > n$), resulting in that the inverse problem of finding $\boldsymbol{T}$ is ill-posed. For such a problem, a unique solution generally does not exists. Also, due to physical constraints, finding an exact configuration satisfying the objective is not always possible. Thus, the control allocation must usually degrade its performance objectives to minimize the allocation error $\boldsymbol{\tau}_d - \boldsymbol{\tau}$, where $\boldsymbol{\tau}$ is the "actual" force applied by the thrust allocation.

The problem is naturally faced as a model-based optimization problem, depending on available actuators. The control allocation problem is a well-studied with a vast amount of literature addressing it. For an overview of different approaches, refer to Johansen and Fossen (2013).

6.2.1 Extended Thrust Formulation for the Concept Vessel ReVolt

The thrust allocation algorithm used in this thesis is provided by DNV GL and uses the *Extended Thrust Formulation* approach for rotatable actuators (Lindfors, 1993; Sørdalen, 1997). Equation (6.48) is nonlinear because of the orientation of each thruster. This implies that a nonlinear optimization problem must be solved to minimize the allocation error. As earlier mentioned, nonlinear optimization problems are hard to solve efficiently. To avoid this, we extend our configuration matrix by dividing each rotatable thruster force into two components; one x and one y component. The *extended thrust configuration* matrix for the ReVolt becomes:

$$\boldsymbol{\tau}_e = \boldsymbol{B}_e \boldsymbol{T}_e = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ -l_{y,1} & l_{x,1} & -l_{y,2} & l_{x,2} & -l_{y,3} & l_{x,3} \end{bmatrix} \begin{bmatrix} T_{x,1} \\ T_{y,1} \\ T_{x,2} \\ T_{y,2} \\ T_{x,3} \\ T_{y,3} \end{bmatrix}, \tag{6.49}$$

where the subscript e denotes extended. Here $\boldsymbol{B}_e \in \mathbb{R}^{n \times q}$ and $\boldsymbol{T}_e \in \mathbb{R}^q$, where $q > r$. The total force T_i and orientation α_i for thruster number i can be calculated by:

$$T_i = \sqrt{T_{x,i}^2 + T_{y,i}^2}, \tag{6.50}$$

$$\alpha_i = \text{atan2}(T_{y,i}, T_{x,i}), \tag{6.51}$$

where atan2(y, x) is the four-quadrant version of $\arctan(y/x)$.

By neglecting all dynamical constraints (saturation and rate constraints) and, for convenience, choosing a quadratic cost function trying to minimize the applied forces $\boldsymbol{T}_e$, we can formulate the optimization problem as:

$$J = \min_{\boldsymbol{T}_e \in \mathbb{R}^q} \quad \boldsymbol{T}_e^\top \boldsymbol{W} \boldsymbol{T}_e \tag{6.52a}$$

$$\text{subject to} \quad \boldsymbol{\tau}_e - \boldsymbol{B}_e \boldsymbol{T}_e = 0, \tag{6.52b}$$

where $\boldsymbol{W} \in \mathbb{R}^{q \times q}$ is a positive definite weight matrix penalizing the thrust efforts of each respective force component. The issue of not just inverting Equation (6.49) is that $\boldsymbol{B}_e$ is not a square matrix. Usually, $\boldsymbol{B}_e$ has full rank, implying that there is an infinite number of $\boldsymbol{T}_e$'s satisfying Equation (6.49). This is solved by introducing the generalized pseudo-inverse of $\boldsymbol{B}_e$ found by the Moorse-Penrose inverse[2]. Using the pseudo-inverse, the optimization problem in Equation (6.52) has an explicit solution if $\boldsymbol{B}_e$ is full rank:

$$\boldsymbol{T}_e = \boldsymbol{W}^{-1} \boldsymbol{B}_e^\top (\boldsymbol{B}_e \boldsymbol{W}^{-1} \boldsymbol{B}_e^\top)^{-1}. \tag{6.53}$$

[2] $\boldsymbol{B}_e^\dagger = \boldsymbol{B}_e^\top (\boldsymbol{B}_e \boldsymbol{B}_e^\top)^{-1}$

Rank deficiency of $\boldsymbol{B}_e$ means that no forces and moments can be produced in a specific direction, resulting in that not all desired forces in $\boldsymbol{\tau}_d$ can be achieved. Usually, this is avoided by thoughtful placement of the actuators during design. Still, the algorithm should be able to handle it due to singularities and possible thruster-losses. An additional regularization term, $\epsilon \boldsymbol{I}$, can solve this, ensuring that the inverse always exists:

$$\boldsymbol{T}_e = \boldsymbol{W}^{-1}\boldsymbol{B}_e^\top(\boldsymbol{B}_e\boldsymbol{W}^{-1}\boldsymbol{B}_e^\top + \epsilon\boldsymbol{I})^{-1}, \tag{6.54}$$

where $\epsilon \geq 0$ is a small constant.

Since no constraints are imposed on, for example, rate limitations in the change of thrust and direction when solving the optimization problem, this must be done post hoc. The simplest way is to saturate the output of the unconstrained solution. This could however lead to sub-optimal allocation. For a more detailed solution, refer to Johansen and Fossen (2013).

6.3 Saturating Element

In order for the maneuvering controller to generate obtainable output for the thrust allocation, the controller must be saturated according to what the real thruster dynamics of the vessel is able to produce. This can be calculated by considering the minimum/maximum forces and moments the vessel can produce in each DOF. For example, using the data and measurements from the ReVolt vessel, given in Table 7.1, we get:

$$X^{max} = \sum_{i=1}^{3} \tau_{X,i}^{max} = 2 \cdot 20.5\ \mathrm{N} = 41\ \mathrm{N}, \tag{6.55}$$

$$Y^{max} = \sum_{i=1}^{3} \tau_{Y,i}^{max} = 2 \cdot 20.5\ \mathrm{N} + 9\ \mathrm{N} = 50\ \mathrm{N}, \tag{6.56}$$

$$N^{max} = \sum_{i=1}^{3} \tau_{N,i}^{max} = 2 \cdot (20.5\ \mathrm{N} \cdot 1.12\ \mathrm{m}) + 9\ \mathrm{N} \cdot 1.08\ \mathrm{m} \approx 55\ \mathrm{Nm}. \tag{6.57}$$

Here $\tau_{X,i}^{max}$ denotes maximum force in surge obtained by thruster number i. The same goes for the others DOFs.

Chapter 7

Experimental Platform and Implementation

The experimental platform used is the same as in the author specialization project (Knædal, 2019), hence the general description of the concept vessel ReVolt, given in Section 2.7, is based on that. Further, the simulator used is described and implementation details are given.

7.1 The ReVolt Test Platform

The ReVolt is equipped with three azimuth thrusters; two main thrusters in the stern and one retractable in the bow. See Figure 7.1 for dimensions. The *Froud number* of the ReVolt vessel is $F_n = 0.27$. According to Fossen (2011c), the vessel can then be classified as a low-speed displacement vessel ($F_n < 0.4$). This means that buoyancy force dominates relative to the hydrodynamic forces (added mass and damping). Thus, the vessel model used, given in Section 2.1, is well suited.

Alfheim and Muggerud (2017) have calculated the hydrodynamic coefficients and other relevant physical parameters. DNV GL has performed a towing tank test to obtain thrust coefficients and the forces they provide. See Table 7.1 for an overview of the placement and the force each respective thruster provide. In addition to the thrusters, the vessel is equipped with, among other things, an embedded computer, a global positioning system (GPS) aided by an inertial measurement unit (IMU), and a heading vector. Further, it is equipped with a Velodyne lidar, a Ladybug camera, and an automatic identification system (AIS) receiver for object detection. For a detailed description of different components onboard and technical details, see Alfheim and Muggerud (2017) and Norbye (2019).

Table 7.1: Thruster placements and their maximum and minimum force contributions. The enumeration of the thrusters are (1) port in stern, (2) starboard in stern, and (3) bow. Courtesy: Alfheim and Muggerud (2017).

Thruster	l_x $[m]$	l_y $[m]$	$T_{max/min}$ $[N]$
1	-1.12	-0.15	± 20.5
2	-1.12	0.15	± 20.5
3	1.08	0	± 9

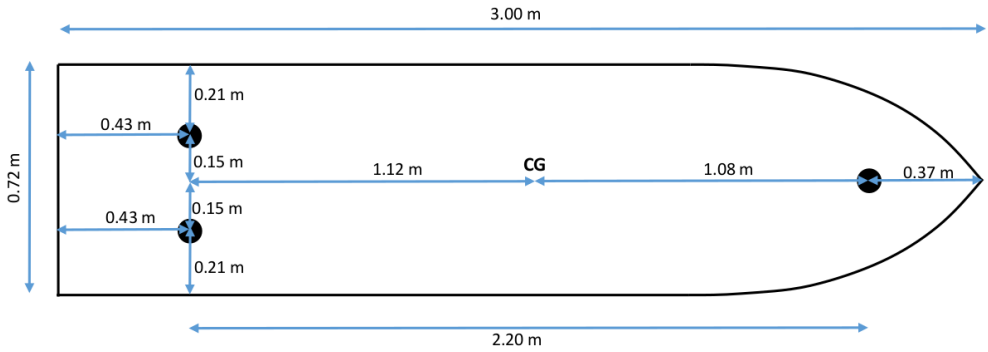


Figure 7.1: Dimensions of the ReVolt model scale ship. Courtesy: Alfheim and Muggerud (2017).

7.2 Simulator

It is an absolute necessity to be able to test software implementations before testing on the real physical model to prevent hazardous events from occurring. DNV GL has developed an Open Simulation Platform (OSP) to establish an ecosystem for models and co-simulations in the maritime industry[1]. The OSP system facilitates efficient construction of digital twin systems and vessels in order to carry out testing and simulation of the software. Such a digital twin has been made of the ReVolt and is the simulator used in this thesis.

The OSP runs on a Microsoft Windows operating system (OS), while the control system runs on a Linux OS. The computers connect through an category 5E Ethernet cable. By separating the simulation environment and the control system, the distance between real-life testing and simulation is minimized since the control system does not know the difference between the real ReVolt and the simulated one.

The OSP incorporates functionalities for modeling environmental forces, thruster loss, and virtual obstacle ships. This gives the opportunity for a more realistic simu-

[1] `https://opensimulationplatform.com/`

Figure 7.2: A visualization of the OSP's GUI with ReVolt in Dorabassenget in Trondheim.

lation environment. Figure 7.2 shows the graphical user interface (GUI) of the OSP. Note that all results obtained using the OSP belongs to DNV GL.

7.3 Software

The implemented code running the control system is written in C++ and Python using the Robotic Operating System (ROS) framework[2]. ROS is a flexible framework for writing robot software that aims to make robotic development faster, more robust, and easy to share[3]. ROS serves as a "core" of the entire program running. Processes running in ROS are represented as nodes in a graph structure, connected by edges called topics. The nodes can communicate with each other through topics, provide services for one another, and share obtained data from a database called the parameter server. The nodes can be written in the preferred programming language. The ROS graph structure convention makes it easy to separate the complete system into subsystems, as represented in Figure 1.1. This again makes it easier to verify and test subsystems for a risk and classification society such as DNV GL.

Most of the implementations are done in Python because of its broad support and available libraries for scientific and numerical computations. The CVXOPT library is used for solving the quadratic optimization problem formulated in Section 4.2.5[4].

[2] https://www.ros.org/
[3] https://www.ros.org/about-ros/
[4] https://cvxopt.org/

Further, the NumPy[5]and SciPy[6] libraries has been used extensively throughout the implementation. The ROS package `tuw_voronoi_graph`, created by Binder (2017), has been forked and used for Voronoi partitioning and the `costmap_2d` package[7] is used for creating costmaps. The state estimates used are obtained from an EKF observer implemented by Andreas Bell Martinsen and written in Python. The thrust allocation algorithm is provided by DNV GL and written in Java. The implemented system has been run and tested on Dell Latitude E7440 with an Intel Core i5-4310U CPU with four cores, and a 2.00 GHz clock. The OS used is Ubuntu 16.04 LTS, 64-bit.

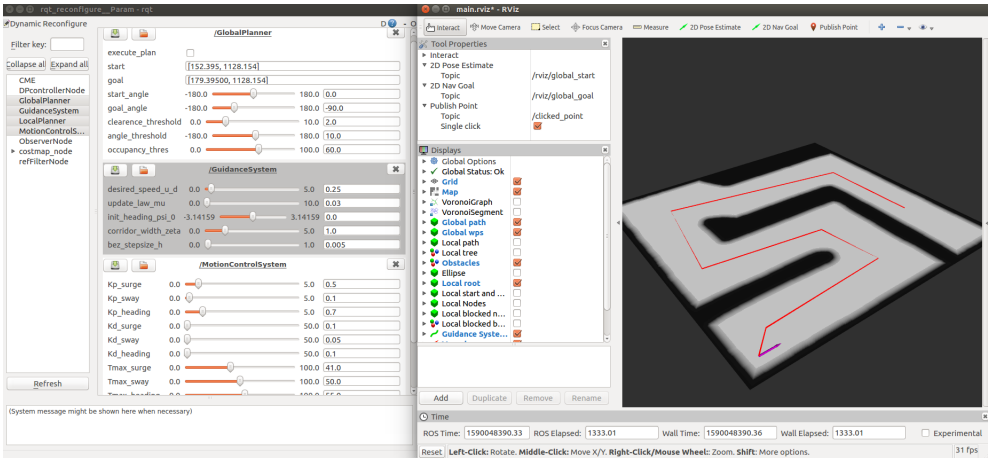


Figure 7.3: A screenshot of the GUI for the control system.

7.4 Implementation

Several ROS packages have been created, each one corresponding to a subsystem in Figure 1.1. Check out the GitHub repository created[8]. The computation graph illustrating how the actual implementation communicates is shown in Figure 7.4. The author would like to emphasize that time has been devoted to writing easily understandable code that is well commented such that it is easy to read and develop further. A new control mode has been added to the ReVolt operating system. A

[5] `https://numpy.org/`

[6] `https://www.scipy.org/`

[7] `http://wiki.ros.org/costmap_2d`

[8] `https://github.com/magnuok/Autonomous-Path-Planning-and-Maneuvering-of-a-Surface-Vessel`

GUI using a Qt-based framework[9] together with RViz[10], a 3D visualization tool for ROS, have been made to control the system. The GUI makes it possible to place initial and target waypoint by clicking directly on the desired location on the map. Simple display tools have been implemented to visualize the implemented algorithms behavior in real-time, which can be turned on and off. Furthermore, configuration and tuning of relevant parameters online have been added to the GUI. See Figure 7.3 for a screenshot of the GUI.

The pragmatic approach was only implemented and simulated in MATLAB. An own GitHub repository was made for the MATLAB code[11]. Data has been recorded using rosbags[12] in both simulations and the experiment. For plotting, both MATLAB and the matplotlib library[13] in Python have been used. Scripts for experimental testing have been made — for example, a script to create OGMs from real data obtained from ReVolt.

[9] http://wiki.ros.org/rqt
[10] http://wiki.ros.org/rviz
[11] https://github.com/magnuok/path_generator
[12] http://wiki.ros.org/rosbag
[13] https://matplotlib.org/index.html
[14] http://wiki.ros.org/rqt_graph

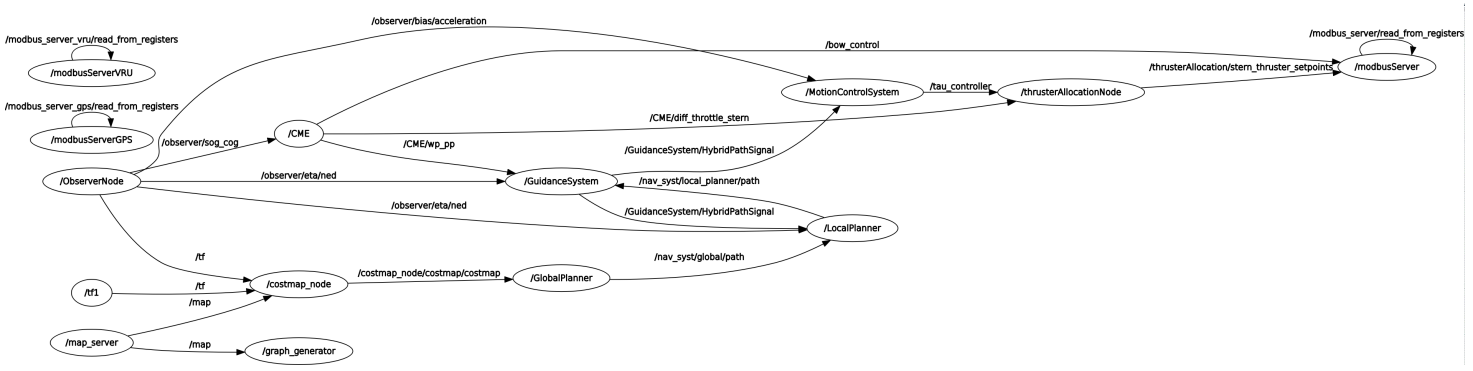


Figure 7.4: The figure shows the computation graph of the implemented GNMC system. The vertices in the graph represent ROS nodes, while the edges between them are ROS topics which the nodes communicate through. The graph is generated by the GUI plugin `rqt_graph` package[14]. Nodes containing "modbus" in the name belongs to the OSP. The CME node is the "Control Mode Executor", which is the superior node deciding in which control mode the vessel should be. The rest is self-explanatory.

Chapter 8

Simulations

This chapter presents simulations, discussions, and results to verify performance for each designed and implemented subsystem. Lastly, a complete simulation is done to verify that the complete system works seamlessly. All simulations involving the control system of the vessel are conducted using the OSP simulator described in Section 7.2. The parameter values used in the control model of ReVolt is given in Appendix A.

8.1 Guidance System

An S-shaped simulation was created to illustrate the performance of the guidance system. The simulation gives a comparison between the pragmatic and optimization approach. The arc length, defined in Equation (2.19), is used as the cost function. The corridor width $\zeta = 1.5$ m is equal for all path segments and the desired speed, $u_d(t) = 0.2$ m/s, is constant. The tuning parameters $\varepsilon_1, \varepsilon_2$, and ε_3 are all set to the same value ε. The desired path consists of a total of eleven waypoints representing an S-shape, given in Appendix D. Relevant parameters and result is found in Table 8.1 for both methods.

Table 8.1: Parameters and results for the S-shaped simulation for the pragmatic and optimization approach.

Pragmatic	**Optimization**
Parameters	
$\zeta = 3$ m	$\zeta = 3$ m
$\mu = 3$	$\varepsilon = \zeta/6$
$\delta_{min} = 1.5$ m	
$u_d(t) = 0.2$ m/s	$u_d(t) = 0.2$ m/s
$u_d^t(t) = 0.0$ m/s^2	$u_d^t(t) = 0.0$ m/s^2
Results	
$L = 53.88$ m	$L = 53.34$ m

8.1.1 Pragmatic Approach

The control points are placed according to Equations (4.12) to (4.14). It should be noted that the placement of the control points is tuned for the given scenario, as emphasized in Line 8. From Figure 8.1, one can see that the pragmatic approach gives a smooth S-formed shape. The control points are placed with equal distance. Furthermore, they are placed inside the corridor, resulting in the curve being inside the corridor. From Figure 8.2, one can see that the rate of change in curvature is continuous, entailing that the hybrid curve is $\mathcal{C}^3$ continuous. The curvature has its peaks right after the waypoints, which coincides with where the turns are performed. The cost function is evaluated to $L = 53.88$ m. Since we are performing similar maneuvers, one can see that a pattern repeats itself from both the curvature, rate of change in curvature, and the speed profile in Figure 8.2.

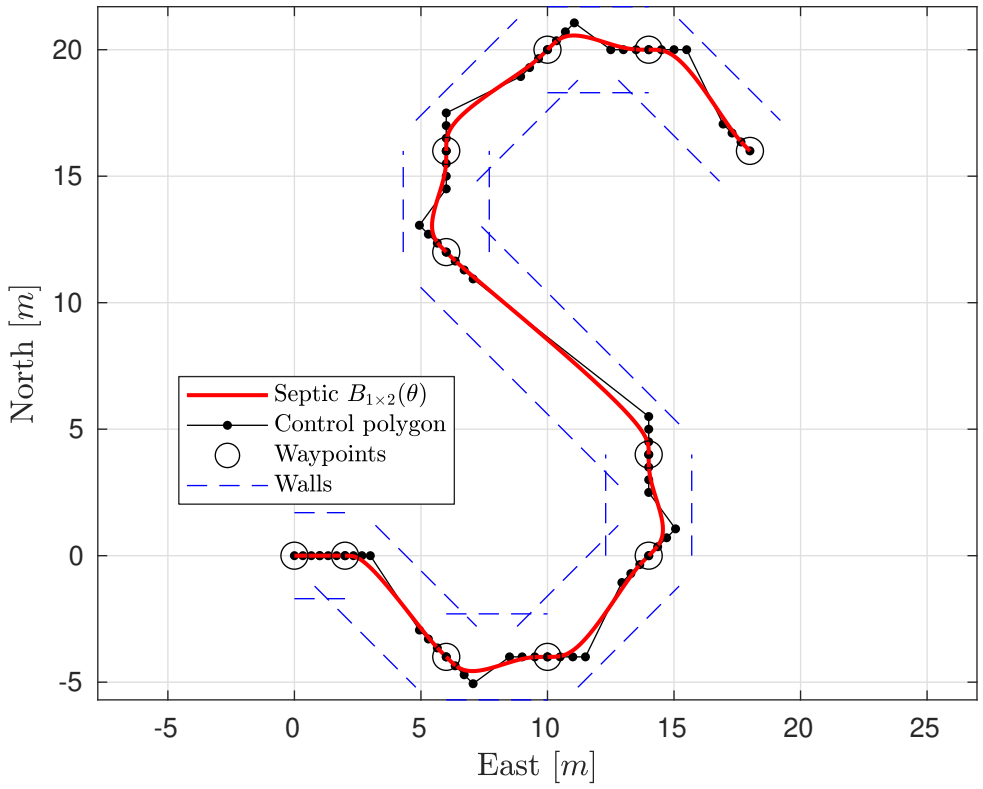


Figure 8.1: A xy-plot of desired path of S-shaped simulation for the pragmatic approach. The Bézier curve is drawn in red on top of its control polygon.

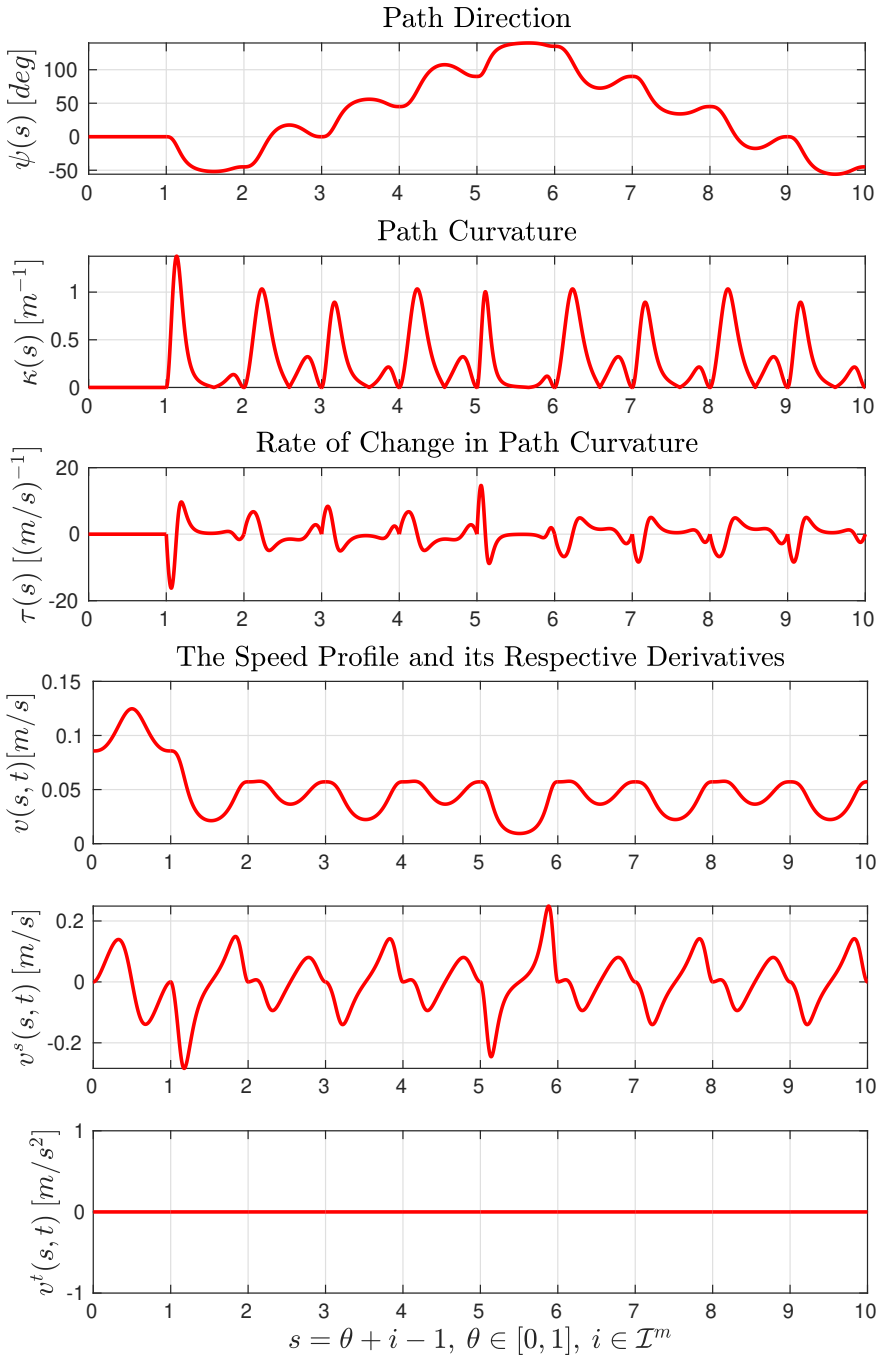


Figure 8.2: The direction, curvature, rate of change in curvature, and the speed profile and its respective derivatives for the Bézier curve in Figure 8.1.

8.1.2 Optimization Approach

The control points are placed by solving the optimization problem formulated in Section 4.2.5. All control points are placed inside the corridor, resulting in the curve being inside the corridor. The optimization approach delivers a smooth S-formed shape. The cost function is evaluated to $L = 53.34\,\mathrm{m}$. As for the pragmatic approach, we have a repeating pattern in curvature, rate of change in curvature, and the speed profile in Figure 8.4. Notice that the control points are placed very close to the waypoints, which contributes to a shorter arc length. However, in return, one can see that the curvature has high peaks before and after the waypoints. This is also the case for the rate of change in curvature.

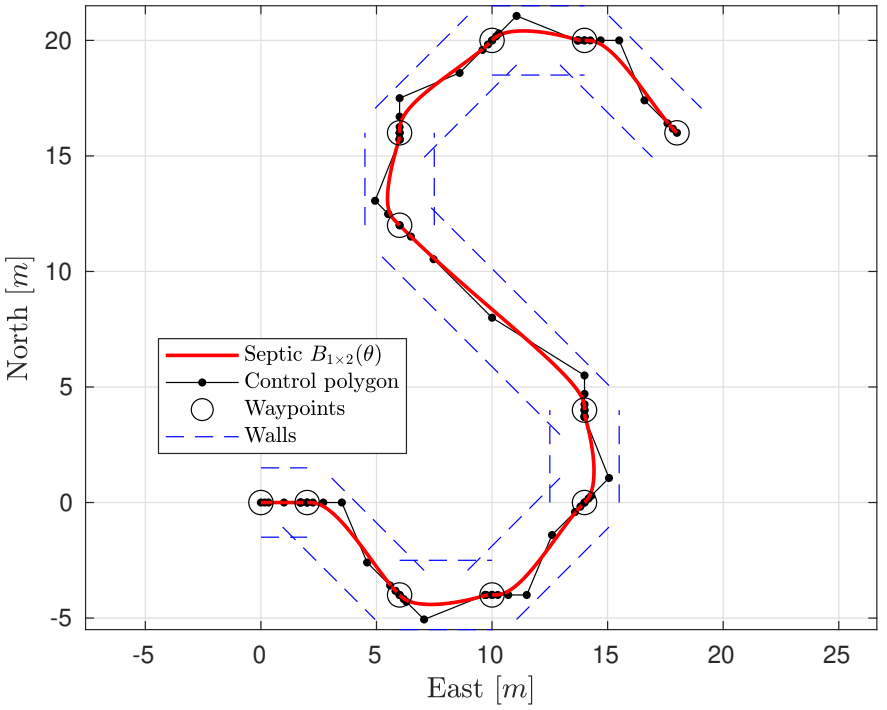


Figure 8.3: A xy-plot of desired path of S-shaped simulation for the optimization approach. The Bézier curve is drawn in red on top of its control polygon.

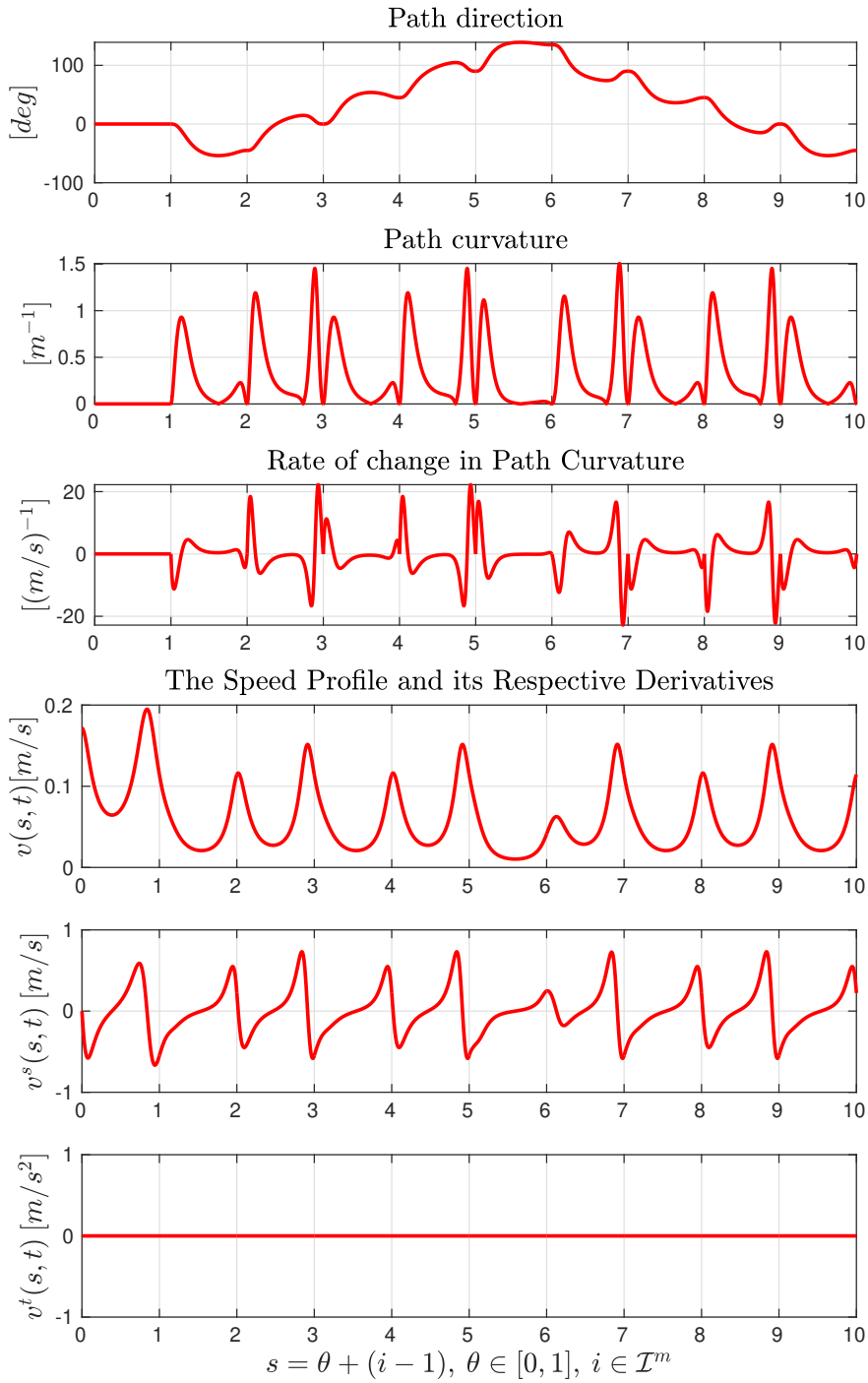


Figure 8.4: The direction, curvature, rate of change in curvature, and the speed profile and its respective derivatives for the Bézier curve in Figure 8.3.

8.1.3 Results

Both methods delivers $\mathcal{C}^3$ continuous path, such that the required outputs from Section 3.3 can be produced. According to the objective function, the optimization approach achieves a slightly better result. Notice how close to the waypoints the optimization procedure puts the control points. In fact, by placing the control infinitely close, the path will degenerate to only consist of straight-line path segments. This coincides with our objective function, which always tries to minimize the arc length. However, seen from Figures 8.2 and 8.4, the optimization approach has higher peaks in curvature, and thus, must perform sharper turns. This is also the case for the rate of change in curvature.

One can see that since the pragmatic approach is placing the control points further away, the path tends to start turning later after the waypoints, compared to the optimization approach. As a result, more maneuvering actions are performed by the pragmatic one. This is clearly seen from the path direction plot in Figures 8.2 and 8.4. The pragmatic approach has greater fluctuations in direction than the optimization approach.

The speed profile, defined in Equation (4.48), has its peaks around the waypoints for both the pragmatic and optimization approach. The optimization approach has higher peaks in speed profile compared to the pragmatic one. Remember that the speed profile is obtained by dividing the desired speed by the norm of the derivative. Logically, the derivative has its minimum around the waypoint since the curvature and rate of change in curvature are set equal to zero in the waypoints, making it a straight line. Thus, the derivative also has its minimum here. Between the waypoints, the derivative must have its maximum because of continuity. Thus the opposite behavior is expected from the speed profile, seen in Figures 8.2 and 8.4. The derivative with respect to time is zero for both methods, as expected since the desired speed is constant.

8.2 Control System

The controller settings were tuned by a trial and error approach and tested on different simulations. The unit tangent gradient update law is used. The settings were $\tilde{\boldsymbol{K}}_1 = \text{diag}(0.5, 0.1, 0.7)$, $\boldsymbol{K}_2 = \text{diag}(0.1, 0.05, 0.1)$, and $\mu = 0.03$. The controller was saturated with $\boldsymbol{\tau}_{max} = [41\ \text{N}, 50\ \text{N}, 55\ \text{Nm}]^\top$, as calculated in Section 6.3. For each simulation the vessel was put to rest in a dynamic position (zero speed), and then the vessel was commanded online to move along the path with a desired speed $u_d = 0.25$ m/s. The simulations are conducted with thrust losses but no environmental forces acting.

During simulations, it was discovered that the thrust allocation was struggling to

allocate desired forces in sway. A workaround was to fix the angle of the bow thruster to 90°(that is $\alpha_3 = \pi/2$ in Equation (6.49)) to simplify the allocation problem. This resulted in much better results, without the loss of any actuation. Two simulations are performed to illustrate the performance of the controller; a straight-line and an S-shaped maneuver.

8.2.1 Simulation 1: Straight-line Maneuver

The vessel is commanded to move in a straight line for 30 m. The results of the straight-line simulation are presented in Figures 8.5 to 8.8.

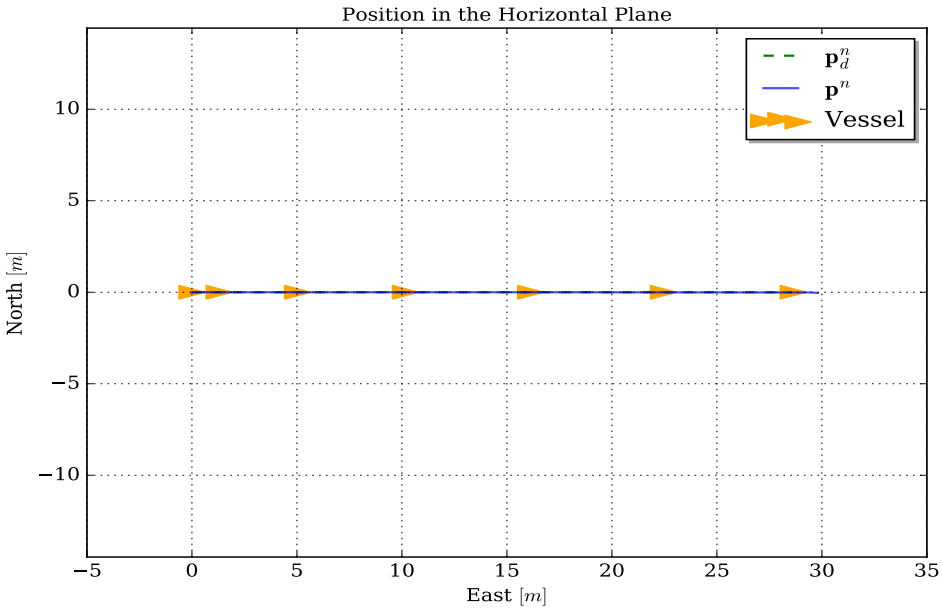


Figure 8.5: Position in horizontal plane.

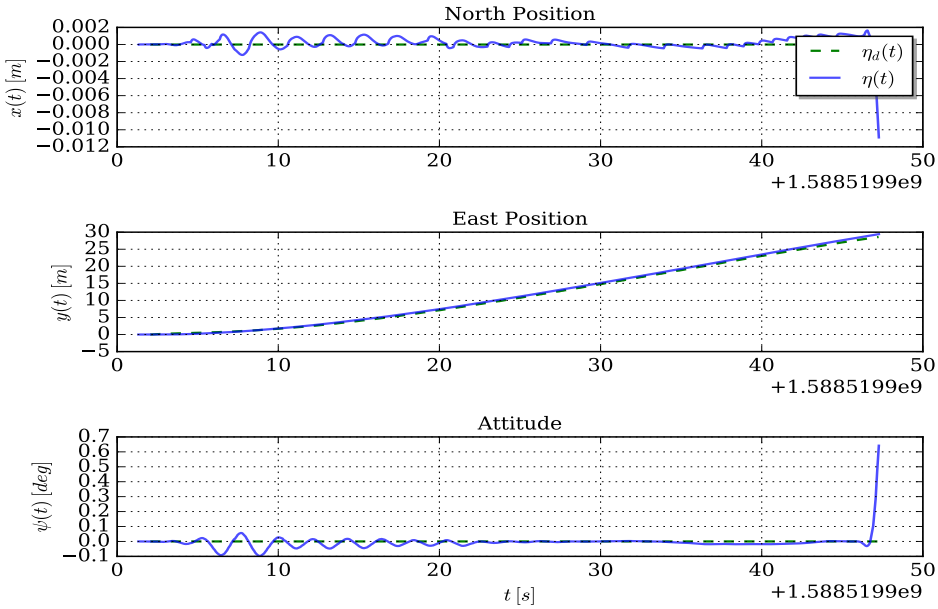


Figure 8.6: Position versus time for each DOF.

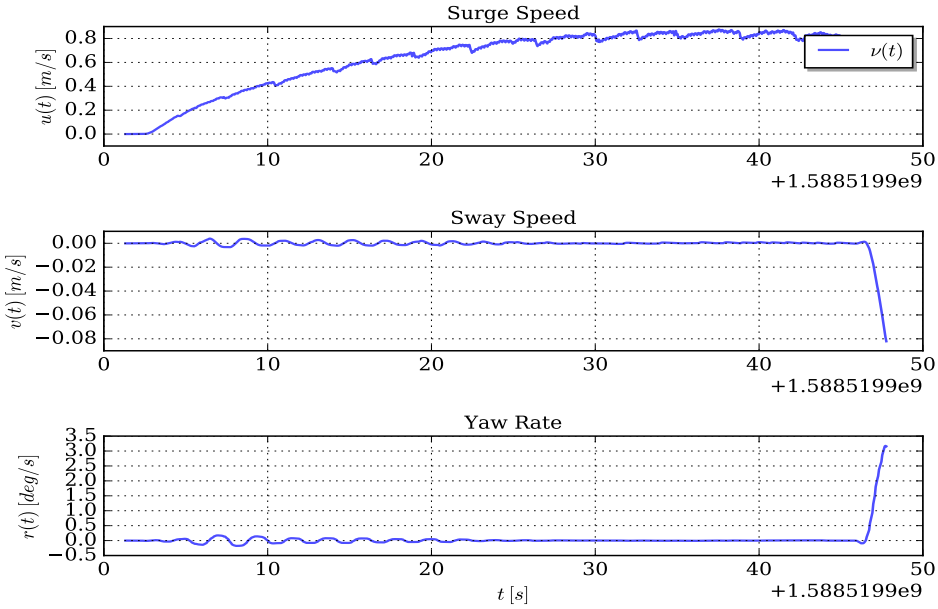


Figure 8.7: Speed versus time for each DOF.

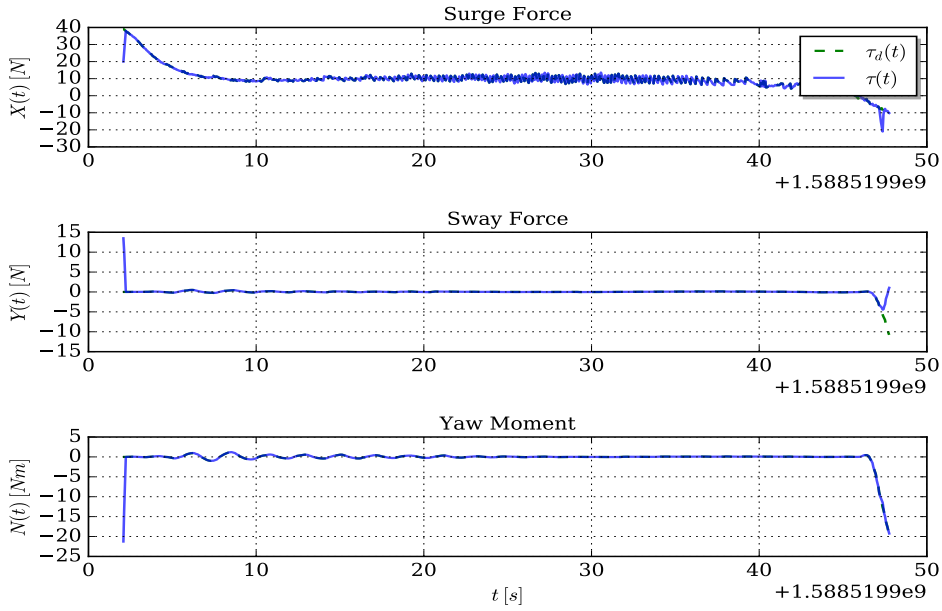


Figure 8.8: Forces versus time for each DOF.

8.2.2 Simulation 2: S-shaped Maneuver

The vessel is commanded to move along an S-shaped path, equal to the one provided by the optimization approach of the guidance system in Section 8.1.2. The waypoints used are the same and given in Appendix D. The results of the S-shaped simulation are presented in Figures 8.9 to 8.12.

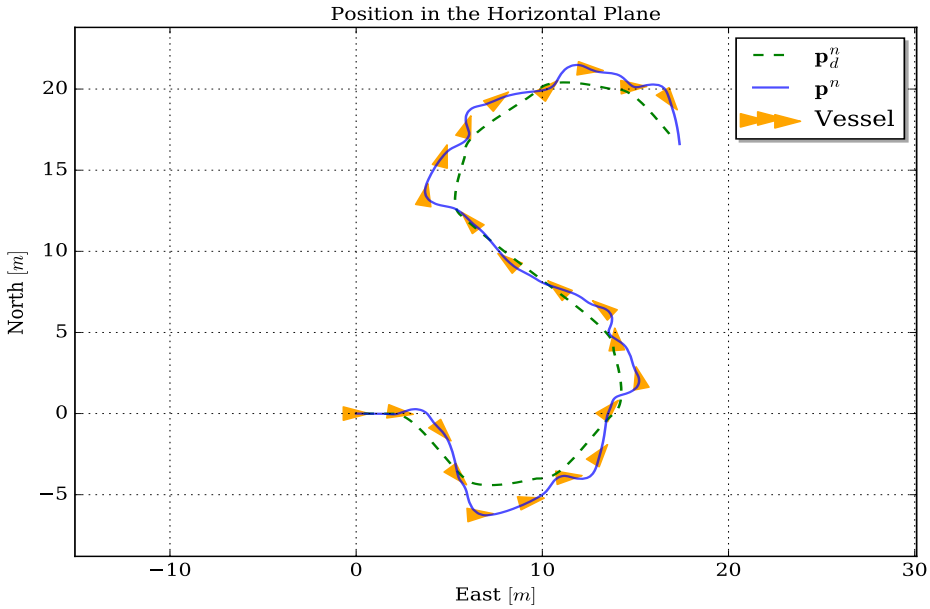


Figure 8.9: Position in horizontal plane.

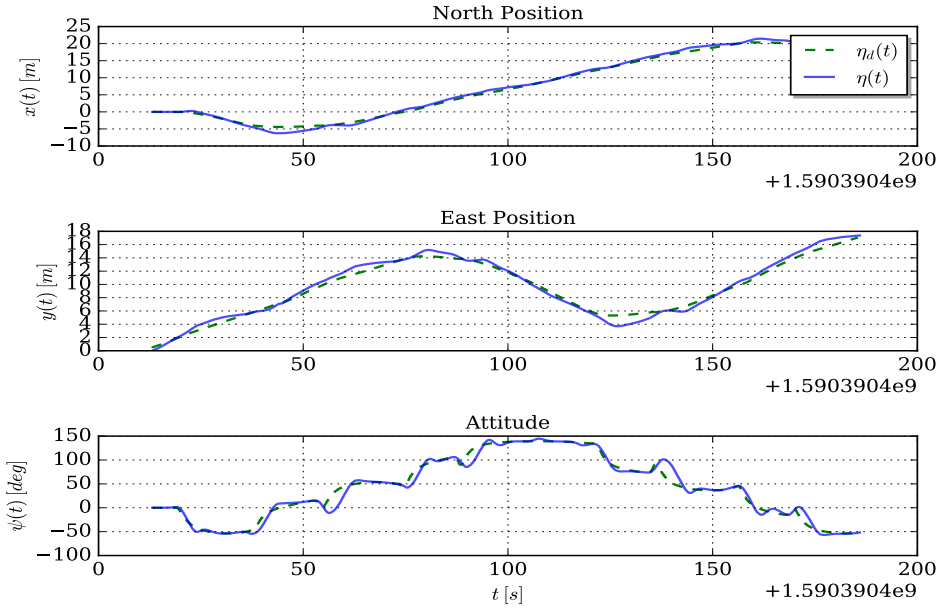


Figure 8.10: Position versus time for each DOF.

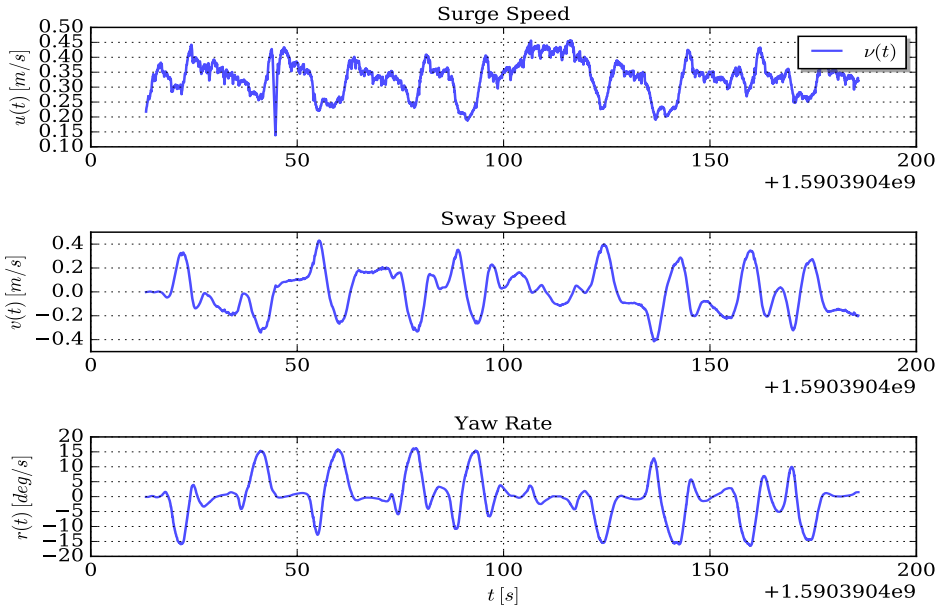


Figure 8.11: Speed versus time for each DOF.

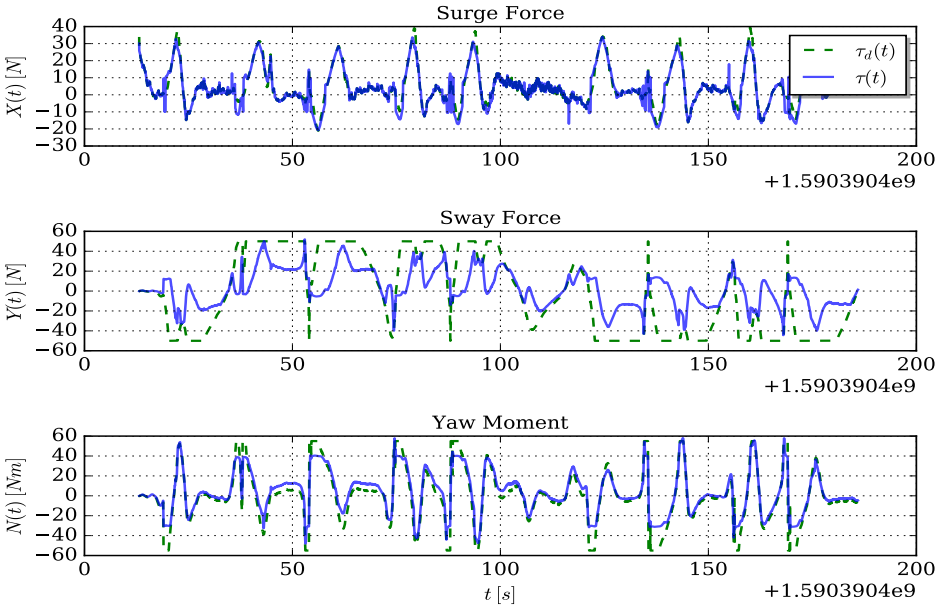


Figure 8.12: Forces versus time for each DOF.

8.2.3 Results

Figures 8.5 and 8.9 shows that the controller can trace the desired pose with good accuracy. From the S-shaped simulation in Figure 8.9, one can see that the vessel is waving a little back and forth in sway. This could be removed to some degree by better tuning. From Figure 8.12, one can see that the "actual" forces and moment from the thrust allocation follow quite well the desired force and moment in surge and yaw, but struggle a bit in sway. Intuitively, it can be seen from Figures 8.9 and 8.10 that the controller has a more challenging time following sharp corners accurately. Also, it seems to struggle with satisfying the speed assignment. This could be caused by suboptimal tuning as well as poor state estimates obtained from the EKF observer.

Heavy oscillations for both simulations characterize the desired surge force. This is an undesirable behavior that could cause unnecessary wear and tear on the propulsion equipment. A problem discovered under the simulations was that no restrictions on the operating frequency for the different subsystems were imposed, which could be the reason for this. No restriction on operating frequency also causes high and low-level control to work equally fast, which can lead to bandwidth trouble. The lack of restrictions on the operating frequency also causes very tight plots with no delay between desired and actual. If real-life experiments are performed, this problem should be solved. The behavior can also indicate too aggressive tuning parameters in surge.

Most likely, the parameter convergence will be different for different paths and desired speeds. More time can be put into the tuning of the controller to achieve better performance, but for our purpose, the performance obtained seemed sufficient.

8.3 Navigation System: Global Planner

The global planner has been implemented and tested on cost maps created from OGMs. The inflation radius is set to $1\ \mathrm{m}$. Two scenarios are simulated to illustrate the performance of the global path planner. For both simulations, the segment length between each node in the Voronoi roadmap is set to maximum $10\ \mathrm{m}$. The occupancy threshold, ranging in the interval $[0, 100]$, is set to 60.

8.3.1 Simulation 1

The following simulation illustrates the behavior of the global planner. The simulation solve the global path planning problem from $\mathbf{WP}_0 = (27\,\mathrm{m}, 3\,\mathrm{m})$ to $\mathbf{WP}_{goal} = (20\ \mathrm{m}, 27\ \mathrm{m})$, with clearance constraint $d_{min} = 1\ \mathrm{m}$ and angle threshold $\psi_{thres} = 10^\circ$. The results are presented in Figure 8.13.

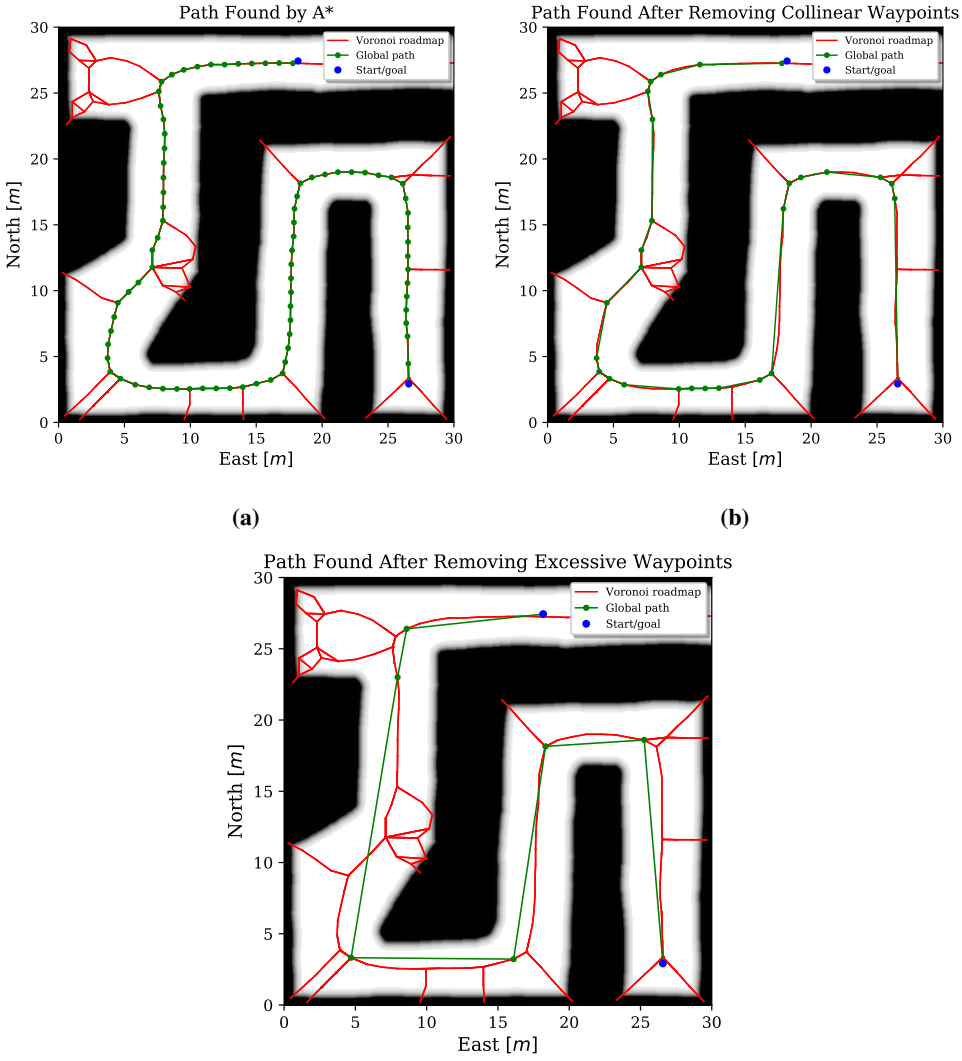
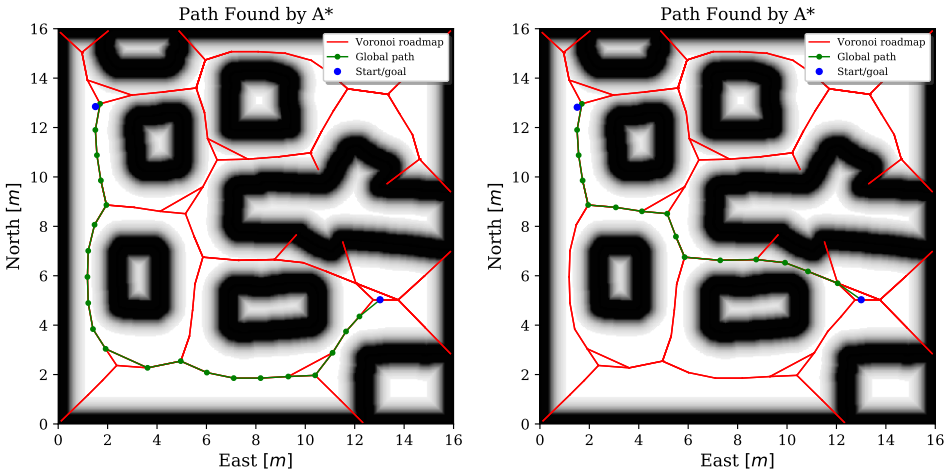


Figure 8.13: Results for Simulation 1 of the global planner.

8.3.2 Simulation 2

The following simulation illustrates the behavior of the global planner with and without a clearance constraint. Two simulations are performed, one with clearance set to $d_{min} = 0$ m, and another with $d_{min} = 0.5$ m. Both simulations solve the global path planning problem from $\mathbf{WP}_0 = (13\text{ m}, 5\text{ m})$ to $\mathbf{WP}_{goal} = (1\text{ m}, 13\text{ m})$, with angle

threshold $\psi_{thres} = 10°$. The OGM is the same as shown in Figure 2.7, created by Binder (2017). The results are presented in Figure 8.14.



(a) **(b)**

Figure 8.14: Results for Simulation 2 of the global planner. Subfigure **(a)**, **(c)**, and **(e)** shows results with no clearance constraint, while **(b)**, **(d)**, and **(f)** shows results with 1 m clearance constraint.

8.3.3 Results

The results obtained from Simulation 1 and 2 shows how the global planner can generate waypoints that guarantee that the vessel is always at a safe distance from obstacles in a cluttered environment. In Figure 8.13a the path found from A* is visualized. Figure 8.13b shows the path after removing collinear waypoints. Finally, Section 8.3.1 shows the final path after removing excessive waypoints. One can see that the algorithm generates a safe path defined by few waypoints. Simulation 2 illustrates the behavior of the global planner with and without clearance constraints. From Figures 8.14a and 8.14b, we can see that the A* algorithm obtains different initial paths. In Figure 8.14a, no clearance constraint is imposed; thus, the path goes through the narrow passage to obtain the shortest path. In Figure 8.14b, one can see that the A* chooses a longer path, but a safe one.

Note that a plotting error causes the triangles to appear in the Voronoi roadmaps. The Voronoi roadmap is found using the OGM without any inflation; thus, the nodes close to lethal zones appear inside the inflation.

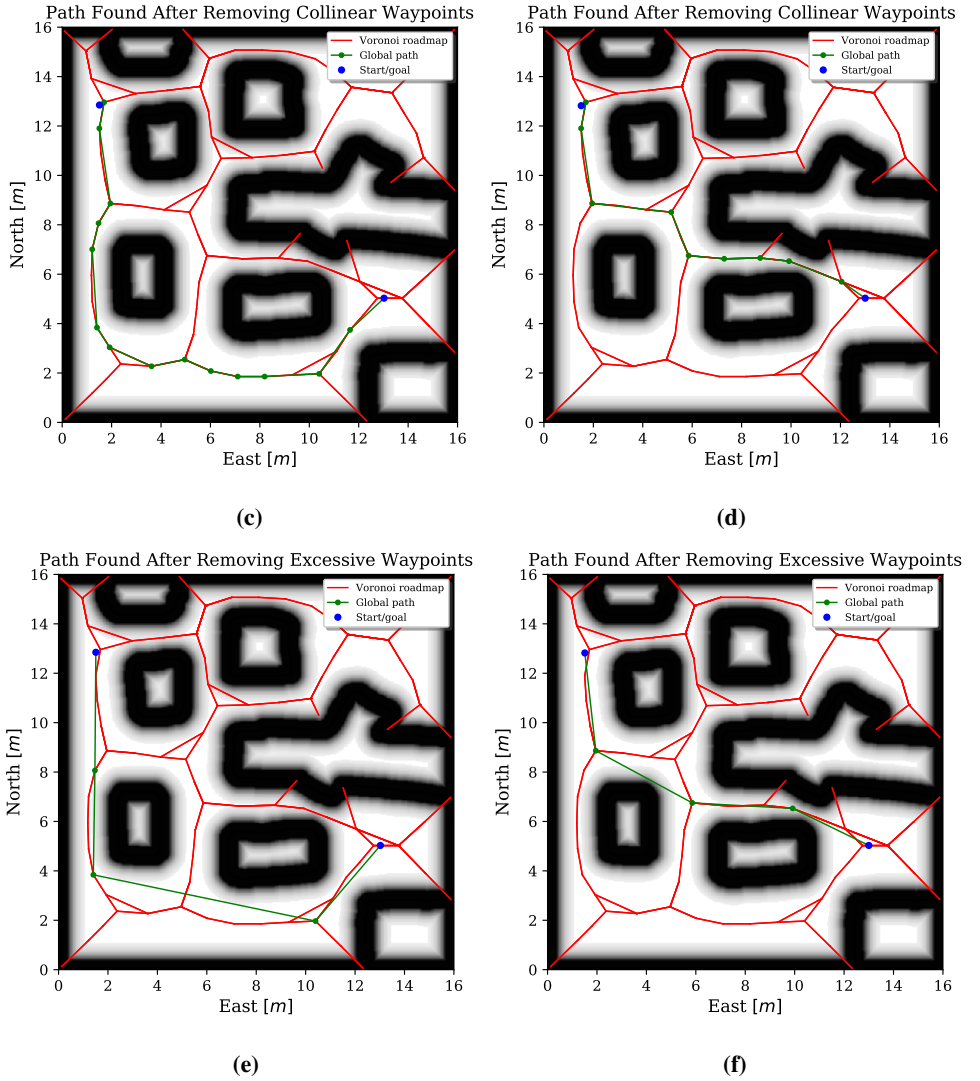


Figure 8.14: Results for global planner for Simulation 2. Subfigure **(a)**, **(c)**, and **(e)** shows results with no clearance constraint, while **(b)**, **(d)**, and **(f)** shows results with 1 m clearance constraint.

8.4 Navigation System: Local Planner

Several parameters need to be tuned to obtain a desired performance for the local planner. Table 8.2 gives an overview and a comprehensive description of each one. The local planner has been implemented and tested on cost maps created from OGMs.

Several simulations are performed to illustrate different aspects of the local planner. The average runtime for expand-and-rewiring 100 nodes was 0.1197 s.

Table 8.2: Tuning parameters for the local planner and their value for each performed simulation. A value set to "-" indicates that the parameter is not relevant for the given simulation.

Param.	Description	S.1	S.2	S.3	S.4	S.5
r_{goal}	Radius of goal region.	1 m	1.5 m	1.5 m	1.5 m	4 m
r_{min}	Min. distance between nodes.	0.5 m	0.2 m	0.2 m	0.2 m	1 m
r_{max}	Max. expanding distance	2 m	2 m	4 m	4 m	6 m
α	Goal sample rate (percentage).	0.05%	0%	0.05%	0.05%	0.05%
δ	Distance cost parameter.	-	1	10	10	10
ϵ	Obstacle cost parameter.	-	0	1	1	1
ζ	Curvature cost parameter.	-	0	1	1	1
ψ_{max}	Max. angle between nodes.	90°	90°	90°	90°	90°
ρ_{max}	Max. "curvature".	10	20	10	10	2
k_{max}	Max. number of node neighbors.	50	15	50	20	100
K	Max. steps for K-step path.	-	3	4	3	5
ι	Roughness of the steering (resolution).	0.1 m	0.1 m	0.1 m	0.1 m	0.1 m
n	Grid dim. for spatial indexing ($n \times n$).	7	7	7	7	8
λ	Occupied thres. for grid $[0, 100]$	30	60	30	30	30
β	Dividing sampling between uniform and ellipse.	-	10^6	2	2	10

8.4.1 Simulation 1: Tree growth

The simulation illustrates how different cost functions affect tree expansion and contribute to significantly different tree topologies. The cost functions represented in Section 5.1.2 are simulated independently from one another. That is, set one of the tuning parameters δ, ϵ, and ζ to one, and the others to zero. At each simulation, 1000 nodes were generated. Except for δ, ϵ, and ζ, the algorithm was tuned using the values given in the S.1 column of Table 8.2. The start and goal pose was set to $x_0 = (8\,\text{m}, 1\,\text{m}, 90°)$ and $x_{goal} = (8\,\text{m}, 14\,\text{m}, 90°)$. Two obstacles are present with an obstacle region $d_{min} = 2$ m. The results are visualized in Figure 8.15.

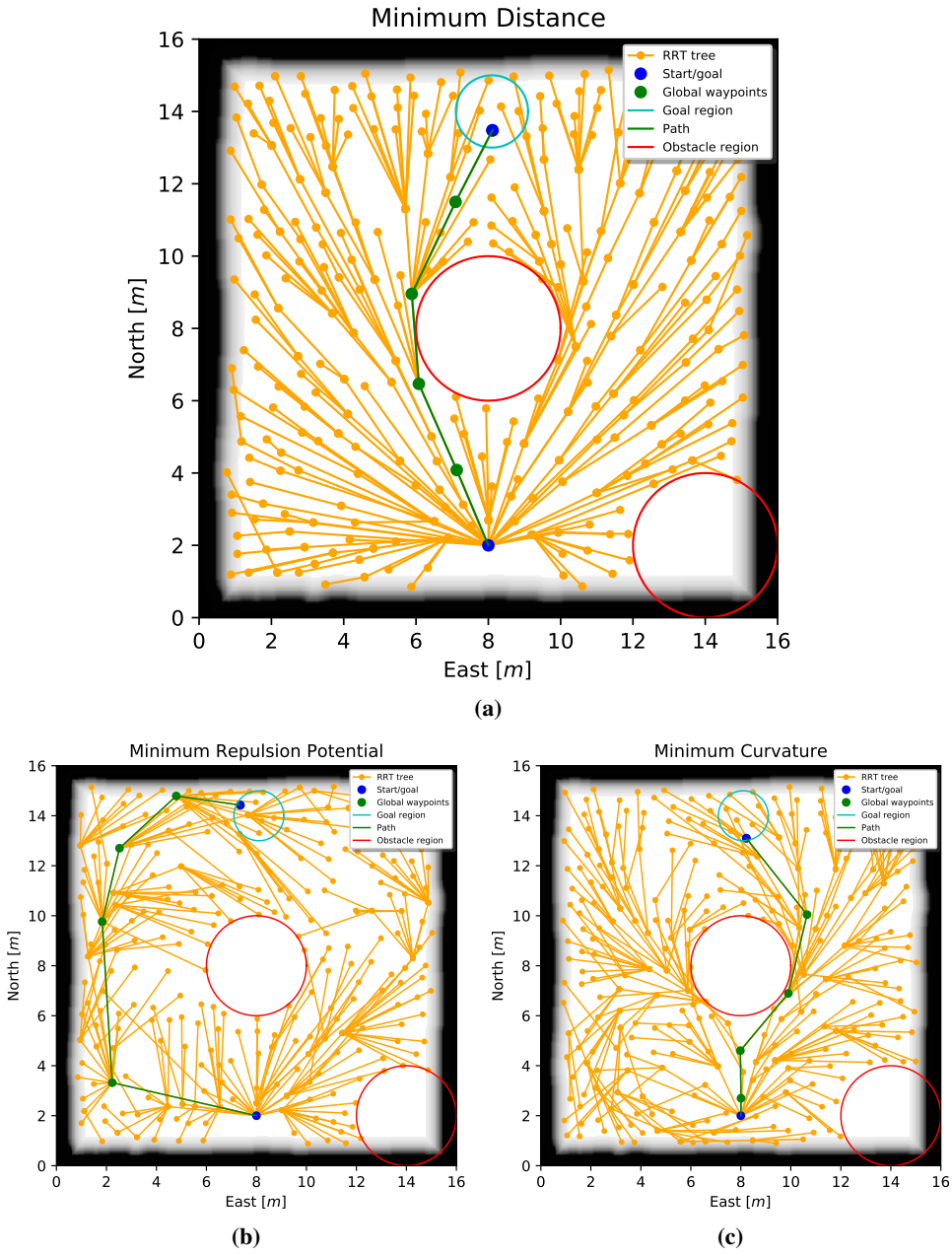


Figure 8.15: The results for the local planner, Simulation 1: Tree Growth.

8.4.2 Simulation 2: Informed Sampling

The simulation illustrates how informed sampling speeds up the convergence rate and focus the sampling on the subset of states that may improve the initially found solution. One simulation is performed where each subfigure in Figure 8.16 shows the state after 50 more nodes added to the tree. The algorithm was tuned using the values given in the S.2 column of Table 8.2. Note that β is set very high to only draw samples inside the ellipse if an initial path from start to goal is found. The start and goal pose was set to $x_0 = (10\,\text{m}, 2\,\text{m}, 135°)$ and $x_{goal} = (14\,\text{m}, 4\,\text{m}, 135°)$.

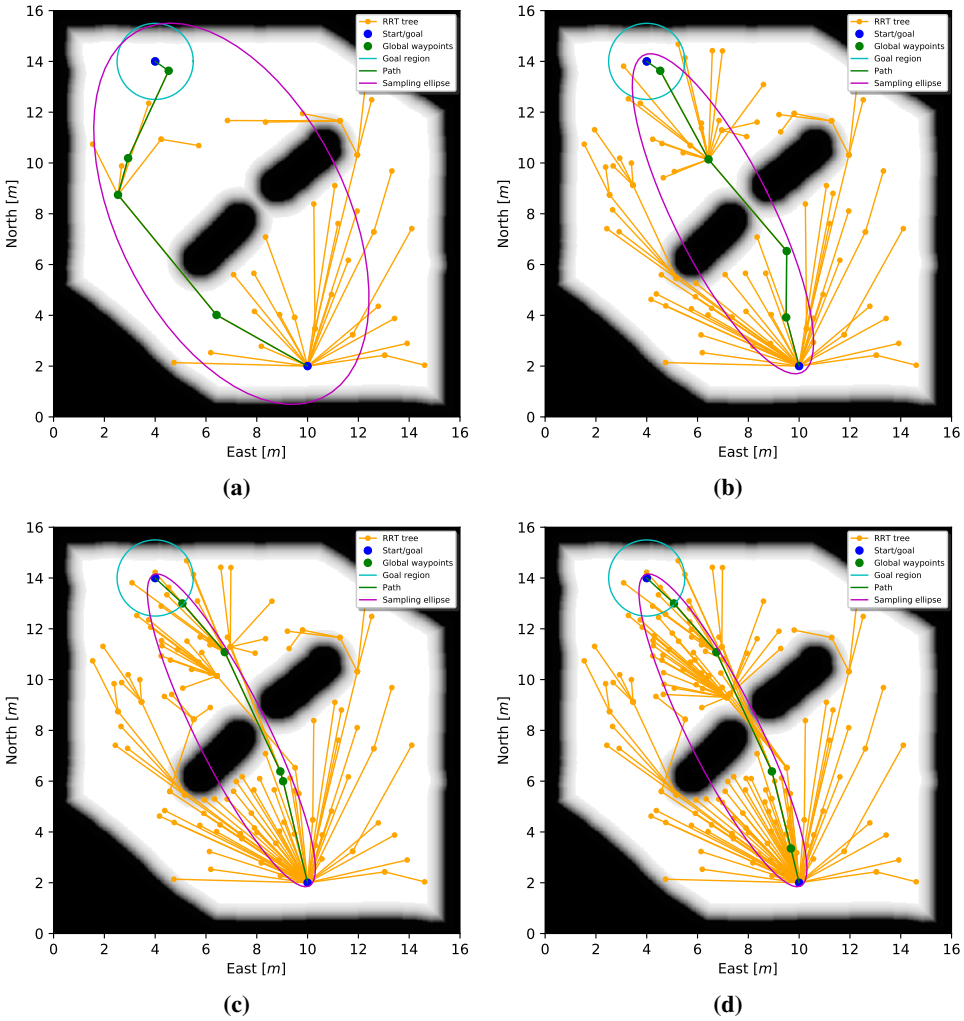


Figure 8.16: The results for the local planner, Simulation 2: Informed Sampling.

8.4.3 Simulation 3: Blocking Branches by Dynamic Obstacles

The simulation illustrates how dynamic obstacles do blocking of branches, and how rewiring of the tree performs obstacle avoidance and finds new paths. One simulation is performed where each subfigure in Figure 8.17 shows the state after 150 more nodes are generated. The algorithm was tuned using the values given in the S.3 column of Table 8.2. The start and goal pose was set to $x_0 = (2\ \mathrm{m}, 8\ \mathrm{m}, 0°)$ and $x_{goal} = (14\ \mathrm{m}, 8\ \mathrm{m}, 0°)$. One obstacle with an obstacle region $d_{min} = 2\ \mathrm{m}$, starts at $(14\ \mathrm{m}, 8\ \mathrm{m})$ and moves from north to south. The size of the grid cell for the spatial indexing is greater than $d_{min} \times d_{min}$ to contain the obstacle region.

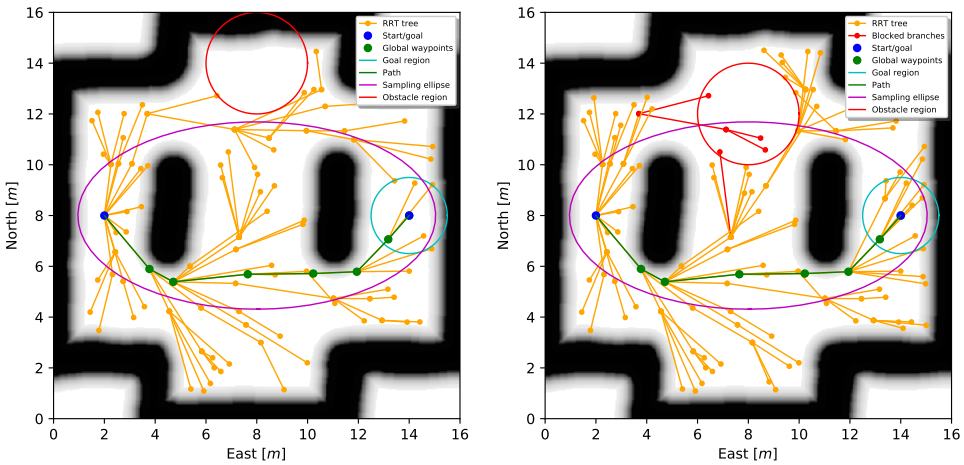


Figure 8.17: The results for the local planner, Simulation 3: Blocking Branches by Dynamic Obstacles.

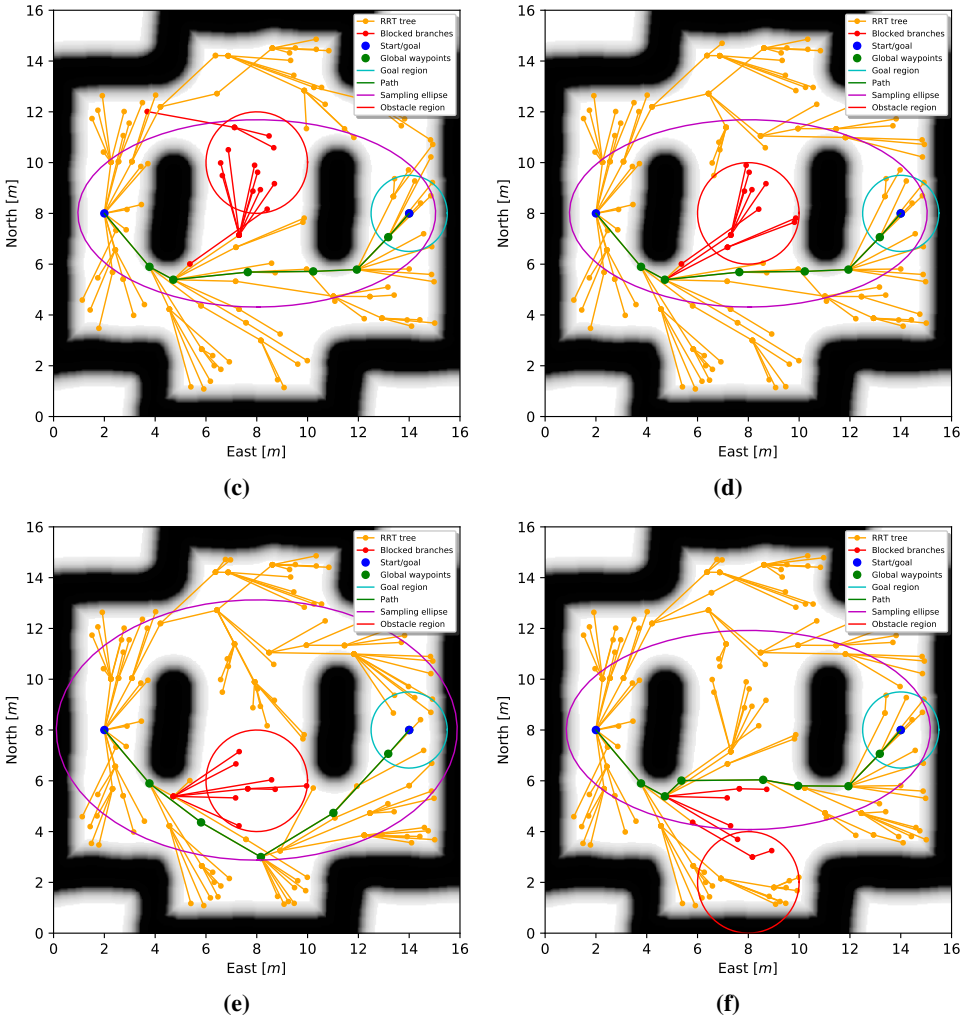


Figure 8.17: The results for the local planner, Simulation 3: Blocking Branches by Dynamic Obstacles.

8.4.4 Simulation 4: Rewiring and Planning "On the Fly"

In this simulation, we update the tree root using the obtained path from Algorithm 5, such that the root moves towards the goal. The simulation illustrates how the tree is gradually rewired "on the fly" to tackle a dynamic obstacle. One simulation is performed where each subfigure in Figure 8.17 shows the state after 150 more nodes are generated. The algorithm was tuned using the values given in the S.4 column of Table 8.2. The start and goal pose was set to $x_0 = (2\,\mathrm{m}, 12\,\mathrm{m}, 0°)$ and $x_{goal} =$

$(14\ \mathrm{m}, 4.5\ \mathrm{m}, 0°)$. One obstacle with an obstacle region $d_{min} = 2\ \mathrm{m}$, starting at the same location as the goal, is set to move from east to west. The size of the grid cell for the spatial indexing is greater than $d_{min} \times d_{min}$ to contain the obstacle region.

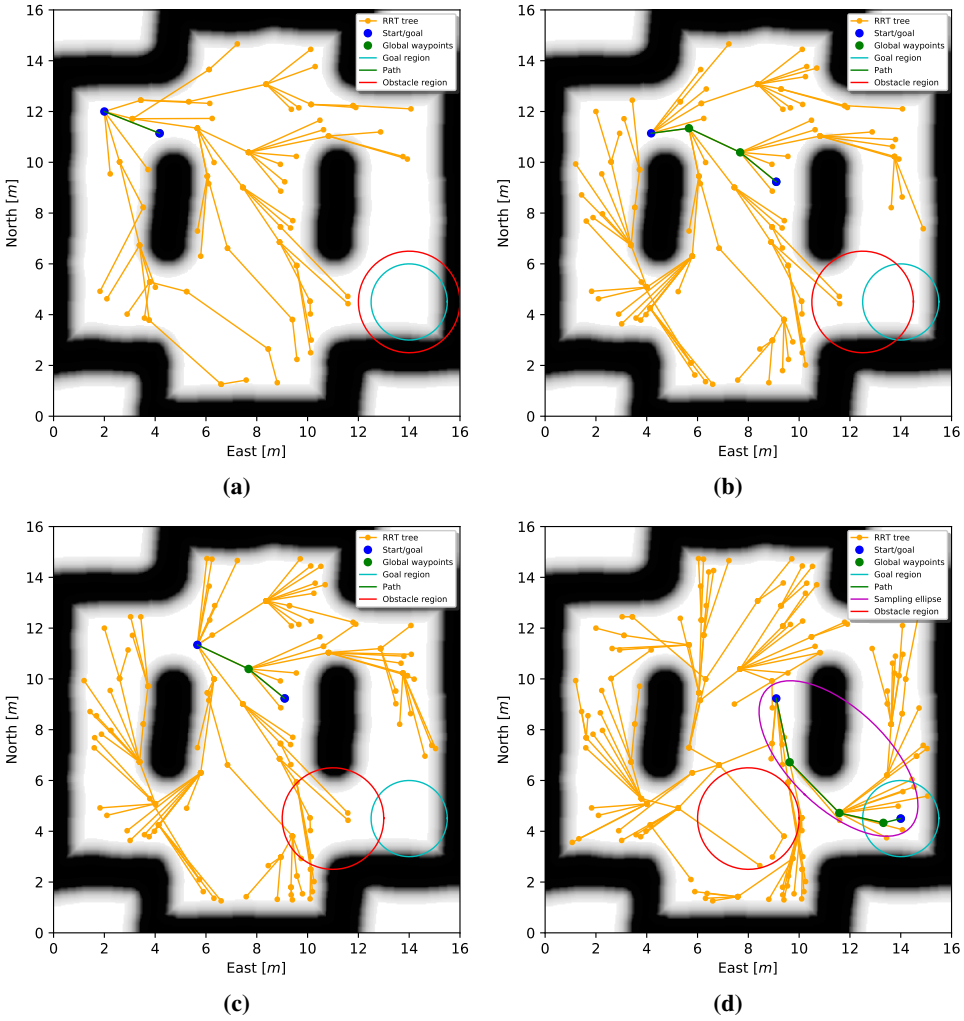


Figure 8.18: The results for the local planner, Simulation 4: Rewiring and Planning "On the Fly".

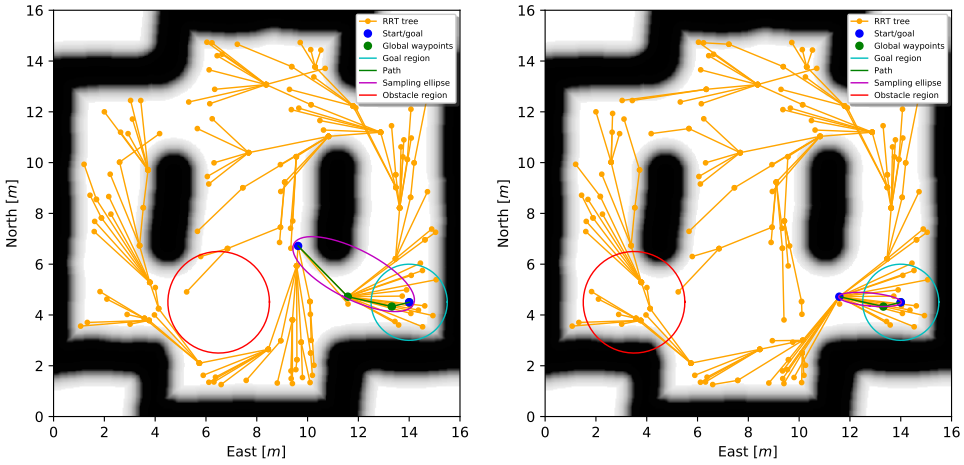


Figure 8.18: The results for the local planner, Simulation 4: Rewiring and Planning "On the Fly".

8.4.5 Results

Four simulations are performed and illustrate different aspects of the local planner. The simulations are performed in different environments to show diversity. All simulations are performed following the assumptions given in Section 3.5, which is first and foremost a maximum restriction of $\pm$ 90° turn between previous, current, and next node. All simulations were tuned to their purpose. We start by discussing the results from Simulation 1, and continue with 2, 3, and 4.

Simulation 1 shows how each respective cost function affects tree growth and expansion. As expected, using the Euclidean distance as cost function results in the shortest path. The path itself wraps around the obstacle in the middle to obtain the shortest path in Figure 8.15a. The branches of the tree extend as far as they can in each direction to find the shortest distance from the root to every other state. Note that no nodes are sampled behind the root because of the restrictions on angle and maneuvering actions. The minimum repulsion and curvature cost functions in Figures 8.15b and 8.15c produces significantly different paths and tree topologies, compared to Euclidean distance cost function. Figure 8.15b shows how the path is chosen to avoid the obstacles at any cost. Bear in mind that only leaf nodes are generated close to obstacles since they have a high cost compared to the ones far away. The minimum curvature cost in Figure 8.15c produces a smooth path with the waypoints distributed at regular distances. By using a combination of the three cost functions, one can take leverage of their respective properties to achieve a reasonable

path for a low-speed vessel such as ReVolt, given in Proposition 5.1.

For the local planner to be able to work in real-time, it is an absolute necessity that the convergence rate is as fast as possible. Simulation 2 shows how informed sampling increases the convergence rate towards the optimal path in terms of path elongation. Figure 8.16a depicts the initial path found after 50 nodes are generated. Already we know, as long as there are no changes in the environment, that the optimal path will lie within the ellipse. Thus we can exclude going around the obstacles in the center of the map on the north side. In Figure 8.16b, the local planner finds the narrow passage between the obstacles and further decreases the size of the ellipse. A sampling of the informed subset increases the density of samples drawn around the optimal solution, which contributes to faster convergence towards the optimal solution. Figures 8.16c and 8.16d further illustrates how the density increases inside the ellipse, while no uniform samples outside of it are done.

In Simulation 3, a dynamic obstacle moves from north to south. Figure 8.17a shows the initial path after 150 nodes are generated. The obstacle starts moving southwards in Figure 8.17b, and the first nodes are blocked. Note how rewiring is done such that nodes north of the obstacle are connected to other parts of the tree. In Figures 8.17c and 8.17d the obstacle moves further south, and nodes that were obtained inside the obstacle region are gradually and continuously connected to the tree again. The initial path is kept until the state portrayed in Figure 8.17e. Here the obstacle has moved such that the initial path found is not valid anymore. The tree is rewired to go around the obstacle on the south side. However, as the obstacle moves further south in Figure 8.17f, rewiring finds a path similar to the one found initially. Remember that the sampling is divided between uniform and inside the ellipse to cope with unforeseen changes. This allows the local planner to find reasonable paths outside the ellipse as well.

In Simulation 4, all aspects of the global planner are tested. The simulation starts with an obstacle placed on top of the goal region. Since no valid path is found in Figures 8.18a to 8.18c, the local planner plans a K-step path in accordance with Algorithm 5. In each step, we update the root to be located at the next node found in our path and rewires continuously. In Figures 8.18b and 8.18c, one can see how the nodes behind the root get connected to new parts of the tree, which does not violate the imposed constraints. As the obstacle moves westwards and away from the goal region, a path to the goal is discovered in Figure 8.18d. The root now follows the path up to the goal region. Note that the sampling ellipse is updated in Figures 8.18e and 8.18f as the root shifts towards the goal. Simulation 4 illustrates clearly the advantage of the hybrid path planning approach, where the path is updated at each step according to obtained information.

Note that Simulation 3 and 4 is performed with no restrictions on time, as indicated in Algorithm 3. This makes us able to rewire the whole tree at each iteration.

Note that if a restriction is made, the rewire procedure starting from the root ensures that the most critical subspace in front is rewired first. With time, other parts of the tree further away will gradually be rewired. This is clearly seen in Figures 8.18d to 8.18f. During simulations, a solution was always found if it existed, which coincides with RRT's property of being probabilistically complete.

8.5 A Complete Simulation

A simulation of the complete GNMC system is performed to illustrate how the subsystems can be integrated in a seamless manner to achieve autonomous guidance and control. The results are shown in Figures 8.19 to 8.21. The global planner solve the path planning problem from $\mathbf{WP}_0 = (152\ \mathrm{m}, 1128\ \mathrm{m})$ to $\mathbf{WP}_{goal} = (179\mathrm{m}, 1128\mathrm{m})$, with clearance constraint $d_{min} = 2\ \mathrm{m}$ and angle threshold $\psi_{thres} = 10^{\circ}$. The global path together with the Voronoi roadmap is shown in Figure 8.19. The global waypoints are are fed in as intermediate waypoints for the local planner. If the vessel is inside the the goal region of the corresponding waypoint, a new global waypoint is set up to be tracked down by the local planner. The local planner are tuned using the values given in the S.5 column of Table 8.2. Two dynamic obstacles starting at $(177\ \mathrm{m}, 1149\ \mathrm{m})$ and $(165\ \mathrm{m}, 1129\ \mathrm{m})$, with an obstacle region $d_{min} = 2\ \mathrm{m}$ are present. Both obstacles move westwards at a slow pace. The limited time set for tree rewiring and expansion is set to $0.01\ \mathrm{s}$. Each local path is constructed in a stepwise manner where each path segment is generated when the vessel is sufficiently close to the following waypoint. Figure 8.21 shows 6 different states of the local planner during the simulation.

The guidance system uses the optimization approach with the same tuning parameters as in Table 8.1. The control system uses the same tuning parameters and configurations, as given in Section 8.2. The desired path constructed by the guidance system, together with the vessel position in the horizontal plane, is depicted in Figure 8.20. Plots for the position, speed, and force in each DOF for the simulation are given in Appendix E.

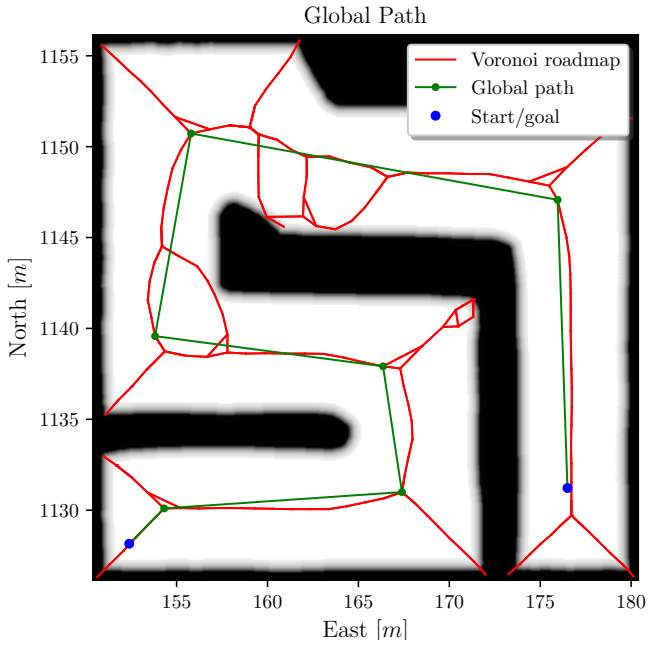


Figure 8.19: The results of the complete simulation for the global planner.

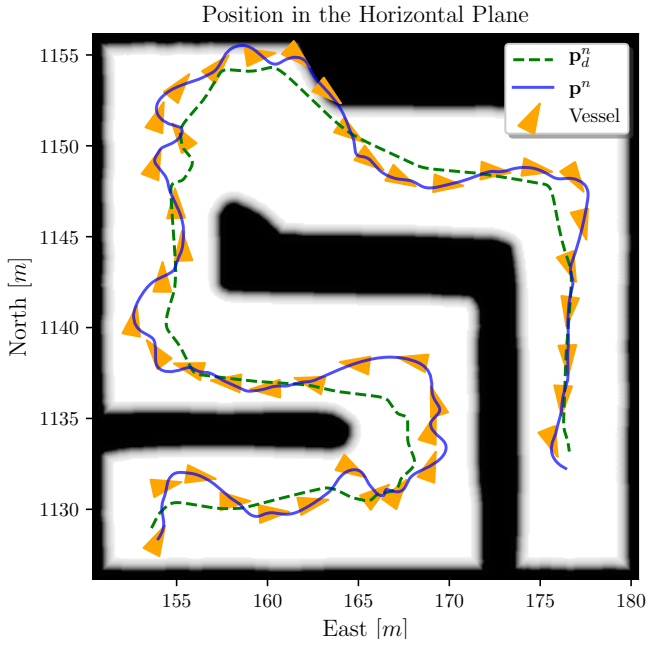


Figure 8.20: The results of the complete simulation for the guidance and control system.

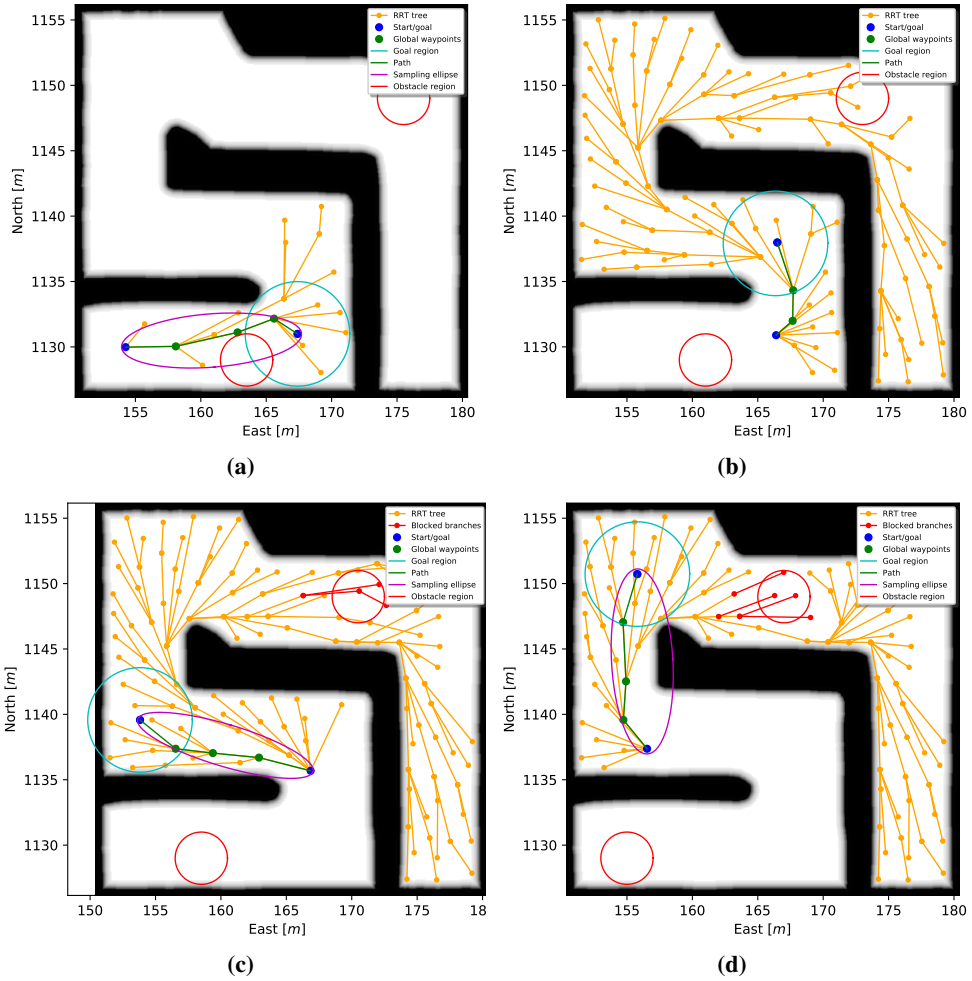
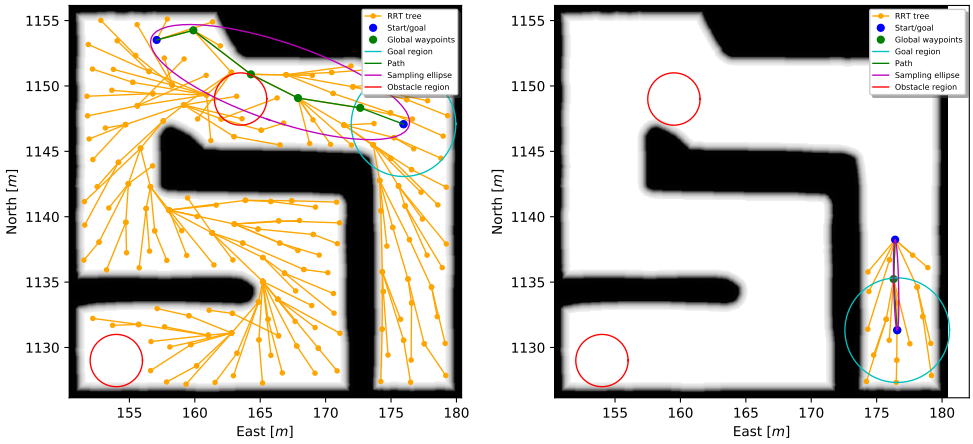


Figure 8.21: The results of the complete simulation for the local planner.



(e) **(f)**

Figure 8.21: The results of the complete simulation for the local planner.

8.5.1 Results

Figure 8.19 shows that the global planner plans a safe path consisting of few waypoints through the configuration space. It is clear that the global planner only takes into consideration the static obstacles defined by the black areas in the configuration space. Figure 8.21 shows how the global waypoints are used as intermediate waypoints to be tracked down by the local planner. One can see how the local planner continuously rewires and expands its tree to avoid dynamic obstacles. By comparing Figures 8.20 and 8.21e, one can see how the local planner replans its route to avoid the approaching obstacle. This results in the detour on the north side of it. Also note how the tree is rewired back towards where the vessel came from in Figure 8.21e, covering the whole configuration space. Intuitively this can be seen as overkill, and it would be smarter to focus the computational effort in front and close by the vessel.

In Figure 8.20, the final desired path constructed by the guidance system is portrayed. A flaw of the guidance system was revealed; it produces wiggling path segments for waypoints placed very close to one another. This leads to the control system having a hard time tracking the desired output from the guidance system. One can see that around path segments with rapidly changing heading, the control system struggles.

Chapter 9

Experiment

Due to limited time and the impact of COVID-19, it looked like the thesis had to be restricted to only simulations. Fortunately, we were able to perform a last-minute experiment of the guidance and control system on the real ReVolt. This chapter represents the experimental setup, the results obtained, and experimental problems experienced. Section 9.2 mentions some important hardware related problems experienced with ReVolt that should be resolved for ReVolt to be correctly operating in the future. The parameter values used in the control model of ReVolt is given in Appendix A. The goal of the sea trial was to validate the performance of the guidance and control system.

9.1 Experimental Setup

The experiment was conducted in Dorabassenget in Trondheim, June 2, 2020. Testing of software and addressing hardware related problems on ReVolt were done at NTNU Gløshaugen before it was shipped down to Skippergata, where ReVolt was ejected into the water. As an escort boat, NTNUI Dykkergruppa's Fjøset II was rented. The experiment was done together with Knut Turøy and Simen Sem Øvereng, who also performed experiments on ReVolt the same day. Two pictures from the experiment are shown in Figure 9.1.

A path consisting of 14 waypoints forming an 8-shaped path, given in Appendix D, was created. The optimization approach was used as the path generator. The same tuning parameters as for the S-shaped simulation in Section 8.1.2 was used, except for the corridor width ζ set to $4\ \mathrm{m}$.

For the control system, the same tuning parameters used in the simulations, given in Section 8.2, was used. In the experiment, the vessel was put to rest in a dynamic position (zero speed) at the first waypoint with heading in the direction of the second waypoint. Then, the vessel was commanded online to move along the path with a desired speed $u_d = 0.25\ \mathrm{m/s}$. When the vessel was close enough to the next waypoint, a new path segment was generated by the path generator. Figure 9.2 shows the computation graph of the control system running on the onboard computer.

(a) **(b)**

Figure 9.1: Figure **(a)** shows me controlling the ReVolt remotely from the escort boat with ReVolt in the background. Figure **(b)** shows Knut Turøy, Simen Sem Øvereng, and his biceps observing ReVolt's behavior during the experiment.

The sea trial was conducted under calm weather conditions with practically no waves present. However, it was evident that some current and light breeze was affecting the vessel. The environmental load was documented by putting the vessel to rest, with the bow facing the wind. A rosbag was recorded to see how the vessel drifted away from the initial point. See Figure 9.3. One can see that the vessel starts drifting away towards the southwest. Notice how the Munk moment turns the vessel such that the port side is facing towards the direction of the average environmental load.

[1] `http://wiki.ros.org/rqt_graph`

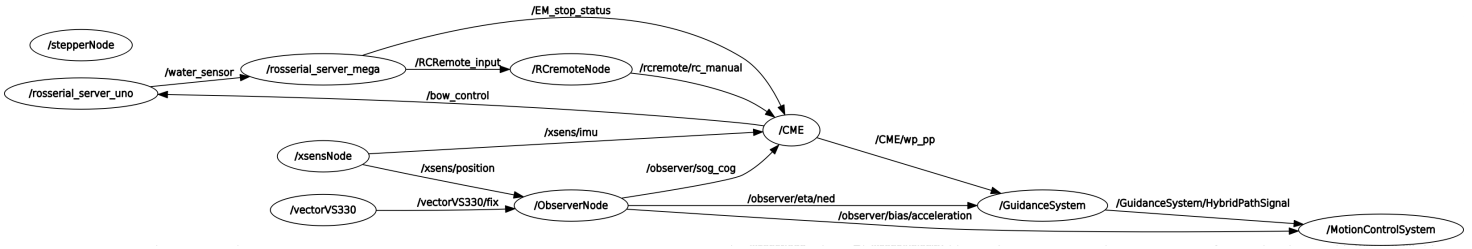
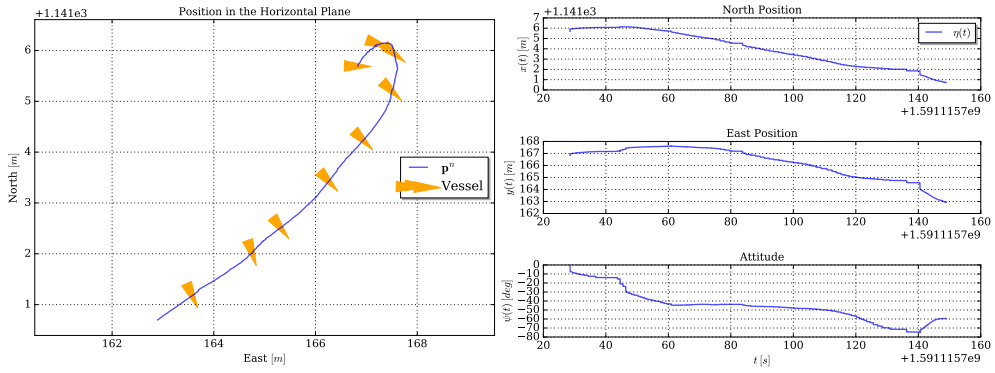


Figure 9.2: The figure shows the computation graph of the control system running on the onboard computer during the experiment. The vertices in the graph represent ROS nodes, while the edges between them are ROS topics which the nodes communicate through. The graph is generated by the GUI plugin `rqt_graph` package[1]. The CME node is the "Control Mode Executor", which is the superior node deciding in which control mode the vessel should be. The Xsens and vector330 node is the IMU and vector, respectively, providing the necessary measurements to the observer.



(a) **(b)**

Figure 9.3: Environmental forces acting on the vessel under the experiment. Figure **(a)** shows position in horizontal plane. Figure **(b)** shows position versus time for each DOF.

9.2 Problems

Physical testing always presents unforeseen issues related to hardware and purely practical problems. Initially, the test was set to May 29, but problems related to the battery and stern thrusters were discovered. Thus, the experiment had to be postponed. During the winter, changes in the location of specific equipment were done. This led to problems with getting heading signals from the VS330 vector. It was found that the settings on the device were outdated and had to be changed physically.

Also, several problems were discovered related to the thrusters. A safety sensor seemed to be damaged during the shipping down to Skippergata. It was blocking all signals going to the bow thruster and thus had to be disabled. It was found that constraining the bow thruster to $90°$ in the code did not match $90°$ physically. A workaround was to constrain the bow thruster to $120°$. A malfunction in the bow thruster made it incapable of applying any thrust below 50% of maximum thrust. This will affect the results, especially related to heading. Further, hysteresis related to rotating the stern thrusters were discovered during the initial test day, May 29. Some duct tape, together with WD-40 spray, lubricated the thrusters and resolved the issue during the test day.

Lastly, it was discovered that ReVolt took in some water during the test day. All these problems should be resolved for ReVolt to be correctly operating in the future.

9.3 Results

Three robag recordings were done when performing the experiment. All three delivered similar result. The results shown in Figures 9.4 to 9.9 are from the second recording. The vessel starts in approximately $\mathbf{WP}_0 = (179\ \mathrm{m}, 1129\ \mathrm{m})$ and performs an 8-shaped maneuver ending up in $\mathbf{WP}_{13} = (184\ \mathrm{m}, 1129\ \mathrm{m})$.

From Figure 9.4, one can see that the guidance system produces stepwise a smooth 8-shaped path to be followed. From Figures 9.6 and 9.7, one can see that ReVolt can track the desired path, but struggles to satisfy the speed assignment. On the straight lines between the corners, it traces the path precisely. However, one can see that is it struggles to trace the path accurately in the corners. One reason for this is the environmental load acting on the vessel. Another one is the physical problems found with the thrusters, explained in Section 9.2.

However, notice how all signals are lost and freezes after approximately 100 s in Figures 9.7 to 9.9. Also, see the jump in the estimated north position of about 5 m in Figure 9.7 around 160 s. One can see the oscillating effect of this on the desired and actual control forces applied to the ReVolt around 160 s in Figure 9.9. This indicates that the experiment was occasionally prone to poor signals from the measurement system, which affect the results. Notice the difference in continuity in signals between the simulations and the experiment. The simulation has continuous smooth signals, while the signals in the experiment have similarities with zero-order-hold sampled signals, as expected.

The experiment indicate that the control design does not need 100% numerically correct values of the hydrodynamic parameters to work. This illustrates the robustness of the control system.

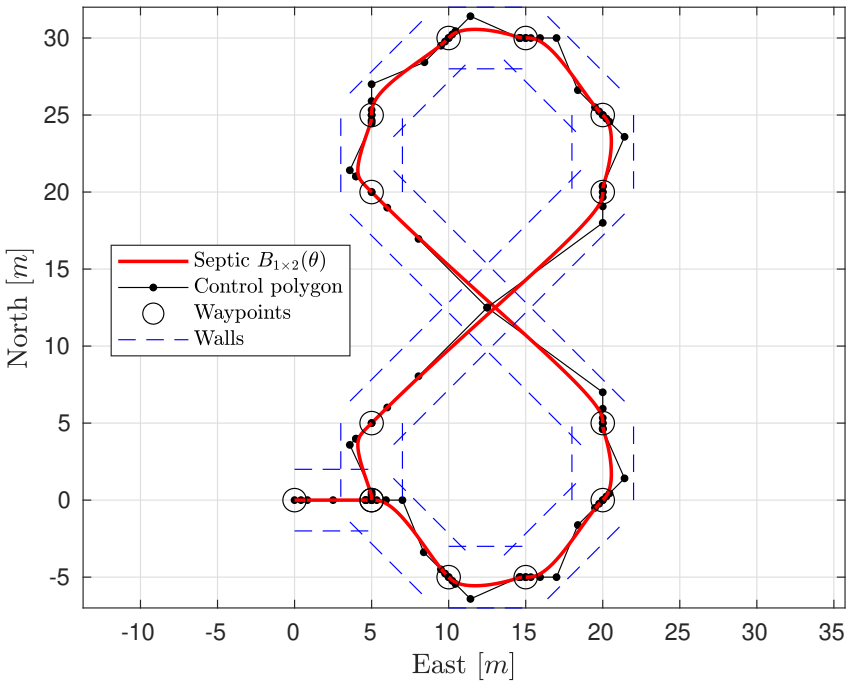


Figure 9.4: A xy-plot of desired path constructed. The Bézier curve is drawn in red on top of its control polygon.

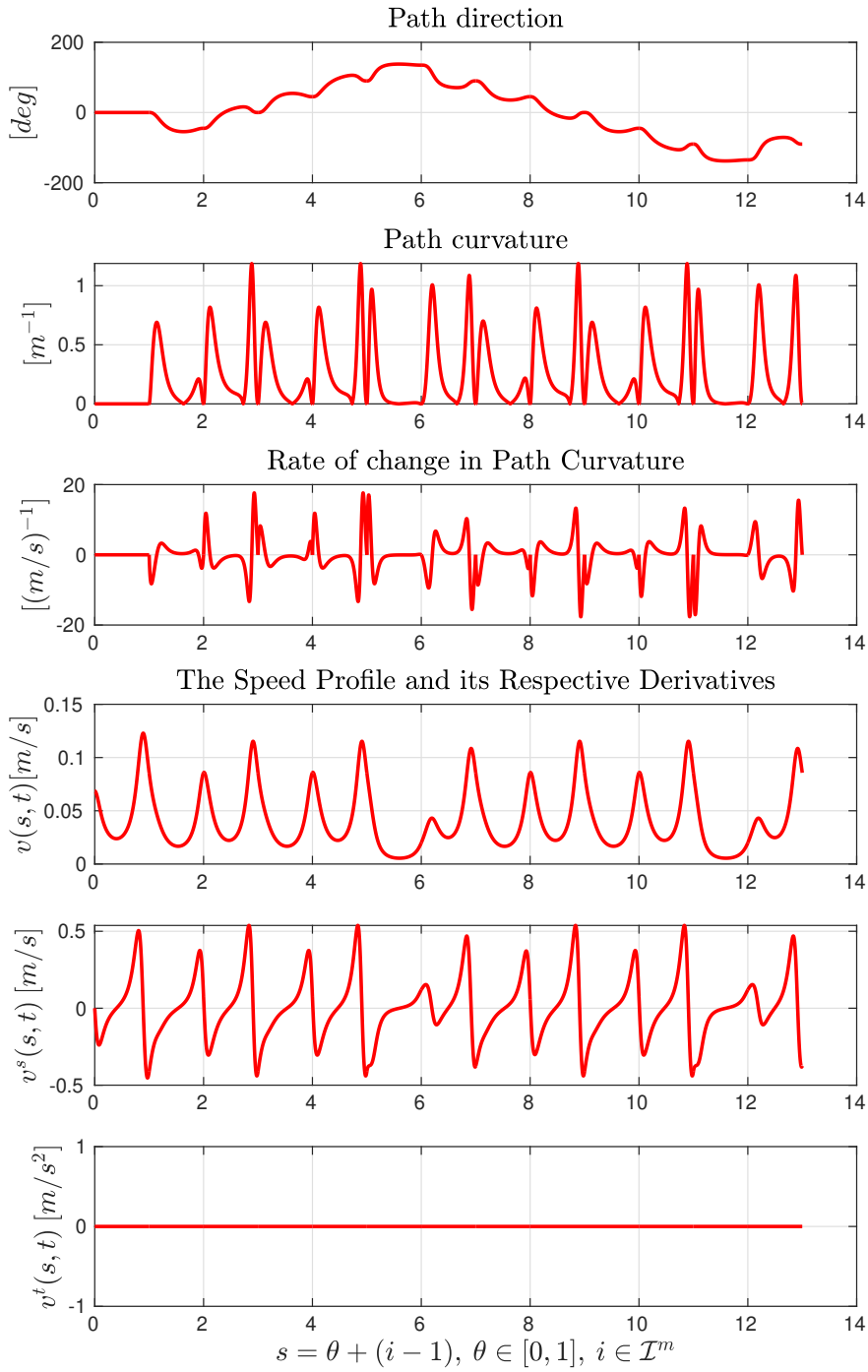


Figure 9.5: The direction, curvature, rate of change in curvature, and the speed profile and its respective derivatives for the desired path in Figure 9.4.

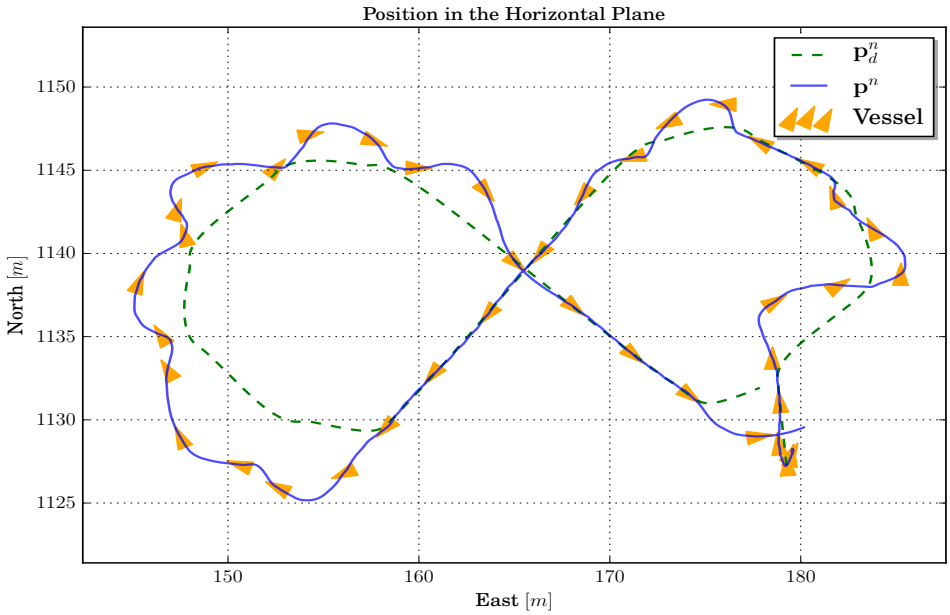


Figure 9.6: Position in horizontal plane.

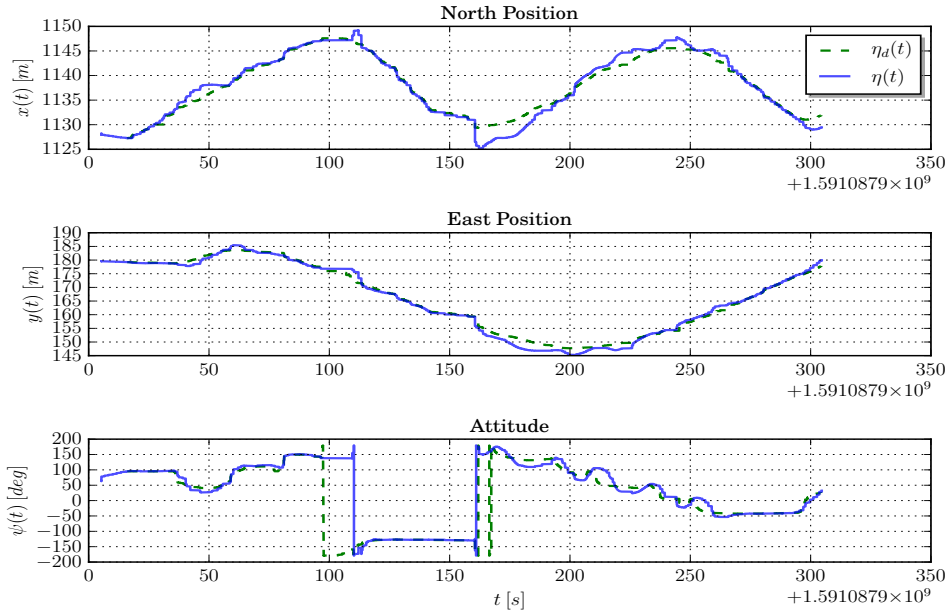


Figure 9.7: Position versus time for each DOF.

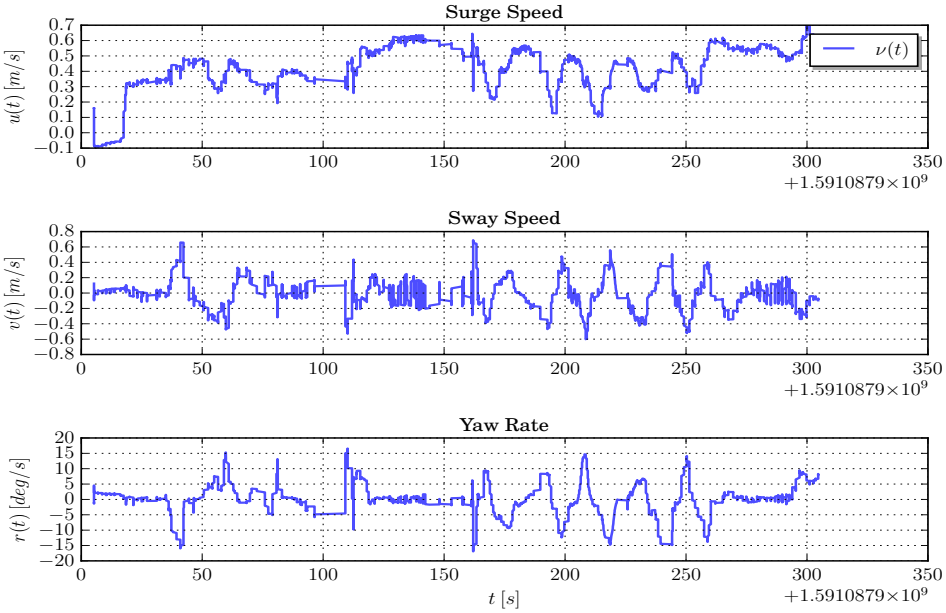


Figure 9.8: Speed versus time for each DOF.

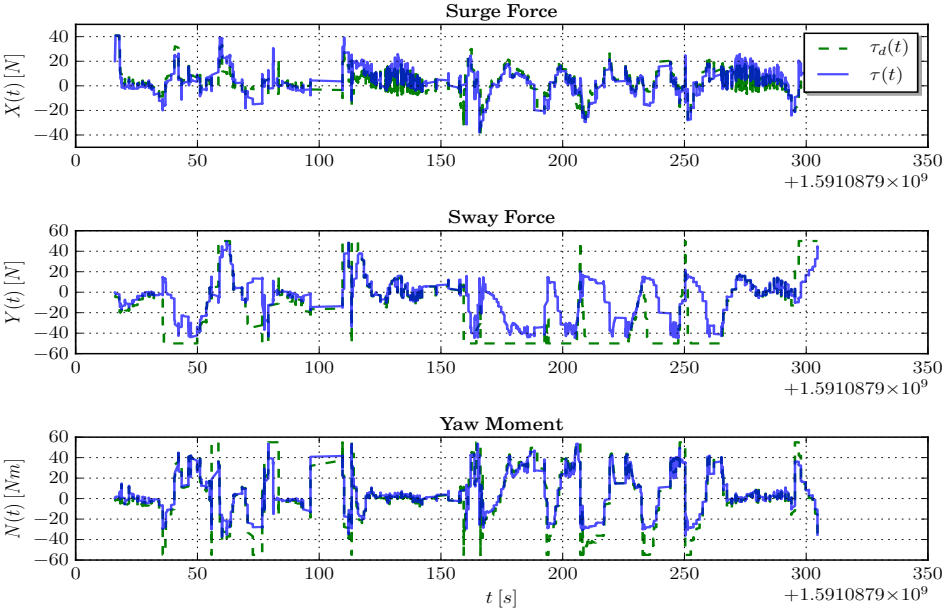


Figure 9.9: Forces versus time for each DOF.

Chapter 10

A Critical Assessment

A diverse set of simulations, together with an experiment, are done to verify the performance of each developed subsystem as well as illustrate how they can be integrated seamlessly to achieve autonomous guidance and control. The navigation system can produce safe and efficient paths through waypoints for simple scenarios. It operates on OGMs, which can be obtained from available sensors on the ReVolt vessel and written scripts. The global planner is computationally efficient due to the low computational effort of generating the Voronoi roadmap and searching through it with the optimal A* algorithm. It should be mentioned that it was assumed that the size of static obstacles was greater than the clearance threshold, introduced in Section 5.2.2. Consequently, static obstacles smaller than the clearance threshold are not detected in the current implementation. This is a defect that should be corrected in the global planner.

A drawback of the current implementation is that both the global and the local path planner operates on the same resolution of the operation area. The global planner could undoubtedly operate on a lower resolution to achieve satisfactory performance.

By reducing the number of waypoints, the number of maneuvering actions is reduced, in accordance with Proposition 5.1. Having a look at the complete simulation, the local planner tends to produce paths defined by many waypoints, resulting in unpractical paths. Better tuning of the local planner would likely reduce the amount. Especially the minimum and maximum distance between the nodes together with the grid dimension for spatial indexing would have a significant impact. Also, penalizing maneuvering actions in the objective function would help. Another solution to reduce the number of waypoints could be to prune waypoints from the local planner similar to the global planner, described in Section 5.2.3.

As for now, the local planner rewires and expands its tree in the whole configuration space. For a single query problem, this is an overkill and leads to unnecessary use of computation. Intuitively, the local planner could be limited to only search the closest surroundings in the search for the next intermediate waypoint received from the global planner.

An important question is if the local planner is fast enough to tackle real-time

planning. The grid-based spatial indexing, together with an upper threshold for the number of nodes nearby, was implemented to put a roof on the computational effort in each iteration (see Section 5.1.4). Profiling of the current implementation was done to identify the bottlenecks in the algorithm, which was first and foremost identifying if grid cells were occupied or not. This is a clear indication of how the resolution of the OGM affects the efficiency of the current implementation. It should also be mentioned that the implementation was done in Python, which is an interpreted language. A considerable boost in speed would, of course, be to implement in a compiled language such as C++.

From the simulation and experiment performed in Section 8.1, it is demonstrated that the Bézier curve and optimization can be combined to produce a reasonable path for a low-speed vessel through stepwise path generation. The suggested algorithms deliver a $\mathcal{C}^3$ continuous path, such that the required outputs from Section 3.3 can be produced. Thus, it is reasonable to say that the algorithms give adequate results, based on the assumptions and delimitations given in Section 3.5.

However, from the complete simulation, it was discovered that the path generator delivers wiggling paths with high curvature for waypoints placed close to one another. This raises the question if the path generator should be tuned for different segment lengths to produce a reasonable path. This is a drawback of the current design and implementation. Referring to the objectives in Section 1.2, it was desired to impose constraints on maximum curvature to respect the dynamical constraints of the vessel. As for now, the curvature control problem is unsolved, resulting in the speed of the vessel has to be adjusted to ensure feasibility. As demonstrated in Section 4.2.5, constraining the curvature led to a highly nonlinear constraint and a challenging optimization problem to solve efficiently.

Having a closer look at the result in Section 8.1, one could ask the question if the pragmatic approach is good enough for our purpose, and the advantage of performing an optimized search is worth it. Nevertheless, one should bear in mind that the pragmatic approach is tuned for the specific scenario.

The maneuvering controller delivers satisfying results when it comes to tracking the desired path. However, it struggles to satisfy the speed assignment in both simulations and experiment. Heavy oscillations for both simulations and experiment characterize the desired surge force. The behavior indicates aggressive tuning parameters in surge and the need for a *dead-zone*, causing the control to not respond to small deviations. As mentioned, this could be caused by suboptimal tuning as well as inadequate state estimates obtained from the EKF observer. However, one could argue that the path generator may occasionally deliver a path violating the dynamics of the vessel, which can be hard to track. Hence, it is difficult to blame the maneuvering controller. The pseudo-inverse thrust allocation algorithm on ReVolt delivers pleasant results in surge and yaw, but significant deviations were seen between the

commanded and actual force applied in sway. By constraining the bow thruster to 90°, better results were obtained. However, one can still see from the plots in Section 8.2 that the thrust allocation struggles. This may be caused by poor tuning of the weight matrix penalizing the thrust efforts of each respective force component in the thrust optimization problem in Equation (6.52a). DNV GL has not given access to tune the weight matrix in its implementation.

The simulations (and partially the experiment) have shown how the subsystems investigated can be integrated seamlessly to achieve autonomous guidance and control. Moreover, the stepwise approach has been successfully interpreted. However, assuming only one waypoint ahead is known may be an oversimplification of the problem, which leads to sub-optimal planning. As for now, the complete system has only been tested with dynamic obstacles for simple scenarios. For the system to be applicable for real-life situations, it must be able to handle more complex scenarios compliant with COLREG.

Finally, it should be noted that the proposed GNMC system requires tuning of many parameters to perform well, which can be challenging.

Chapter 11

Conclusion

This thesis has proposed an intelligent guidance concept for an ASV, moving from initial to target point. The problem was faced in a stepwise manner to facilitate real-time execution and the ability to replan in an online. For convenience, we proposed to divide the complete closed-loop system into four submodules: the guidance, navigation, measurement, and control system.

The optimal A* algorithm working on a Voronoi roadmap, together with a real-time variant of RRT combining the useful aspects of different variants adapted to our use, constituted the navigation system. The global planner, mostly based on existing methods, generated safe paths efficiently, due to the low computational cost of generating the Voronoi roadmap and the optimal search of A*. With its probabilistic completeness, the local planner always found a feasible path if one existed. It showed how RRT could be used to avoid dynamical obstacles and gradually replan in a stepwise manner. However, questions were raised if the current solution is computationally efficient enough to tackle real-life situations. Also, it tended to produce paths consisting of many waypoints, leading to irrational planning for a stepwise problem assuming only one waypoint ahead is known.

A new way to generate a stepwise, smooth, and continuous path in the horizontal plane using the Bézier curve and quadratic optimization was developed. Also, a pragmatic approach was suggested. It was shown how the path generators, together with a dynamic assignment (which makes up the guidance system), was able to produce the necessary signals for a maneuvering controller. Nonlinear adaptive backstepping was used to design a high-level maneuvering control law. Due to problems with the already implemented thrust allocation algorithm provided by DNV GL, it was added to the scope of work.

A considerable amount of simulations were performed to verify performance for each designed subsystem. The simulations provided a sound basis for experimental testing. Due to limited time and the impact of COVID-19, only the guidance and control systems were tested experimentally on the real ReVolt.

11.1 Recommendations for Further Work

As for any project, there is always a constraint on time. There are several features of the proposed GNMC system that can be approved and made more sophisticated. We start with the vessel's behavior on a practical level. For convenience, we assumed when approaching the problem that the vessel knew at most one waypoint ahead in time. However, as seen from the complete simulation performed in Section 8.5, we often know more than only one. Especially the path generator could utilize this information to give a more elegant path behavior. For instance, the desired heading at the next waypoint could be the mean of the current and next segment heading, which will result in a smoother path. Also, one could argue that the vessel does not need to go through the waypoints, but rather within a defined circle of acceptance. See, e.g., Fossen et al. (2003). This would give more flexibility for the path generator and contribute to more elegant paths.

In the current implementation of the local planner, informed sampling, together with grid spatial indexing, was introduced to speed up the convergence rate and put a roof on computational effort in each iteration. For a real-time planner, it will always be in its best interest to improve more. Jaillet et al. (2008) presented the method Transition-based RRT, which combines the strengths of RRT with the efficiency of stochastic optimization methods that use transition tests to accept or to reject a new potential state. This could be a promising extension of the current implementation that can be implemented, as the navigation system already operates on cost maps.

Further work would undoubtedly investigate how to make the local planner comply with COLREG. Simulation 3 of the local planner illustrates a great example where COLREG would block the whole area in front of the obstacle and force the vessel to move behind it. Chiang and Tapia (2018) has already started research on an RRT-based COLREG-compliant motion planner.

As stated in the objectives and scope of work, it was desired to constrain the maximum curvature allowed for the path generator to comply with the vessel's dynamics. As shown in Section 4.2.5, this results in a highly nonlinear constraint and a hard problem to solve. To the best of the author's knowledge, it remains open to solve the curvature control problem when the arc length of the curve is to be minimized. However, several workarounds have been proposed in the literature and could be investigated in further work. Another interesting point is how the replanning technique, introduced in Section 4.2.3, could be exploited not just for replanning, but to produce smoother paths with low curvature.

For now, the vessel is only intended to operate on calm sea states. The system should be made more robust by taking into account environmental loads as waves, wind, and current. As for all tunable systems, more time and effort could be invested to achieve better performance through tuning. The same tuning parameters were used

in simulation and experiment. Even with realistic simulations, the tuning parameters will never perfectly fit the real system. Some proper tuning of the parameters on the real system will probably improve the performance. Also, for the maneuvering controller, one could add an integral state to better compensate for the constant bias.

At last, it would be favorable to perform more complex simulations and sea trials to validate the performance better, especially for the navigation system.

Bibliography

Abrahamsen, B., 2019. Fault Tolerant Dynamic Positioning for the Autonomous Test Platform ReVolt. Master's thesis. Norwegian University of Science and Technology. Trondheim.

Alfheim, H., Muggerud, K., 2017. Development of a Dynamic Positioning System for the ReVolt Model Ship. Master's thesis. Norwegian University of Science and Technology. Trondheim.

Aurenhammer, F., 1991. Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure. ACM Comput. Surv. 23, 345–405. URL: https://doi.org/10.1145/116873.116880, doi:10.1145/116873.116880. place: New York, NY, USA Publisher: Association for Computing Machinery.

Barsky, B.A., DeRose, A.D., 1984. Geometric Continuity of Parametric Curves. Technical Report UCB/CSD-84-205. EECS Department, University of California, Berkeley. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5752.html.

Binder, B., 2017. Spatio-temporal prioritized planning. Master's thesis. Technische Universität Wien. Wien, Austria.

Bruce, J., Veloso, M., 2002. Real-Time Randomized Path Planning for Robot Navigation. doi:10.1007/978-3-540-45135-8_23.

Candeloro, M., Lekkas, A.M., Sørensen, A.J., Fossen, T.I., 2013. Continuous Curvature Path Planning using Voronoi diagrams and Fermat's spirals. IFAC Proceedings Volumes 46, 132 – 137. URL: http://www.sciencedirect.com/science/article/pii/S147466701646146X, doi:https://doi.org/10.3182/20130918-4-JP-3022.00064.

Casselman, B., 2008. From Bézier to Bernstein. URL: http://www.ams.org/publicoutreach/feature-column/fcarc-bezier.

Cheng, P., Shen, Z., Lavalle, S., 2001. RRT-based trajectory design for autonomous automobiles and spacecraft. Archives of Control Sciences 11.

Chiang, H.L., Tapia, L., 2018. COLREG-RRT: An RRT-Based COLREGS-Compliant Motion Planner for Surface Vehicle Navigation. IEEE Robotics and Automation Letters 3, 2024–2031. doi:10.1109/LRA.2018.2801881.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. Introduction to Algorithms, Third Edition. 3rd ed., The MIT Press.

Dubins, L.E., 1957. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. American Journal of Mathematics 79, 497–516. URL: http://www.jstor.org/stable/2372560. publisher: Johns Hopkins University Press.

Elfes, A., 1989. Using Occupancy Grids for Mobile Robot Perception and Navigation. Computer 22, 46 – 57. doi:10.1109/2.30720.

Elia Nadira, S., Omar, R., Hailma, C.K.N., 2016. Potential field methods and their inherent approaches for path planning 11, 10801–10805.

Farouki, R.T., 2012. The Bernstein polynomial basis: A centennial retrospective. Computer Aided Geometric Design 29, 379 – 419. URL: http://www.sciencedirect.com/science/article/pii/S0167839612000192, doi:https://doi.org/10.1016/j.cagd.2012.03.001.

Fossen, T.I., 2011a. Guidance Systems, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 241–284. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch10, doi:10.1002/9781119994138.ch10. section: 10 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch10.

Fossen, T.I., 2011b. Introduction, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 227–239. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch9, doi:10.1002/9781119994138.ch9. section: 9 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch9.

Fossen, T.I., 2011c. Introduction, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 1–14. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch1, doi:10.1002/9781119994138.ch1. section: 1 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch1.

Fossen, T.I., 2011d. Kinematics, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 15–44. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch2, doi:10.1002/9781119994138.ch2. section: 2 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch2.

Fossen, T.I., 2011e. Maneuvering Theory, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 109–132. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch6, doi:10.1002/9781119994138.ch6. section: 6 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch6.

Fossen, T.I., 2011f. Motion Control Systems, in: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons, Ltd, pp. 343–415. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch12, doi:10.1002/9781119994138.ch12. section: 12 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch12.

Fossen, T.I., Breivik, M., Skjetne, R., 2003. Line-of-sight path following of underactuated marine craft. IFAC Proceedings Volumes 36, 211 – 216. URL: http://www.sciencedirect.com/science/article/pii/S1474667017378096, doi:https://doi.org/10.1016/S1474-6670(17)37809-6.

Gammell, J.D., Srinivasa, S.S., Barfoot, T.D., 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2997–3004. doi:10.1109/IROS.2014.6942976. iSSN: 2153-0866.

Gasparetto, A., Boscariol, P., Lanzutti, A., Vidoni, R., 2015. Path Planning and Trajectory Planning Algorithms: A General Overview. Mechanisms and Machine Science 29, 3–27. doi:10.1007/978-3-319-14705-5_1.

Gertler, M., Hagen, G., 1960. Handling quality criteria for surface ships. Technical Report. Naval Ship Resarch and Development Center. Washington D.C.

Goldman, R., 2005. Curvature formulas for implicit curves and surfaces.

Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107. doi:10.1109/TSSC.1968.300136.

Jaillet, L., Cortes, J., Simeon, T., 2008. Transition-based RRT for path planning in continuous cost spaces, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2145–2150. doi:10.1109/IROS.2008.4650993. iSSN: 2153-0866.

Johansen, T.A., Fossen, T.I., 2013. Control allocation—A survey. Automatica 49, 1087–1103. URL: https://linkinghub.elsevier.com/retrieve/pii/S0005109813000368, doi:10.1016/j.automatica.2013.01.035.

Joy, K.I., 1999. Breshenham's algorithm. Computer Science Department, University of California, Davis .

Joy, K.I., 2000a. Bernstein Polynomials. Visualization and Graphics Research Group Department of Computer Science University of California, Davis .

Joy, K.I., 2000b. A Matrix Formulation of the Cubic Bézier Curve. Visualization and Graphics Research Group Department of Computer Science University of California, Davis .

Kamsvåg, V., 2018. Fusion Between Camera and Lidar for Autonomous Surface Vehicles. Master's thesis. Norwegian University of Science and Technology. Trondheim.

Karaman, S., Frazzoli, E., 2010. Incremental sampling-based algorithms for optimal motion planning. Robotics Science and Systems VI 104.

Karaman, S., Frazzoli, E., 2011. Sampling-based algorithms for optimal motion planning. The international journal of robotics research 30, 846–894. Publisher: Sage Publications Sage UK: London, England.

Kavraki, L.E., Kolountzakis, M.N., Latombe, J., 1998. Analysis of probabilistic roadmaps for path planning. IEEE Transactions on Robotics and Automation 14, 166–171. doi:10.1109/70.660866.

Kavraki, L.E., Svestka, P., Latombe, J., Overmars, M.H., 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12, 566–580. doi:10.1109/70.508439.

Khalil, H., 2002. Nonlinear Systems. Pearson Education, Prentice Hall. URL: https://books.google.no/books?id=t_d1QgAACAAJ.

Knædal, M., 2019. Stepwise Path-Generation using Bézier Curves. Specialization project. Norwegian University of Science and Technology. Trondheim.

Knædal, M., Sagild, J., Johansen, J., Koivumäki, S., 2018. Monte Carlo Localization of a Mobile Robot. Lab report 1. Insituto Superior Tecnico. URL: https://github.com/magnuok/as_group15_mcl.

Kreyszig, E., Kreyszig, H., Norminton, E.J., 2011. Advanced Engineering Mathematics. Tenth ed., Wiley, Hoboken, NJ.

Krstic, M., Kokotovic, P.V., Kanellakopoulos, I., 1995. Nonlinear and Adaptive Control Design. 1st ed., John Wiley & Sons, Inc., USA.

Kuwata, Y., Karaman, S., Teo, J., Frazzoli, E., How, J., Fiore, G., 2009. Real-Time Motion Planning With Applications to Autonomous Urban Driving. Control Systems Technology, IEEE Transactions on 17, 1105 – 1118. doi:10.1109/TCST.2008.2012116.

LaValle, S.M., 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical Report. Department of Computer Science, Iowa State University. Ames, IA 50011 USA.

LaValle, S.M., 2005. RRT Page: About RRTs. URL: http://msl.cs.uiuc.edu/rrt/about.html.

LaValle, S.M., 2006. Planning Algorithms. Cambridge University Press. doi:10.1017/CBO9780511546877.

LaValle, S.M., Kuffner, J., 2000. Rapidly-Exploring Random Trees: Progress and Prospects. Algorithmic and computational robotics: New directions .

Lekkas, A., 2014. Guidance and Path-Planning Systems for Autonomous Vehicles. PhD Thesis. Norwegian University of Science and Technology.

Lindfors, I., 1993. Thrust allocation method for the dynamic positioning system, in: 10th international ship control systems symposium (SCSS'93), pp. 3–93.

Liu, Z., Zhang, Y., Yu, X., Yuan, C., 2016. Unmanned surface vehicles: An overview of developments and challenges. Annual Reviews in Control 41, 71 – 93. URL: http://www.sciencedirect.com/science/article/pii/S1367578816300219, doi:https://doi.org/10.1016/j.arcontrol.2016.04.018.

Lu, D.V., Hershberger, D., Smart, W.D., 2014. Layered costmaps for context-sensitive navigation, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 709–715. doi:10.1109/IROS.2014.6942636. iSSN: 2153-0866.

Lyche, T., Mørken, K., 2008. Spline Methods Draft. Department of Informatics, Centre of Mathematics for Applications, University of Oslo , 3–4,14–19.

Marley, M., 2019. BézierPathGeneration_asimpleexample.m.

Martelli, M., Zaccone, R., 2018. A random sampling based algorithm for ship path planning with obstacles. doi:10.24868/issn.2631-8741.2018.018.

MiT, 2009. Definition of Bézier curve and its properties — MiT- Massachusetts Institute of Technology. URL: http://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node12.html.

Naderi, K., Rajamäki, J., Hämäläinen, P., 2015. RT-RRT*: a real-time path planning algorithm based on RRT*, pp. 113–118. doi:10.1145/2822013.2822036.

Norbye, H.G., 2019. Real-time sensor fusion for the ReVolt model-scale vessel. Master's thesis. Norwegian University of Science and Technology. Trondheim.

Panati, S., Baasandorj, B., Chong, K., 2015. Autonomous Mobile Robot Navigation Using Harmonic Potential Field. IOP Conference Series: Materials Science and Engineering 83, 012018. doi:10.1088/1757-899X/83/1/012018.

Reif, J.H., 1979. Complexity of the mover's problem and generalizations, in: 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), pp. 587–769. doi:10.1109/SFCS.1979.10. iSSN: 0272-5428.

Runge, C., 1901. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. Zeitschrift für Mathematik und Physik , 224–243.

Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach. 3rd ed., Prentice Hall Press, USA.

Sarfraz, M., Samreen, S., Hussain, M.Z., 2018. A quadratic trigonometric weighted spline with local support basis functions. Alexandria Engineering Journal 57, 1041 – 1049. URL: http://www.sciencedirect.com/science/article/pii/S1110016817300789, doi:https://doi.org/10.1016/j.aej.2017.02.016.

Skjetne, R., 2005. The Maneuvering Problem. PhD Thesis. Norwegian University of Science and Technology.

Skjetne, R., 2019. Maneuvering control design of a low-speed fully-actuated vessel with stepwise path generation. Revision D.

Smogeli, O., 2006. Control of Marine Propellers: from Normal to Extreme Conditions. PhD Thesis. Norwegian University of Science and Technology. Trondheim.

SNAME, 1950. The Society of Naval Architecture and Marine Enigneers. Nomenclature for Treating the Motion of a Submerged Body Through a Fluid. Published: *Technical and Research Bulletin No. 1-5.*

Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., Feyzeau, P., 2013. Path planning: A 2013 survey. Proceedings of 2013 International Conference on Industrial Engineering and Systems Management, IEEE - IESM 2013 .

Speck, J., 2014. Continuity and Discontinuity.

Sørdalen, O.J., 1997. Optimal thrust allocation for marine vessels. Control Engineering Practice 5, 1223 – 1231. URL: http://www.sciencedirect.com/science/article/pii/S0967066197843614, doi:https://doi.org/10.1016/S0967-0661(97)84361-4.

Thrun, S., 2003. Robotic Mapping: A Survey, in: Exploring Artificial Intelligence in the New Millennium. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1–35.

Thrun, S., Burgard, W., Fox, D., 2005. Probabilistic robotics. MIT Press, Cambridge, Mass. URL: http://www.amazon.de/gp/product/0262201623/102-8479661-9831324?v=glance&n=283155&n=507846&s=books&v=glance.

Tvete, H.A., 2020. The ReVolt. URL: https://www.dnvgl.com/technology-innovation/revolt/index.html.

Vagale, A., Bye, R., Oucheikh, R., Osen, O., Fossen, T.I., 2020a. Path Planning and Collision Avoidance for Autonomous Surface Vehicles II: A Comparative Study of Algorithms.

Vagale, A., Oucheikh, R., Bye, R., Osen, O., Fossen, T.I., 2020b. Path Planning and Collision Avoidance for Autonomous Surface Vehicles I: A Review.

Wang, X., Jiang, P., Li, D., Sun, T., 2017. Curvature Continuous and Bounded Path Planning for Fixed-Wing UAVs. Sensors 17, 2155. doi:10.3390/s17092155.

Yu, J., 2016. Intractability of Optimal Multirobot Path Planning on Planar Graphs. IEEE Robotics and Automation Letters 1, 33–40.

Zhang, S., Qian, W.q., 2017. Dynamic backstepping control for pure-feedback nonlinear systems. arXiv:1706.08641 [cs] URL: http://arxiv.org/abs/1706.08641. arXiv: 1706.08641.

Zhang, S., Qian, W.q., Golub, G.H., Van Loan, C.F., 2017. Matrix Computations. arXiv:1706.08641 [cs] URL: http://arxiv.org/abs/1706.08641. edition: 3 ISBN: 9780801854149 Place: Baltimore Publisher: Johns Hopkins.

Appendix A

Parameter Values for ReVolt

This appendix gives the parameter values used for modeling ReVolt. The parameters is adopted from Alfheim and Muggerud (2017). The system inertia matrix $\boldsymbol{M}$ in 3 DOF is the sum of the rigid body inertia:

$$\boldsymbol{M}_{RB} = \begin{bmatrix} 257 & 0 & 0 \\ 0 & 257 & 0 \\ 0 & 0 & 298 \end{bmatrix}, \tag{A.1}$$

and the added mass:

$$\boldsymbol{M}_A = \begin{bmatrix} 6.930 & 0 & 0 \\ 0 & 49.440 & 7.007 \\ 0 & 7.028 & 24.556 \end{bmatrix}. \tag{A.2}$$

Their respective sum becomes:

$$\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A = \begin{bmatrix} 263.93 & 0 & 0 \\ 0 & 306.44 & 7.00 \\ 0 & 7.03 & 322.15 \end{bmatrix}. \tag{A.3}$$

The numerical values for the Coriolis and centripetal matrix $\boldsymbol{C}(\boldsymbol{\nu})$ is:

$$\boldsymbol{C}(\nu) = \begin{bmatrix} 0 & 0 & -207.56v + 7.00r \\ 0 & 0 & 250.07u \\ 207.56v - 7.00r & -250.07u & 0 \end{bmatrix}. \tag{A.4}$$

Lastly, the values of the (linear) damping matrix $\boldsymbol{D}$:

$$\boldsymbol{D} = \begin{bmatrix} 50.66 & 0 & 0 \\ 0 & 601.45 & 83.05 \\ 0 & 83.10 & 268.17 \end{bmatrix}. \tag{A.5}$$

Appendix B

Control Points and Plot of Derivatives from Example 4.1

Table B.1: Control points from Example 4.1.

Segment	Control points
1	$\{(0,0),(.36.15),(.73,.30),(1.10,0.45),$ $(1.15,1.15),(1.43,1.43),(1.7,1.7),(2,2)]\}$
2	$\{(2,2),(2.28,2.28),(2.56,2.56),(2.84,2.84),$ $(3.80,2),(4.20,2),(4.60,2),(5,2)\}$
3	$\{(5,2),(5.40,2),(5.80,2),(6.20,2),$ $(5.46,2.92),(5.64,3.28),(5.82,3.64),(6,4)\}$

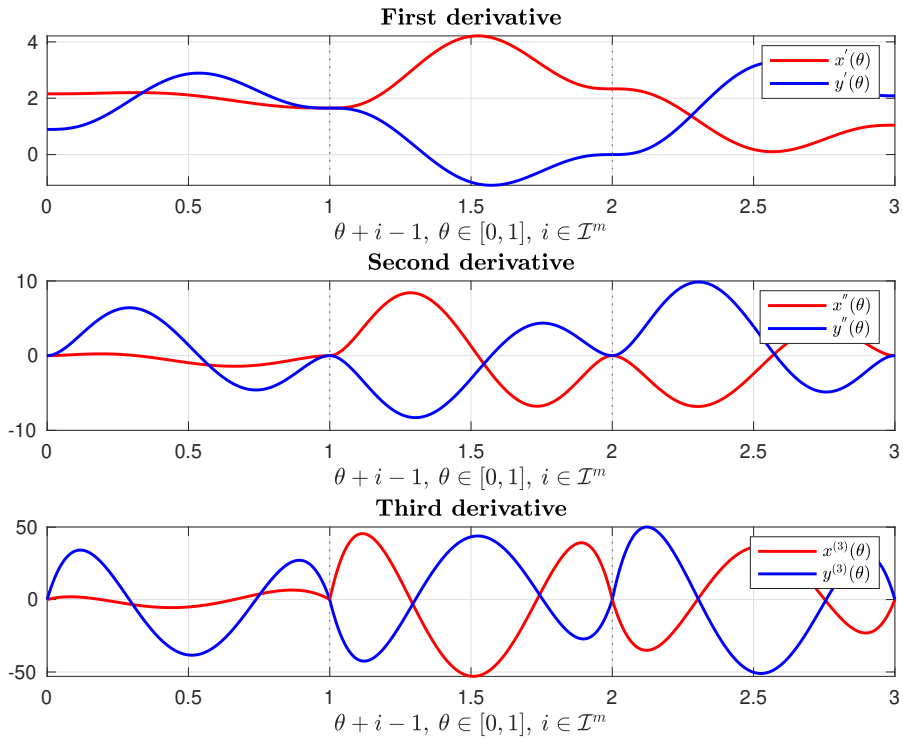


Figure B.1: First-,second-, and third derivatives for the septic Bézier curve in Example 4.1.

Appendix C

Primitive Procedures for RRT

The following section is based on Karaman and Frazzoli (2011) and states the primitive procedures used in RRT*, given in Algorithm 2, and needed for designing the local path planner.

- **Sample:** Uniformly samples from $\mathcal{X}_{free}$. That is:

$$x_{rand} \sim \mathcal{U}(\mathcal{X}_{free}).$$

- **Nearest:** Given $\mathcal{T} = (V, E)$ and $x \in \mathcal{X}_{free}$, the procedure returns the vertex in V that is "closest" to x in terms of a given cost function.
- **Near:** Given $\mathcal{T} = (V, E)$ and $x \in \mathcal{X}_{free}$, the procedure returns the vertices in V that is contained within a ball of a certain radius r. Note that the radius is often implemented as a function of the sample dispersion. See LaValle (2006).
- **Steering:** Given two points $x, y \in \mathcal{X}_{free}$, the function returns a point $z \in \mathcal{X}_{free}$ that minimizes $|z - y|$ while at the same time maintaining that $|z - x| \leq r_{max}$, for a specified $r_{max} > 0$. $|\cdot|$ denotes the distance between the nodes.
- **Collision Test:** Given two nodes $x, y \in \mathcal{X}_{free}$, the procedure checks if the line segment between x and y lies in $\mathcal{X}_{free}$.
- **Parent:** Given a node x, the procedure returns the parent of x.
- **Cost:** Given a node x, the procedure returns the cost of x for a specified cost function (e.g., Euclidean distance).

Appendix D

Waypoints for Simulations and Experiment

The waypoints for S-shaped maneuver used in simulations of the guidance and control system in Chapter 8:

$\mathbf{WP}_0 = (0, 0)$ $\mathbf{WP}_3 = (10, -4)$ $\mathbf{WP}_6 = (6, 12)$ $\mathbf{WP}_9 = (14, 20)$
$\mathbf{WP}_1 = (2, 0)$ $\mathbf{WP}_4 = (14, 0)$ $\mathbf{WP}_7 = (6, 16)$ $\mathbf{WP}_{10} = (18, 16)$
$\mathbf{WP}_2 = (6, -4)$ $\mathbf{WP}_5 = (14, 4)$ $\mathbf{WP}_8 = (10, 20)$ $\mathbf{WP}_{11} = (22, 16)$

The waypoints for 8-shaped maneuver used in experiment conducted of the guidance and control system in Chapter 9:

$\mathbf{WP}_0 = (0, 0)$ $\mathbf{WP}_4 = (20, 0)$ $\mathbf{WP}_8 = (10, 30)$ $\mathbf{WP}_{11} = (20, 20)$
$\mathbf{WP}_1 = (5, 0)$ $\mathbf{WP}_5 = (20, 5)$ $\mathbf{WP}_9 = (15, 30)$ $\mathbf{WP}_{12} = (5, 5)$
$\mathbf{WP}_2 = (10, -5)$ $\mathbf{WP}_6 = (5, 20)$ $\mathbf{WP}_{10} = (20, 25)$ $\mathbf{WP}_{13} = (5, 0)$
$\mathbf{WP}_3 = (15, -5)$ $\mathbf{WP}_7 = (5, 25)$

Appendix E

Plot of Position, Speed, and Forces for the Complete Simulation

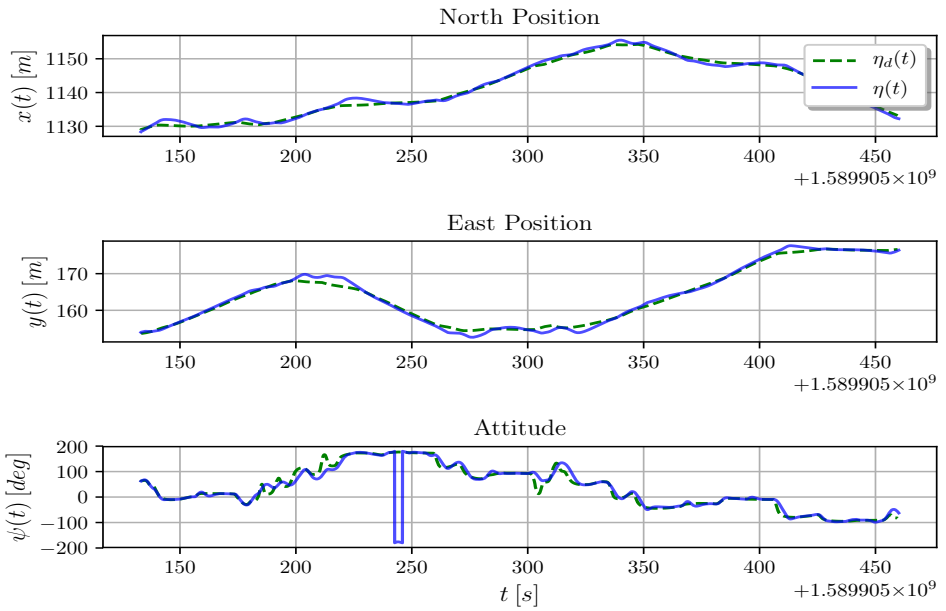


Figure E.1: Position versus time for each DOF for the complete simulation in Section 8.5.

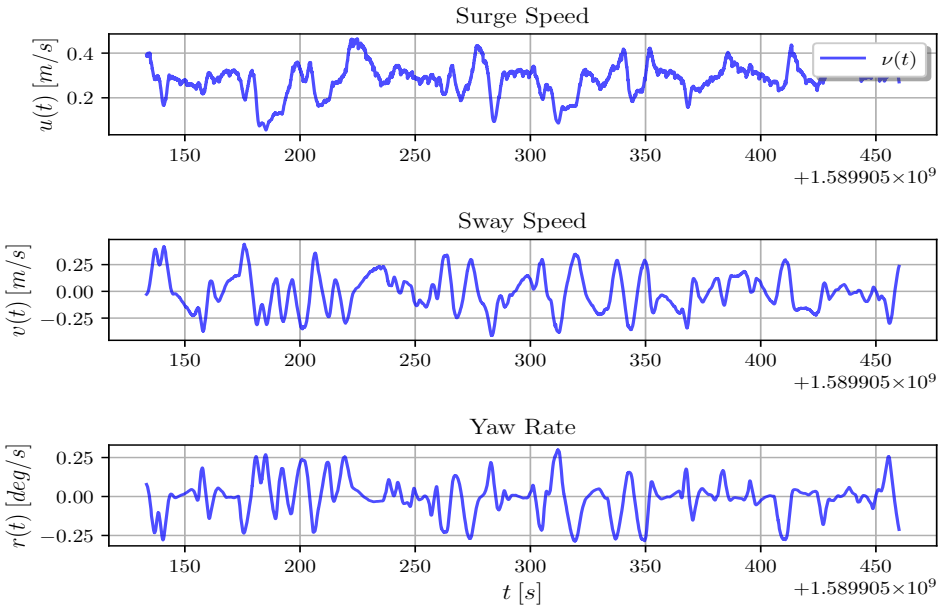


Figure E.2: Speed versus time for each DOF for the complete simulation in Section 8.5.

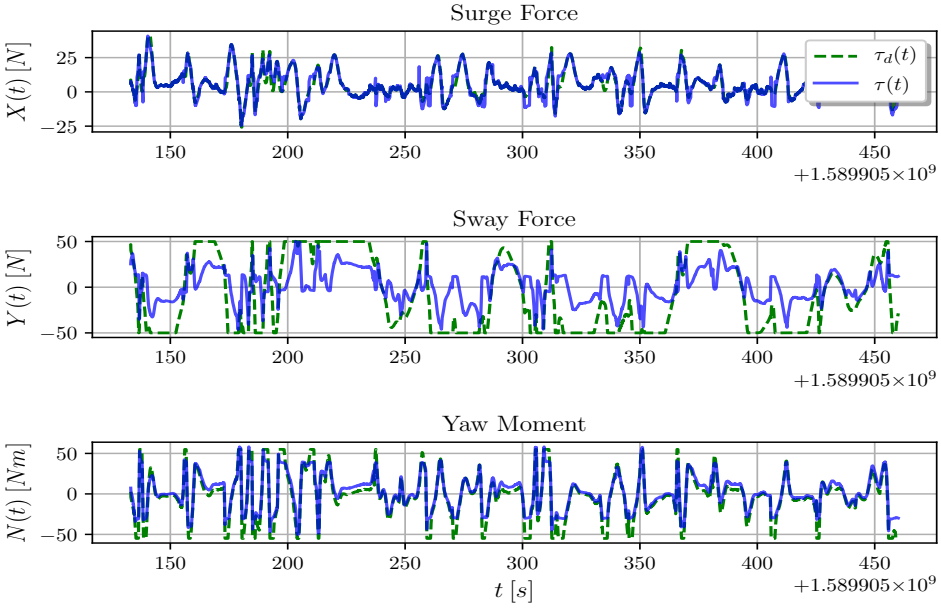


Figure E.3: Forces versus time for each DOF for the complete simulation in Section 8.5.

Magnus Knædal

Autonomous Path Planning and Maneuvering of a Surface Vessel

NTNU
Norwegian University of
Science and Technology