Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering

Kristian Sem

Implementing a Planning-Based Docking System for ReVolt

Master's thesis in Subsea Technology
Supervisor: Christian Holden
Co-supervisor: Tom Arne Pedersen
June 2023

NTNU
Norwegian University of
Science and Technology

Kristian Sem

Implementing a Planning-Based Docking System for ReVolt

Master's thesis in Subsea Technology
Supervisor: Christian Holden
Co-supervisor: Tom Arne Pedersen
June 2023

Norwegian University of Science and Technology
Faculty of Engineering

Preface

This thesis concludes my 2-year Master of Science degree in Subsea Technology at the Norwegian University of Science and Technology (NTNU). The thesis was written in collaboration with DNV and continues the pre-project submitted in the fall of 2022.

I would like to thank my supervisors, Christian Holden and Tom Arne Pedersen, for their patience and wise words through the journey of a pre-project and a master thesis, while a lot of my focus has been on being Deputy Lead at Vortex NTNU.

Furthermore, I want to express my gratitude to my friends, family, and especially my partner Sofie, who brightened my life as a student and surely for years to come.

Abstract

This thesis proposes a method for path planning utilizing the rasterization of Environmental Systems Research Institute (ESRI) shapefiles. These files contain geospatial vector data that gets transformed into a local navigation frame and then into an array through a rasterization process. This transformation yields a grid of cells symbolizing various obstacles on a spatial map, thereby presenting an effective means of modeling the environment for navigation tasks.

By using this rasterization, one can adapt the resolution of the grid-based array in alignment with the required precision of the path planning task. This flexibility allows for optimizing the balance between computational load and accuracy in diverse applications, opening up the possibility for high-precision path planning and navigation tasks.

The approach utilizes the strengths of the A* algorithm, an established and recognized algorithm for its efficiency and precision in pathfinding tasks. In combination with this, the Douglas-Peucker algorithm is employed to simplify the path. This approach ensures the robustness of our method in finding an initial path that is free from obstacles.

To evaluate the feasibility of the proposed path, a simplistic kinematic model is implemented as part of an optimization problem. This process ensures that the planned path is practically navigable and feasible under closer to real-world conditions.

While developing the scripts for this method, extra attention was paid to writing testable, maintainable code in line with best practices for software development. The thesis follows a meticulous development process to ensure the reliability and robustness of the proposed method. This approach to design and testing ensures the practicality of the method and provides a robust foundation for future improvements and adaptations.

Sammendrag

Denne oppgaven foreslår en metode for baneplanlegging ved bruk av rasterisering av formfiler fra Environmental Systems Research Institute (ESRI). Disse filene inneholder geospatiale vektordata som blir transformert til en lokal navigasjonsramme og deretter til en matrise gjennom en rasteriseringsprosess. Denne transformasjonen gir et rutenett av celler som symboliserer ulike hindringer på et romlig kart, og presenterer dermed et effektivt middel for å modellere miljøet for navigasjonsoppgaver.

Ved å bruke denne rasteriseringen kan man tilpasse oppløsningen til den grid-baserte matrisen i samsvar med den nødvendige presisjonen til baneplanleggingsoppgaven. Denne fleksibiliteten tillater å optimere balansen mellom beregningsbelastning og nøyaktighet i ulike applikasjoner, og åpner for muligheten for høypresisjons stiplanlegging og navigasjonsoppgaver.

Tilnærmingen utnytter styrkene til A*-algoritmen, en etablert og anerkjent algoritme for sin effektivitet og presisjon i banesøkende oppgaver. I kombinasjon med dette brukes Douglas-Peucker-algoritmen for å forenkle banen. Denne tilnærmingen sikrer robustheten til metoden vår når det gjelder å finne en første vei som er fri for hindringer.

For å evaluere gjennomførbarheten av den foreslåtte banen, implementeres en forenklet kinematisk modell som en del av et optimaliseringsproblem. Denne prosessen sikrer at den planlagte stien er praktisk navigerbar og gjennomførbar under nærmere virkelige forhold.

Under utviklingen av skriptene for denne metoden ble det lagt ekstra vekt på å skrive testbar, vedlikeholdbar kode i tråd med beste praksis for programvareutvikling. Oppgaven følger en nitid utviklingsprosess for å sikre påliteligheten og robustheten til den foreslåtte metoden. Denne tilnærmingen til design og testing sikrer at metoden er praktisk og gir et robust grunnlag for fremtidige forbedringer og tilpasninger.

Contents

Preface i

Abstract iii

Sammendrag v

Abbreviations xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Description . . . 2
- 1.3 Outline . . . 2

2 Theoretical background 3
- 2.1 Notation and reference frames . . . 3
  - 2.1.1 ECEF . . . 3
  - 2.1.2 NED . . . 3
  - 2.1.3 Body frame . . . 4
- 2.2 3-DOF Ship Maneuvering Model . . . 5
- 2.3 Path finding with A* . . . 6
- 2.4 Waypoint reduction . . . 7
- 2.5 Optimal Control and Thrust allocation . . . 8
  - 2.5.1 Integrating System Matrices into the Solver . . . 8
  - 2.5.2 Optimizing for Energy . . . 9

3 System description 11
- 3.1 The scale model . . . 11
  - 3.1.1 The scale model matrices . . . 11
- 3.2 Simulator . . . 12
- 3.3 Software . . . 13
  - 3.3.1 Programming language . . . 13
  - 3.3.2 ROS . . . 13
  - 3.3.3 Robot Operating System (ROS) . . . 13
- 3.4 gRPC . . . 14

4 Design and implementation 15
- 4.1 Methodology: Development and Operations . . . 15
- 4.2 Development . . . 16
  - 4.2.1 Step 1: Plan . . . 16
  - 4.2.2 Step 2: Coding . . . 16

4.2.3 Step 3: Building/Integrating 17
4.2.4 Step 4: Test 17
4.2.5 Step 5-8 17
4.3 Path planning 19
4.3.1 Array extraction from shapefiles 19
4.3.2 Pathfinding 19
4.3.3 GUI 22
4.3.4 Unity 23
4.3.5 ROS wrapping 24
4.4 Replanning 24
4.4.1 Path optimization 26

5 Discussion **29**

6 Conclusion **31**
6.1 Recommended further work 31

List of Figures

1.1 The concept ship ReVolt[4] 2

2.1 Reference frames and an example relation between them. [5] 4
2.2 Body frame notations. [5] 5
2.3 Positioning of a vessel in NED. [2] 5
2.4 A* example, moving to adjacent cells costs 10 and diagonal costs 14. . . . 7
2.5 Douglas-Pucker algorithm applied with different tolerances, [3] 8

3.1 The dimensions of the ReVolt ship model. [1] 11
3.2 Image from the Unity simulator 13
3.3 simple graph of communication in ROS 14
3.4 simple graph of service in ROS 14
3.5 gRPC communication 14

4.1 DevOps process loop [16] 15
4.2 Upperlevel flowcharts describing the required modules for each part of the system. 16
4.3 ros-param 17
4.4 Test flowchart 18
4.5 Shapefile extraction flowchart 19
4.6 Different levels of rasterization. One cell is one unit in NED. The blue area in the upper left subfigure corresponds to the black area in the remaining subfigures. The white area is obstacle free space. 20
4.7 Astar and path smoothing flowchart 20
4.8 Plot of time taken based on robot size 21
4.9 Vessel size acting as a safe barrier when finding a path 21
4.10 Waypoint reduction with Douglas-Peucker 22
4.11 ReVolt in Unity (left) and in RviZ (right) 23
4.12 Body axis marker is the vessel, DP_PID axis marker is the desired end pose and position for next waypoint. Red spheres are waypoints. Red arrow is the end position and pose for the vessel 23
4.13 Astar and path smoothing flowchart 24
4.14 image in Two horizontal 25
4.15 Flowchart for handling size and resolution when extracting geometries . . . 25
4.16 Path optimized for time usage with the simple model. Path is feasible with the current waypoints 27
4.17 Path optimized for time usage with the simple model. The shortest path found by the solver would have gone over land 28

List of Tables

2.1 Table fo reference frames and their axises [5] 4

3.1 The thruster placement of the vessel [14] 12

4.1 Processing time for extracting arrays at different resolutions 19
4.2 Search times for A* with robot radius acting as a safety radius 21
4.3 Search times for A* with border radius around obstacles instead of robot safety radius 22

Abbreviations

ASV	=	Autonomous Surface Vehicle
CI/CD	=	Continuous Integration and Continuous Delivery
COLAV	=	Collision Avoidance
COLREG	=	Convention on the International Regulations for Preventing Collisions at Sea
DevOps	=	Development and Operations
DNV	=	Det Norske Veritas
DOF	=	Degrees Of Freedom
DP	=	Dynamic Positioning
ECEF	=	Earth Centered Earth Fixed
ESRI	=	Environmental Systems Research Institute
ENC	=	Electronic Navigational Chart
GNSS	=	Global Navigation Satellite Systems
IaC	=	Infrastructure as Code
IAE	=	Integral absolute Error
NE	=	North East
NED	=	North, East Down
OCP	=	Optimal Control Problem
PID	=	Proportional–Integral–Derivative
ROS	=	Robot Operating System
RPC	=	Remote Procedure Calls
RPS	=	Rounds Per Second
SNAME	=	Society of Naval Architects and Marine Engineers
QP	=	Quadratic Programming
OS	=	Operating System

Chapter 1

Introduction

1.1 Motivation

Over the last few decades, there has been a growing interest in developing autonomous vessels that can perform various tasks with little to no human intervention [11, 9].

This trend has been driven by a combination of factors, where three is summarized here. (*i*), the desire to reduce human intervention, meaning that the vessel can operate in conditions too dangerous for a human crew while also reducing both human errors and costs. (*ii*), the potential for increased efficiency, where a computer program can adhere to predefined schedules and routes, and (*iii*) the ambition to decrease the environmental impact of shipping as a computer program can consider reducing fuel consumption to a greater extent than a human captain. Since autonomous vessels are yet to be implemented commercially, there exist little research on the operational benefits of such vessels [8].

Even though autonomy is becoming an increasingly mature technology on land based vehicles, its development on autonomous vessels are still focusing on the fundamental technology behind its operation. In [7], the authors found that two thirds of the papers on autonomous vessels still discuss trajectory planning and obstacle avoidence. Trajectory planning is important when developing autonomous vessels, as they mostly do not operate on fixed roads as land based vehicles. Similarily, obstacle avoidance can be solved differently for vessels as it often is possible to circumfere obstacles rather than simply waiting as one would do in a car driving in traffic.

One of the most challenging aspects of developing autonomous vessels is ensuring that they can safely and effectively navigate in complex environments, such as harbors and docks. These areas are characterized by tight spaces, high traffic, and a variety of potential obstacles. This makes them some of the most challenging areas for any vessel to navigate, while also being the most essential part of an autonomous vessel's route.

To address the need for environmentally friendly, unmanned cargo ships for short-sea shipping, DNV GL designed The ReVolt concept ship visualized in figure 1.1. It is a small, slow-moving vessel that is charged while docked, allowing it to operate without the need for a crew. This design makes the ship more efficient and cost-effective while also reducing the risk of accidents and injuries to crew members [4].

Figure 1.1: The concept ship ReVolt[4]

1.2 Problem Description

The aim of this thesis is to take steps toward creating an automated method for maneuvering and docking the ReVolt model. The objective is to build a reliable, robust and energy efficient automated docking system, which will reduce the need for human intervention and minimize the risk of accidents or errors. The contributions of this thesis is summarized in the following points:

- Develop a trajectory planner
- Obstacle map for planning trajectory
- Use A* to compute a trajectory for starters
- Optimize the trajectory to allow for a smoother trajectory
- GUI to allow for easier operation and testing
- Reschedule if new obstacles appear in the original path

1.3 Outline

The Introduction sets the stage by discussing the broader context of autonomous navigation and the role of pathfinding algorithms within it.

- Chapter 2 - Theoretical background presents relevant theoretical knowledge for the thesis.
- Chapter 3 - System description, describes the system and choice of software tools for the thesis.
- Chapter 4 - Design and implmentation, describes the method for how the scripts gets devloped and integrated.
- Chapter 5 - The Discussion chapter presents our findings and explores their implications, demonstrating the algorithm's efficiency and potential applications.
- Chapter 6 - Conclusion, wraps up the thesis by summarizing our findings and discussing potential avenues for future research.

Chapter 2

Theoretical background

2.1 Notation and reference frames

The three frames described in this section play an important role in autonomous vessel navigation. The ECEF frame is beneficial for extracting map data and determining the vessel's position on Earth. It allows for precise positioning information based on GNSS coordinates, essential for accurate navigation. The NED frame, on the other hand, is crucial for local navigation, obstacle avoidance, and pathfinding. This frame allows the vessel to determine its position relative to its immediate surroundings and navigate accordingly. Lastly, the Body frame is essential for state estimation and applying the correct forces to the vessel. With the vessel's center of gravity as the origin, this frame provides a reference point for measuring the vessel's linear and angular velocities and applying appropriate control actions.

The different frames and their notation is based on the SNAME (1950) [13] notation for marine vessels. The notations describe the:

- Global frame - Earth-Centered-Earth-Fixed (ECEF) frame
- Local Area frame - North-East-Down (NED) frame
- Vessel frame - BODY frame

2.1.1 ECEF

The ECEF frame $\{e\}$ represents a point $P_e = [x_e\ y_e\ z_e]^T$ with respect to the Earth center and its rotation. This reference system is also commonly represented in terms of the Global Navigation Satellite System (GNSS) coordinates, which consist of longitude, latitude, and altitude: $P_g = [l\ \mu\ h]^T$. The point is fixed in relation to the earth's rotation.

2.1.2 NED

The North-East-Down (NED) frame $\{n\}$ is a local frame that is commonly used in navigation. It is defined such that its origin is fixed to a specific ECEF coordinate, and its axes are aligned with true North, East, and the direction of the normal to the Earth's

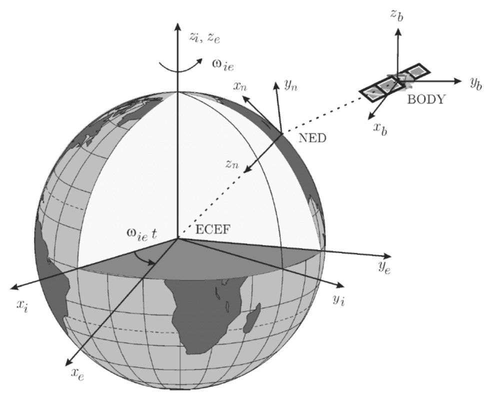


Figure 2.1: Reference frames and an example relation between them. [5]

surface at the origin. A point $P_n = [x_n\ y_n\ z_n]^T$ on the tangent plane can be represented using this frame.

2.1.3 Body frame

The body reference frame $\{b\}$ for a vessel is defined by three axes $[x_b\ y_b\ z_b]^T$, where x_b is parallel to and aligned with the aft-to-fore line of the vessel, y_b is positive on the starboard side, and z_b is normal to the vessel and points downward. The center of the frame is located at the vessel's center of gravity (CG).

		Body		NED
DOF	- (Degrees of freedom)	Forces and moments	Linear and angular velocities	Positions and euler angles
1	Motion in the x_b-direction (Surge)	X	u	x^n
2	Motion in the y_b-direction (Sway)	Y	v	y^n
3	Motion in the z_b-direction (Heave)	Z	w	z^n
4	Rotation in the x_b-axis (Roll)	K	p	ϕ
5	Rotation in the y_b-axis (Pitch	M	q	θ
6	Rotation in the z_b-axis (Yaw	N	r	ψ

Table 2.1: Table fo reference frames and their axises [5]

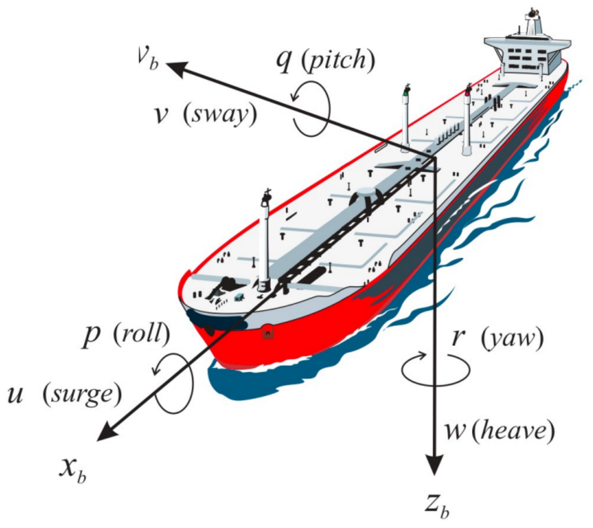


Figure 2.2: Body frame notations. [5]

2.2 3-DOF Ship Maneuvering Model

The full maneuvering model for a vessel is divided into 6 degrees of motion (DOF) as seen in table 2.1. Due to the size of ReVolt and the goal to control it in the horizontal plane, the heave, roll, and pitch forces can be assumed to be $\phi = \theta = z \approx 0$. This results in a 3-DOF model

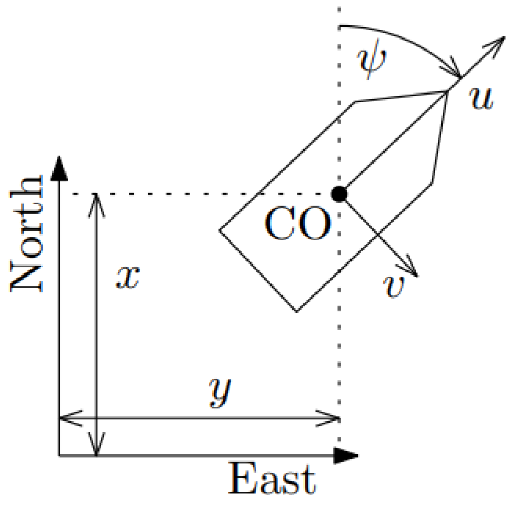


Figure 2.3: Positioning of a vessel in NED. [2]

The nonlinear maneuvering model can be written as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{2.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} + \boldsymbol{\tau}_{environmental}. \tag{2.2}$$

Since roll and pitch are neglected, the relation between NED and BODY can be described through the rotational matrix based on the yaw rotation around the z-axis, where the matrices are expressed by

$$\boldsymbol{C} = \boldsymbol{C}_A(\nu) + \boldsymbol{C}_{RB}(\nu) \tag{2.3}$$

$$\boldsymbol{M} = \boldsymbol{M}_A + \boldsymbol{M}_{RB}. \tag{2.4}$$

The Coriolis and centripetal matrices are expressed as:

$$\begin{aligned}\boldsymbol{C}(\nu) = \boldsymbol{C}_{RB}(\nu) + \boldsymbol{C}_A(\boldsymbol{\nu}) &= \begin{bmatrix} 0 & -mr & -mx_g r \\ mr & 0 & 0 \\ mx_g r & 0 & 0 \end{bmatrix} \\ &+ \begin{bmatrix} 0 & 0 & Y_{\dot{v}} v_r + Y_{\dot{r}} r \\ 0 & 0 & -X_{\dot{u}} u_r \\ -Y_{\dot{v}} v_r - Y_{\dot{r}} r & X_{\dot{u}} u_r & 0 \end{bmatrix}.\end{aligned} \tag{2.5}$$

The system inertia consisting of added mass to the system $\boldsymbol{M}_A$ and the rigid body inertia $\boldsymbol{M}_{RB}$ is expressed as:

$$\boldsymbol{M} = \boldsymbol{M}_A + \boldsymbol{M}_{RB} = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{u}} & -Y_{\dot{r}} \\ 0 & -N_{\dot{u}} & -N_{\dot{r}} \end{bmatrix} + \begin{bmatrix} m & 0 & 0 \\ 0 & 0 & mx_g \\ 0 & mx_g & I_z \end{bmatrix}. \tag{2.6}$$

The dampening matrix $\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}$

$$\boldsymbol{D} = \begin{bmatrix} -X_u & 0 & 0 \\ 0 & -Y_\nu & -Y_r \\ 0 & -N_\nu & -N_r \end{bmatrix}. \tag{2.7}$$

The yaw-dependent rotational matrix from the body frame to the NED frame is given by:

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.8}$$

2.3 Path finding with A*

A popular choice for solving a path-finding problem is the A* algorithm since it finds a solution if one exists. While it also finds the shortest solution of the options available, it is also quite quick for even complex and large environments.

To use the A* algorithm, the environment needs to be discretized into a configuration space, like a grid. The nodes created by A* represent points in the environment and the lines represent the connections between them. The algorithm starts at the given starting point and calculates the cost of reaching each node by adding the cost of moving from the starting node to the current node and the estimated cost of moving from the current node to the goal. The algorithm then selects the node with the lowest total cost and continues to search from that node until it reaches the goal or determines that a solution is not possible.

However, A* can only provide an accurate initial guess if the heuristic is admissible, meaning that it never overestimates the actual cost of reaching the goal from any position. If the heuristic is not admissible, A* may not be able to find the optimal path, even if one exists [15]. The following equation represents the estimated cost $f(n)$ of reaching a goal from node (n)

$$f(n) = g(n) + h(n) \tag{2.9}$$

where $g(n)$ represents the cost of the path, and $h(n)$ represents a problem-specific cost. In figure 2.4, it can be observed that the algorithm chooses the shortest path in total cost even though the diagonal steps are more expensive for each iteration. The octile movement cost in 2.4 is calculated using the following:

$$\text{heuristic}(a, b) = \max(\Delta x, \Delta y) + (\sqrt{2} - 1) \times \min(\Delta x, \Delta y). \tag{2.10}$$

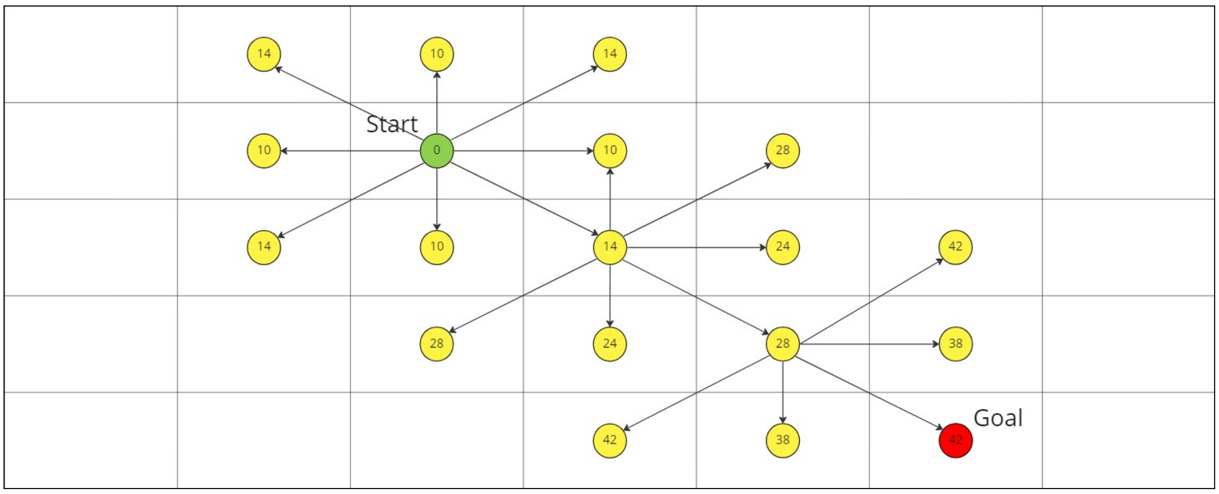


Figure 2.4: A* example, moving to adjacent cells costs 10 and diagonal costs 14.

Heuristics

The A* algorithm requires a configuration space to search through to generate an initial path for the vessel. A two-dimensional grid is the simplest form for a configuration space since it provides simple cost estimation when the value of the grid cells is either occupied or unoccupied. A two-dimensional grid will limit the possible direction of movement, essentially making Euclidean and octile movement the same.

- Manhattan: Manhattan distance, suitable for cardinal movements only
- Octile: Octile distance, suitable for cardinal and diagonal movements
- Euclidean: Euclidean distance, suitable for any direction of movement

2.4 Waypoint reduction

To reduce the number of waypoints the Douglas-Peucker algorithm [3] follows a recursive approach to divide a line into segments. It keeps the endpoints of the line and creates a segment as the first approximation. It then finds the point with the greatest distance to this segment and divides the existing segment into two new if the point is greater than ϵ away. It works through the line until all points within the threshold ϵ gets filtered out. By doing so, the algorithm preserves the most important parts of the line and maintains the essential shape.

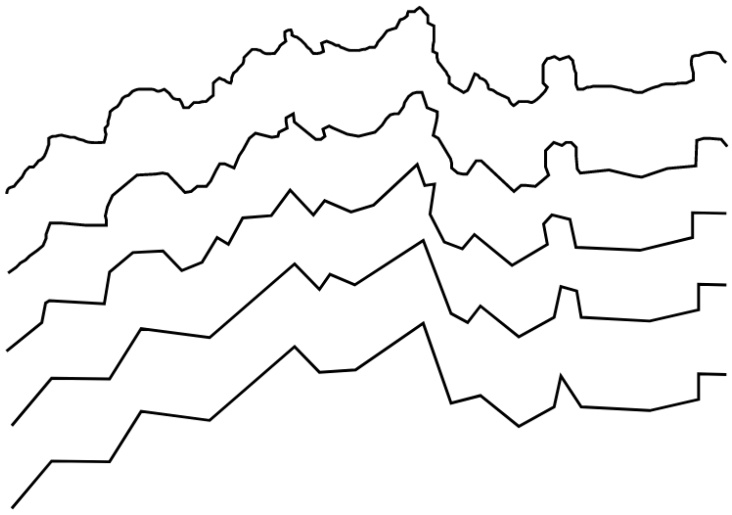


Figure 2.5: Douglas-Pucker algorithm applied with different tolerances, [3]

2.5 Optimal Control and Thrust allocation

Optimal control is a powerful mathematical framework used to determine the best control inputs over time for a dynamic system, considering specific objectives, constraints, and system dynamics. It finds application in various fields, including robotics, aerospace, process control, and energy systems. By formulating an optimization problem, optimal control algorithms provide solutions that optimize system behavior, improve performance, and achieve desired goals.

2.5.1 Integrating System Matrices into the Solver

To solve an optimal control problem, it is crucial to incorporate the system dynamics into the solver. This involves constructing system matrices that represent the state dynamics, control inputs, and their relationship over time. The system matrices typically include the state transition matrix, control input matrix, and possibly additional matrices representing constraints or performance criteria.

The state dynamics can be represented as

$$\dot{x}(t) = A(t)x(t) + B(t)u(t), \tag{2.11}$$

where $x(t)$ is the state vector, $u(t)$ is the control input vector, and the matrices $A(t), B(t)$ are the system matrices capturing the time-varying dynamics.

By discretizing the system dynamics using numerical methods, such as Euler's method or Runge-Kutta methods, we can approximate the continuous-time dynamics in discrete time steps. This results in the following system update equation

$$x_{k+1} = A_k x_k + B_k u_k, \tag{2.12}$$

where x_k and u_k represent the state and control input at time step k, and A_k and B_k are the system matrices at that time step.

The optimal control problem can then be formulated as an optimization problem, aiming to minimize an objective function J subject to constraints. The objective function can be defined based on system performance criteria, such as minimizing energy consumption, maximizing efficiency, or achieving desired trajectories. Constraints can include physical limitations, operational bounds, or safety requirements.

By incorporating the system matrices into the solver, optimal control algorithms can efficiently compute the optimal control inputs that achieve the desired system behavior while satisfying constraints and performance criteria.

2.5.2 Optimizing for Energy

Energy optimization plays a vital role in surface vessel operations, aiming to minimize fuel consumption, increase efficiency, and reduce environmental impact. Optimal control techniques offer a systematic approach to optimize vessel behavior and control inputs to achieve energy-efficient operation.

In the context of surface vessels, optimal control can be applied to different aspects, such as propulsion systems, navigation, and route planning. By considering factors such as vessel speed, load conditions, environmental conditions, and operational constraints, optimal control algorithms can determine the optimal control actions that minimize energy consumption while meeting operational requirements.

For propulsion systems, over the operational time period $[t_0, t_f]$ the energy optimization problem can be formulated as

$$\min J = \int_{t_0}^{t_f} f(x(t), u(t))dt, \tag{2.13}$$

where $x(t)$ represents the vessel state, $u(t)$ represents the control input (e.g., thrust), and $f(x(t), u(t))$ represents the energy cost function. The objective is to minimize the integral of the energy cost function over the operational time period t_0 to t_f.

By solving the energy optimization problem using optimal control techniques, surface vessels can achieve energy-efficient operation. The optimal control algorithm determines the optimal control inputs, such as the thrust profile, that minimize energy consumption while considering operational requirements and constraints.

Furthermore, optimal control can also be utilized in navigation and route planning to optimize energy usage. By considering factors like sea currents, wind conditions, and vessel characteristics, optimal control algorithms can determine the most energy-efficient routes and navigation strategies. This involves optimizing vessel speed, heading, and trajectory planning to minimize fuel consumption and maximize energy efficiency.

The general form of an OCP, with the objective function to be minimized and the constraints to be satisfied:

$$\begin{aligned} J = \min_{u(t)} \quad & \int_{t_0}^{t_f} L(x(t), u(t), t) dt \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t), t) \\ & x(t_0) = x_0 \\ & x(t_f) = x_f \end{aligned} \tag{2.14}$$

Chapter 3

System description

3.1 The scale model

The test platform version is a 1:20 scale model of the concept idea. DNV ordered the project keeping in mind that the boat was meant for testing and development. For an extensive walk-through, the reader should be directed to Alfheim and Muggerud's thesis [1]. They examined the physical features, extracted the values needed for the 3-DOF maneuvering model, and outlined the steps in getting the vessel ready. The authors thoroughly examine these topics, offering a comprehensive understanding of the vessel's components and setup process. In addition, the code base is built to interface with ROS, which takes care of the communication and the data types used between scripts.

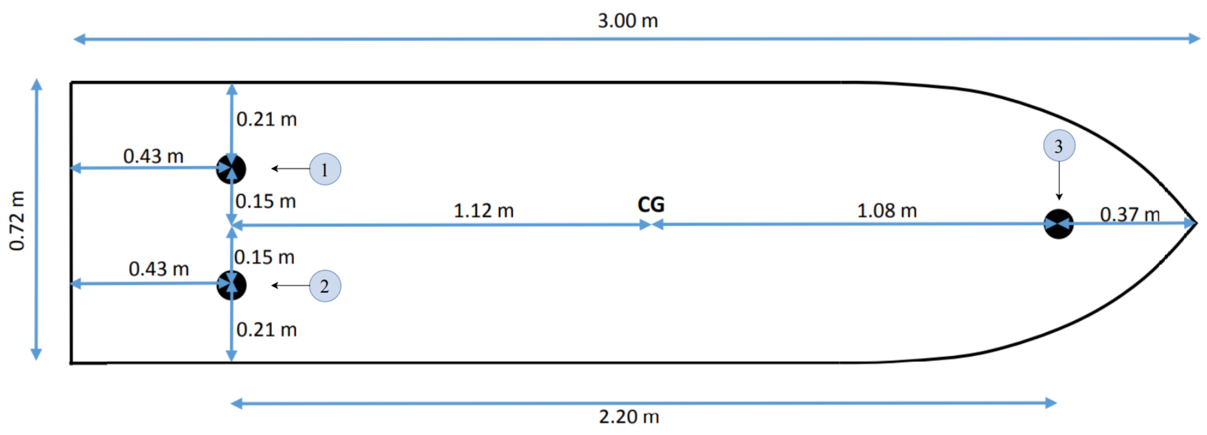


Figure 3.1: The dimensions of the ReVolt ship model. [1]

The scale model, visualized in figure 3.1, is about three meters long and has three thrusters. Two stern thrusters and one bow thruster. In addition to the original sensors in Alfheim and Muggerud's thesis, Radar, Lidar, and a 360° camera have been added. The existing control allocation to the vessel was provided by DNV.

3.1.1 The scale model matrices

The ReVolt scale model matrices and thruster coeeficients found in the thesis by Alfheim and Muggerud [1] are expressed as:

Thruster	$\boldsymbol{l_x}[m]$	$\boldsymbol{l_y}[m]$	Azimuth angles
1	-1.12	-0.15	Fully rotatable
2	-1.12	0.15	Fully rotatable
3	1.08	0	(-270°, 270°)

Table 3.1: The thruster placement of the vessel [14]

$$M = MRB + MA = \begin{bmatrix} 263.93 & 0 & 0 \\ 0 & 306.44 & 7.00 \\ 0 & 7.03 & 322.15 \end{bmatrix} \tag{3.1}$$

$$C(\nu) = CRB(\nu) + CA(\nu) = \begin{bmatrix} 0 & 0 & -207.56\nu + 7.00r \\ 0 & 0 & 250.07u \\ 207.56\nu - 7.00r & -250.07u & 0 \end{bmatrix} \tag{3.2}$$

$$D = \begin{bmatrix} -Xu & 0 & 0 \\ 0 & -Yv & -Yr \\ 0 & -Nv & -Nr \end{bmatrix} = \begin{bmatrix} 50.66 & 0 & 0 \\ 0 & 601.45 & 83.05 \\ 0 & 83.10 & 268.17 \end{bmatrix} \tag{3.3}$$

$$K_{\pm 1} = K_{\pm 2} = 2.7 \times 10^{-3} \quad K_{-3} = 6.172 \times 10^{-4} \quad K_{+3} = 1.518 \times 10^{-3}. \tag{3.4}$$

3.2 Simulator

The Unity simulator is used to create a 3D visualization of the ReVolt ship and the surrounding environment. In addition to simulating the environment, the simulator is also able to simulate LIDAR, Radar, and camera. The data generated from the simulated sensors is based on the 3D environment and the models existing in the vicinity of the sensors. It receives position, pose, and speed data from the control system. It sends out simulated sensor data and positional data.

The environment within the simulator was carefully crafted to reflect real-world measurements and behaviors as described in [19], and a lot of work was done to achieve a high-fidelity simulation that reflects the real world.

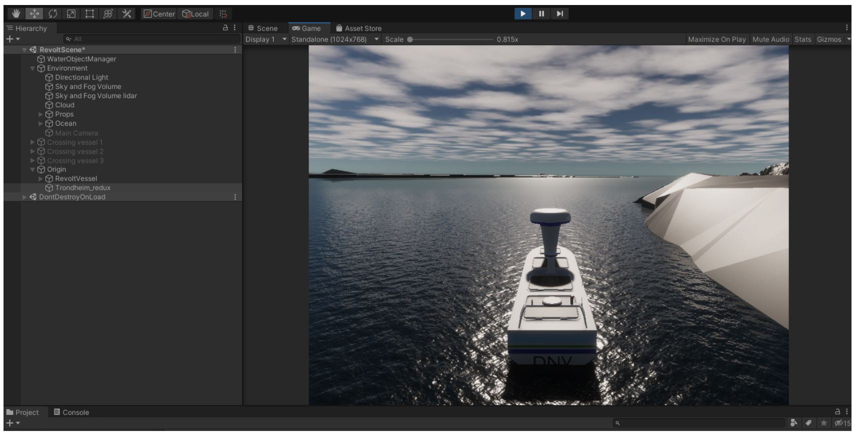


Figure 3.2: Image from the Unity simulator

3.3 Software

The selected software is based on the software already used on the platform and the authors familiarity with the tools.

The following subsection will introduce the softwares used in this thesis.

3.3.1 Programming language

During offline testing, or in a simulator, you don't have the need for exceptional performance or effective code as it is in the development phase. To exercise faster development and since it is well supported in ROS the choice fell on Python. The sheer amount of packages available is also supporting the choice.

3.3.2 ROS

The ReVolt uses ROS to communicate and share data between its various scripts and modules. ROS is a framework that allows developers to create robot applications by providing and open-source set of libraries and tools. It has a set of standard Application Programming Interfaces (APIs) and services for common robot functionality, such as controlling hardware, implementing often used functions, and sending messages between processes. Data transfer can be seastblished with a one way tunnel called a "topic" (figure 3.4) or create a server client relation ship, where every request gets a respons (figure 3.4). This makes it easier for developers to focus on building their robot applications. ROS also allows multiple processes to run on different computers and communicate with each other over a network, enabling the creation of distributed and collaborative robot systems.

3.3.3 Robot Operating System (ROS)

The ReVolt project is developed around ROS, where it is used to handle data streams. Due to this, it is necessary to keep developing with ROS to have access to previously developed and implemented software.

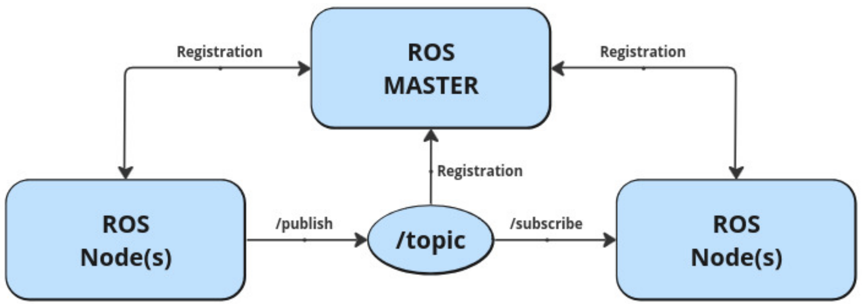


Figure 3.3: simple graph of communication in ROS

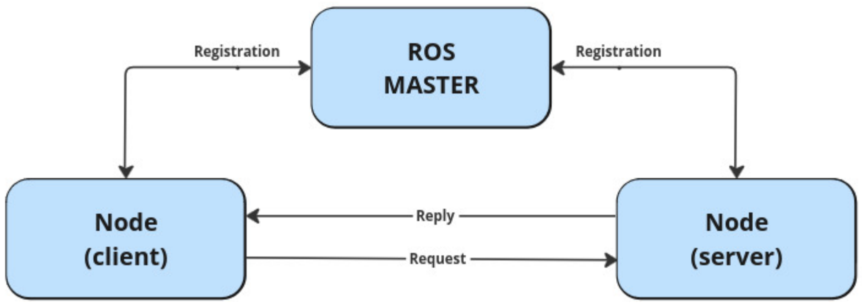


Figure 3.4: simple graph of service in ROS

3.4 gRPC

gRPC is a framework for Remote Procedure Calls (RPC) that allows you to establish client-server communication between different programs and services. Unlike ROS, gRPC does not require a master node to establish communication between server and client. The request and response message and communication are defined in a Protocol Buffer file (.proto file). The compiler generates the necessary server and client files to various programming languages (C++, Java, Python, etc...) based on what the user requests. How gRPC communicates is illustrated in fig. 3.5, where it is evident that the communication process is similar to ROS services.

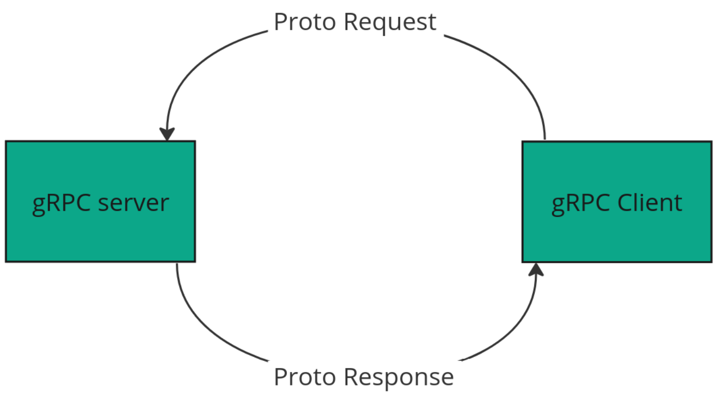


Figure 3.5: gRPC communication

Chapter 4

Design and implementation

4.1 Methodology: Development and Operations

Development and operations (DevOps) is a software development and delivery methodology. From the entirety of the DevOps concept, the most important parts for this project have been continuous integration and Continuous delivery (CI/CD), and Infrastructure as code (IaC). These parts is strongly connected to the left part of the DevOps process loop in figure 4.1.

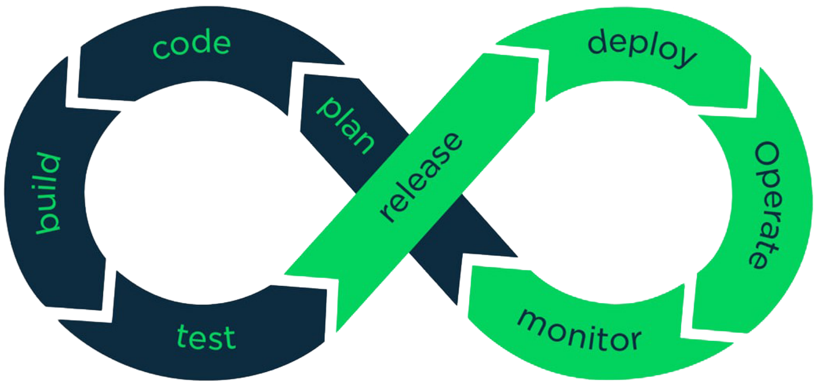


Figure 4.1: DevOps process loop [16]

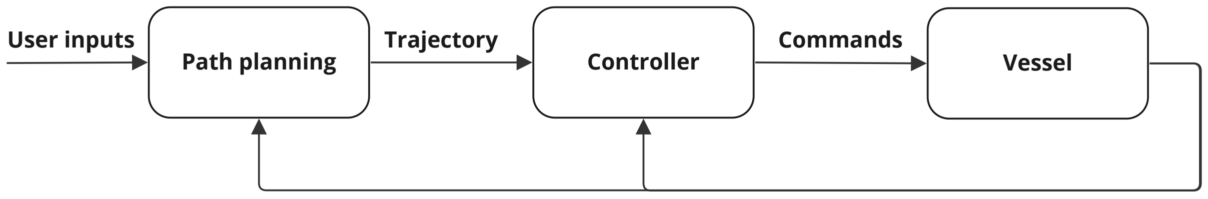
CI allows for changes to be made frequently and tests if the code compiles and returns expected messages after receiving input. The CD part acts as a performance check to see how the changes in the code have affected the returns. The performance measurement is based on user-defined expectations or previous results.

Keeping IaC in mind while finishing a module allows for standardizing the setup, changing values and creating repeatable environments. This is important as it is desirable to develop modules that will be maintained and used by others in the future, making standardization and reproducability core aspects of development.

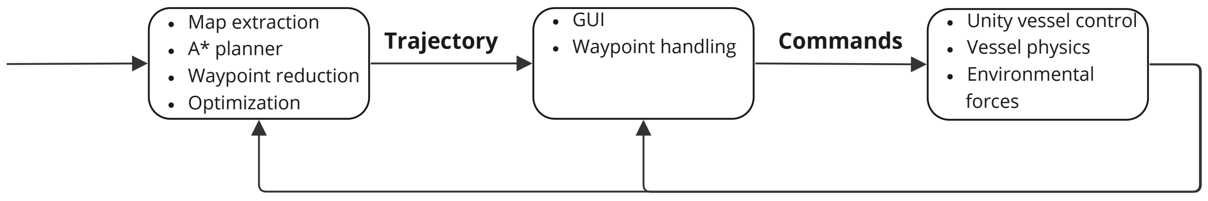
4.2 Development

4.2.1 Step 1: Plan

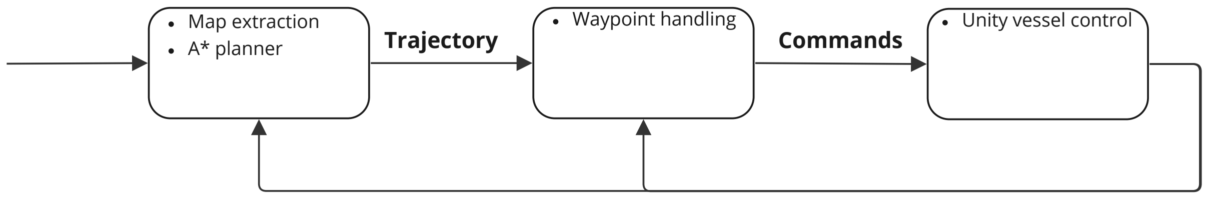
The first step was to map out what modules that are necessary to complete the task. There were some existing packages, but a complete overview of how they performed and their integration status to the rest of the system needed to be determined. Three different flowcharts were made in figure 4.2. The top flowchart (figure 4.2a) is a simplified flowchart showing the different parts of the system. The middle flowchart (figure 4.2b) shows a more detailed overview of which modules belong to what part of the complete system. The last flowchart (figure 4.2c) shows the minimum required to complete the flow of data for the system to work continuously.



(a) Simplified flowchart



(b) Detailed flowchart



(c) Reduced requirements flowchart

Figure 4.2: Upperlevel flowcharts describing the required modules for each part of the system.

4.2.2 Step 2: Coding

With the general plan in the previous step, it is easier to start programming when there is an expectancy for what data each module should produce. The procedure can be broken down into the following steps:

- Sketch up the flow of data
- Describe the process in a few steps
- Create skeleton code based on the bullet points above

- Fill in the skeleton functions
- Get a functioning script and improve code as needed

4.2.3 Step 3: Building/Integrating

When adding new modules and scripts to a system, keeping all the settings in one place is a good idea. This usually means using a config file. It can store initialization variables, which are the settings the system starts with, in yaml files. This way, the code can be kept clean, and changes can easily be changed if needed.

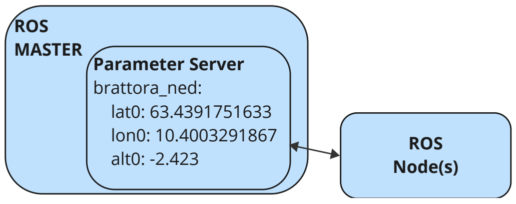


Figure 4.3: ros-param

The modules don't do much good on their own - they need to be integrated into the larger system to be useful. This is especially important when scripts have to run in real-time, as we often need to change parameters while the script is running. Parameters that more than one node uses or that change over time should be loaded into the ROS parameter server. When it needed to change parameters or a server node needs to send new data, services can be used, as shown in figure 4.3. This way, new data or conditions can trigger desired functionality.

4.2.4 Step 4: Test

The system gets tested in the loop with the new modules at this stage and if a test passes, it can move on to the next phase, which is releasing the software to a stage where it gets used to operate the Digital Twin or real vessel.

This part utilizes the performance evaluation tools for each of the metrics that is desired to measure. The main parts relevant for this system is:

- Path measures: length, smoothness, precision and tractability
- Energy usage
- Time usage

To streamline the tests they can be structured using the the flowchart in figure 4.4. When using the Digital Twin vessel in the simulator, the same environmental setup can be utilized to get a better behaviour comparrsion between runs. If the collected data differs to much from the desired bahviour the system should call for adjusting the parameters to se if there is any changes.

4.2.5 Step 5-8

The operate part of the DevOps cycle is not in the scope of this thesis and scripts developed in this theis won't be released in the traditional sense (merging the code to the main part of code stack), it has been prepared to a point where it could be released given further development and the right environment. The parts can be described as:

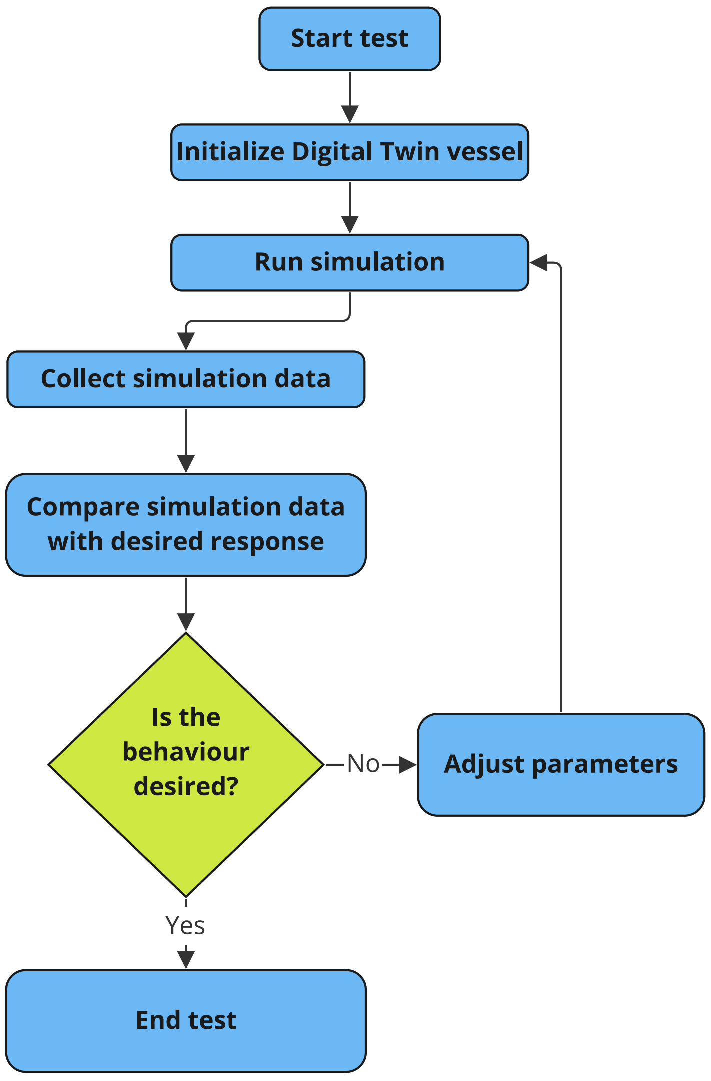

Start test
Initialize Digital Twin vessel
Run simulation
Collect simulation data
Compare simulation data with desired response
Is the behaviour desired?
No
Adjust parameters
Yes
End test

Figure 4.4: Test flowchart

- The Release stage refers to the point at which software is considered ready for deployment.
- The Deploy stage sees the software configured and merged in to the production environment.
- In the Operate stage, the software functions within a live environment.
- Finally, the Monitor stage involves continual surveillance of the software's operation to identify and rectify any issues, and find potential improvements.

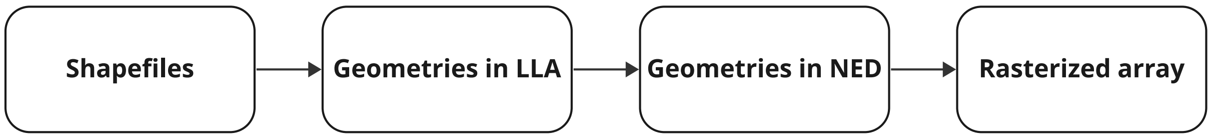


Figure 4.5: Shapefile extraction flowchart

4.3 Path planning

4.3.1 Array extraction from shapefiles

The map extraction from shapefiles follows the process in figure 4.5. The shapefiles get loaded and combined into one GeoDataFrame using python libraries, GeoPandas [10] and Pandas [17]. Within this process, certain geometry types can either be filtered out or included (linestrings, polygons, multipolygons) based on how the different obstacles are represented in the shapefiles.

The geometries get converted to NED, so they can be applied in pathplanning later or be loaded as polygons into a optimization problem as obstacles. For visualization purposes the rasterized map in figure 4.6 is rotated to North-East-Up (NEU) to match the regular view of a map.

When rasterizing the geometries with the features package from rasterio [6], any grid cell that a geometry touch is labeled as occupied, no matter how much or little of it is actually occupied. It is also possible to only return the cells that are fully within the shapes. The returned array will only reflect the real world as close as the resolution allows. And the bounds decide how many meters of that get included in the array.

Extracting the array at different resolutions uses the same amount of time even though the number of cells exponetially grows between the different scenarios 4.1.

Area	Cells per meter	Time taken (s)
$1km^2$	0.1	8.78
$1km^2$	1	8.83
$1km^2$	10	8.84

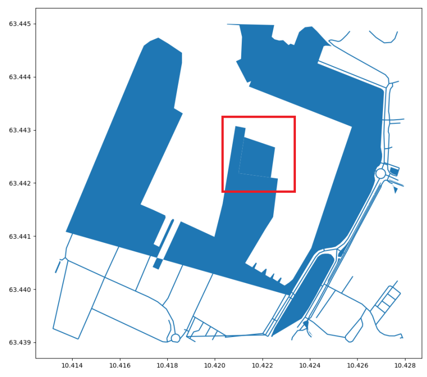
Table 4.1: Processing time for extracting arrays at different resolutions

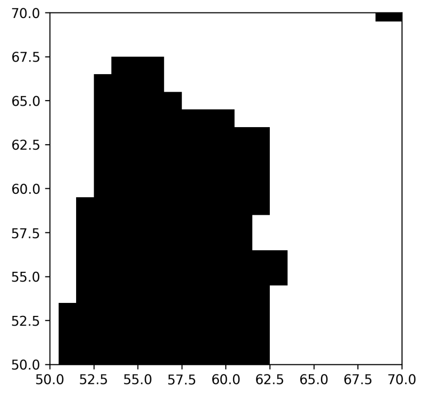
4.3.2 Pathfinding

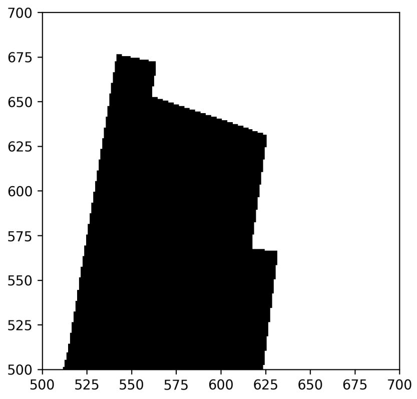
The pathfinding pipeline was initially planned as:

A* algorithm

The A* algorithm searches through the grid between the vessel's current position and the vessel's desired endpoint. Orientation is not taken into account when finding an initial path. However, it takes in an argument, robotsize. This forces the A* to find a feasible path with the size of the robot (or, in this case, vessel) when finding a path. This creates a virtual boundary that requires all nodes within the set radius to be without obstacles.


(a) Plot of GeoDataFrames


(b) Rasterized to 0.1 cells per meter


(c) Rasterized to 1 cells per meter

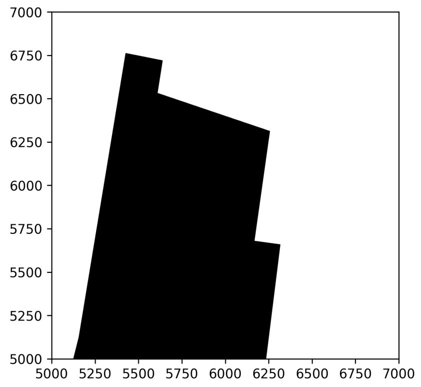

(d) Rasterized to 10 cells per meter

Figure 4.6: Different levels of rasterization. One cell is one unit in NED. The blue area in the upper left subfigure corresponds to the black area in the remaining subfigures. The white area is obstacle free space.

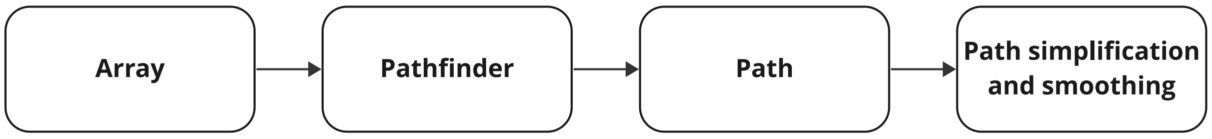


Figure 4.7: Astar and path smoothing flowchart

As the robot size increases, the extra cells that get checked for obstacles increase quadratically. Tabel 4.2 shows the time used as the robot size increases and figure 4.8 gives plots the time taken for each step. The number of extra checked cells results in the equation

$$\text{Extra cells checked} = (2 * \text{robot\_size} + 1)^2 - 1 \tag{4.1}$$

Instead of a robot size, an extra border can be added with a chosen "certainty value" to make the planner check fewer blocks. Regardless of the robot size variable or the extra border addition, an additional step for planning the trajectory closer to the quay is needed. The idea is to make all other points except the endpoints outside the border, and

Robot size	Time taken (s)	Path length
0	0.38	209
1	1.72	211
2	4.21	213
3	8.17	215
4	13.39	217
5	20.70	219
6	28.56	221
7	37.71	223
8	50.47	225
9	63.52	227
10	78.65	229

Table 4.2: Search times for A* with robot radius acting as a safety radius

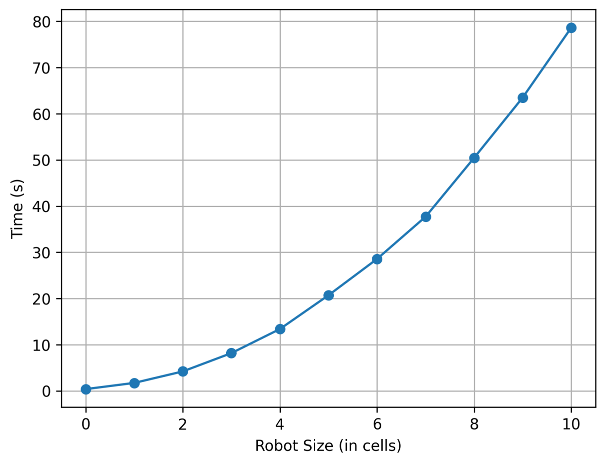
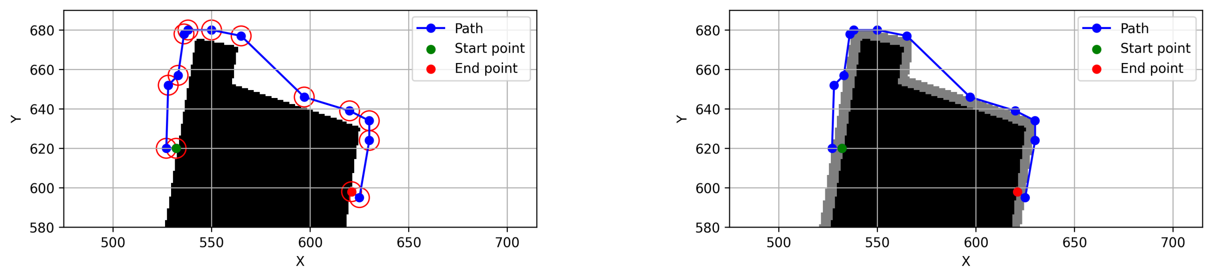


Figure 4.8: Plot of time taken based on robot size

the endpoints can be kept within the border if placed there. A simple and naive method of completing the path when the end points are in an occupied area is to find the nearest unoccupied cell and draw the path between it and the endpoint.



(a) Robot size set to 5 cells **(b)** Border size set to 5 cells

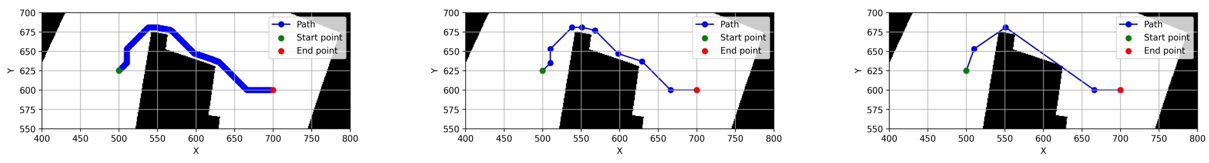
Figure 4.9: Vessel size acting as a safe barrier when finding a path

Border radius	Time taken (s)	Path length
0	0.20	174
1	0.20	174
2	0.27	175
3	0.25	179
4	0.26	186
5	0.26	192
6	0.32	200
7	0.28	212
8	0.30	225
9	0.33	241
10	0.35	258

Table 4.3: Search times for A* with border radius around obstacles instead of robot safety radius

Waypoint reduction

The output from the A* algorithm consists of the nodes it has traversed through in order to find a path between the start and endpoint. Due to the composition of the configuration space, the list of waypoints gets quite large for short distances. Many of the waypoints do not add any extra detail to the path and can be regarded as redundant waypoints. When reducing the number of waypoints with Douglas-Peucker, it is important to select a reasonable threshold. In figure 4.10, it is observable that selecting a threshold close to the resolution of the grid drastically reduces waypoints, while greater thresholds remove to much of the original shape the waypoints created.



(a) $\epsilon = None$ **(b)** $\epsilon = 2$ **(c)** $\epsilon = 10$

Figure 4.10: Waypoint reduction with Douglas-Peucker

4.3.3 GUI

The fact that most of Revolt's codebase interacts with ROS makes the use of various visualization tools within in the framework convinient. One of these tools, ROS visualization (RviZ), has the capability to subscribe to user-defined topics and then display them according to user preferences. This becomes particularly useful when considering that the control system was designed to plan most operations within a unified context, specifically the NED frame. Consequently, this makes the task of presenting live, pertinent data in RviZ much simpler and more straightforward.

In figure 4.12b the pathplanner gets the start pose from the vessel state and finds the path from the vessel to the userdefined end. The desired pose given by the DP-PID controller

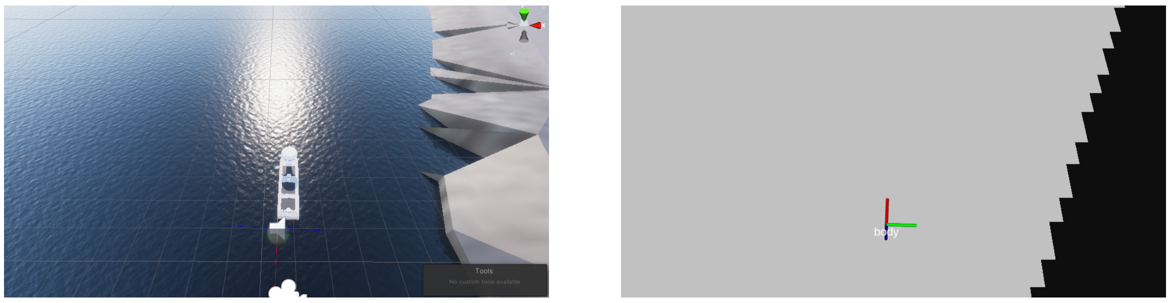
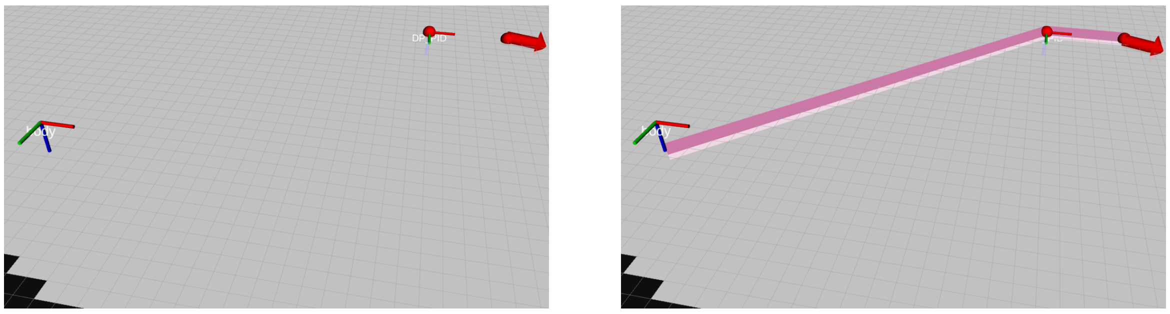


Figure 4.11: ReVolt in Unity (left) and in RviZ (right)

is in addtion shown as a axis marker to show the desired pose.



(a) Without path published (plotted) **(b)** With path published (plotted)

Figure 4.12: Body axis marker is the vessel, DP_PID axis marker is the desired end pose and position for next waypoint. Red spheres are waypoints. Red arrow is the end position and pose for the vessel

4.3.4 Unity

Since Unity is a physics engine it will simulate every unit you enter into the world as often as you tell it to. The simulated inertial measurement unit (IMU) was set to update at a higher frequency than necessary (200Hz), which caused the simulator to struggle. Instead of simulating the IMU at a high rate like this it would be beneficial to base noisy IMU measurements of the ground truth of the vessel in the simulator and add random noise and extra measurements with per simulator ground truth message.

Physics

The physics in in Unity is mainly decided by these components [18]:

- Mass
- Drag
- Angular drag
- Use Gravity

These values are tunable, but hard to match with the physical matrices for the ReVolt vessel. Another option is to instead control the position and pose through the *Is Kinematic* option. This allows for simulating the physics of the vessel outside the Unity game engine. This will change the object collision from unity-generated physics to a bool-based collision warning.

The purpose of the simulator is to get a 3D visualization of the vessel and surroundings while also having the opportunity to gather data from exteroceptive sensors such as lidar and camera. The collision simulation does not have to be as advanced as the game engine might provide.

4.3.5 ROS wrapping

With the scripts written, they need to be integrated to the rest of the system. A good practice is to write the ROS wrapper as compact as possible, since its main purpose is to allow data transfer between the nodes in the system. In the A* node (figure 4.13), the three modules A*, Douglas Peucker and Path smoothing are just three lines, while the rest of the script performs data formatting to standard messages and transformations between frames if needed.

```
path = astar(array, start, goal, border_size, heuristics="euclidean")
reduced_wp_path = simplify_path.main(path, epsilon=1)
smooth_path = smooth_path.main(path)
```

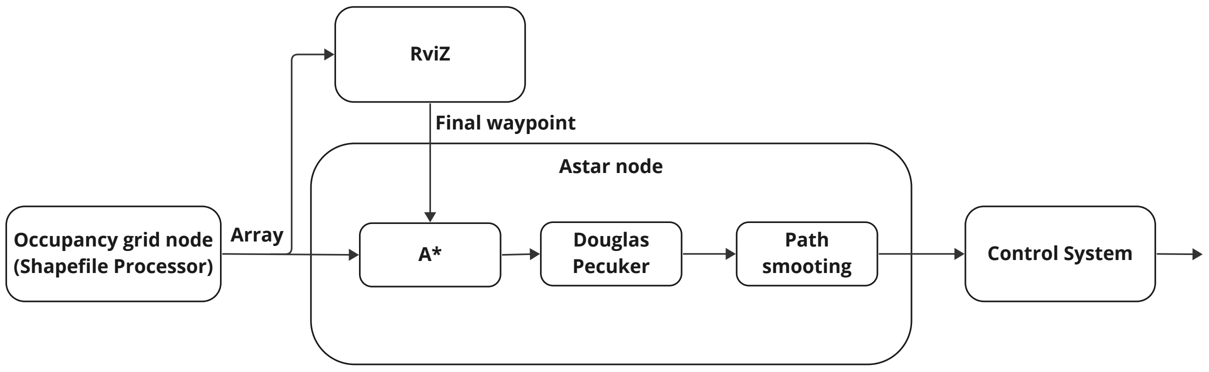
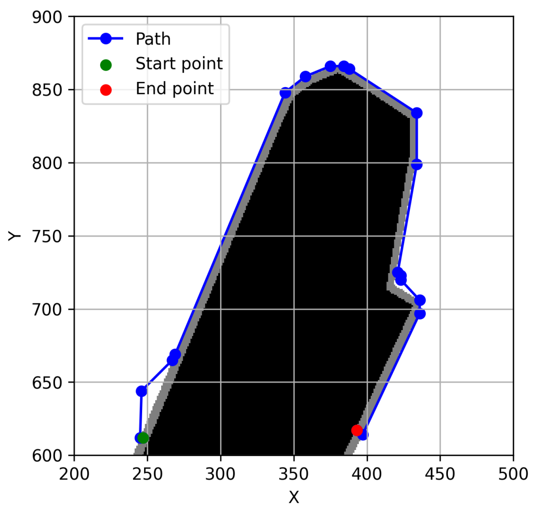


Figure 4.13: Astar and path smoothing flowchart

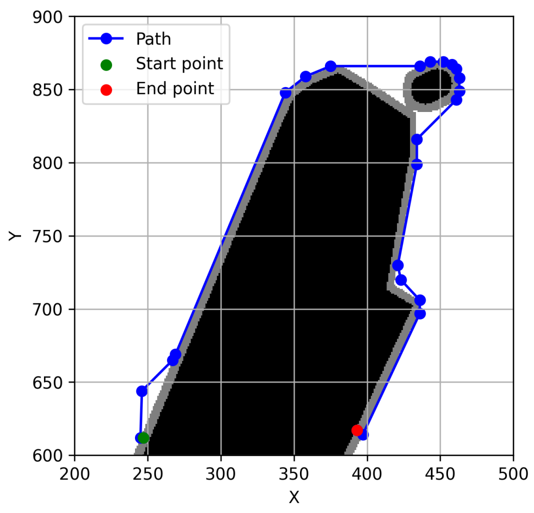
4.4 Replanning

Since there might be new or unknown obstacles along the previously found path, the planner should be able to replan based on data from a collision avoidance (COLAV) module. The path planner can handle replanning queries based on the risk level determined by such a system. An example is static unmapped obstacles as seen in figure ??. These are easier to account for since a predetermined safety radius has been decided.

To reduce computing time and power, the shapefile processor scales resolution and map size based on direct Euclidean distance between current the start and end position. If no path is found by the pathfinder, the map resolution and scale should be changed. If



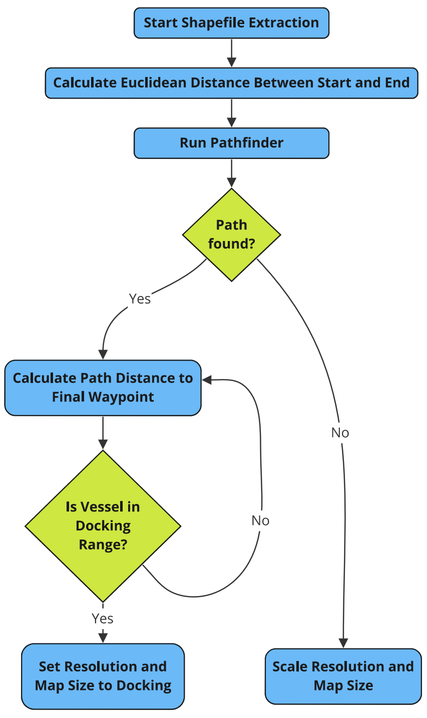
(a) replanning no obstacle



(b) replanning obstacle

Figure 4.14: image in Two horizontal

the path distance to the final waypoint is withing docking range, the map get scaled to account for the greater resolution that is needed. This makes it possible to use a lower-resolution map for larger areas and a higher-resolution map for smaller areas where detail is necessary.



Start Shapefile Extraction
Calculate Euclidean Distance Between Start and End
Run Pathfinder
Path found?
Yes
Calculate Path Distance to Final Waypoint
Is Vessel in Docking Range?
No
Yes
Set Resolution and Map Size to Docking
No
Scale Resolution and Map Size

Figure 4.15: Flowchart for handling size and resolution when extracting geometries

4.4.1 Path optimization

After the path has been found and waypoints have been reduced, it can be further processed in an optimization problem. In Lekkas (2021) [12] a simple kinematic model attempts to follow the path in order to test its feasibility. The same has been done in figure 4.16, but to mimic a vessels slow turn rate, the turning rate was reduced drastically (seen in the defined model in equation 4.2). The solver got a lower threshold for the last waypoint in regard to position and rotation error. Without obstacles stated in the solver it could easily move left of the end waypoint where land was an obstacle. In figure 4.17 this is the case. Without obstacles defined in the optimization problem, the solver can utilize all of the space necessary to achieve the goal.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) * v \\ \sin(\psi) * v \\ \psi * 0.05 \end{bmatrix} \tag{4.2}$$

In figure 4.16 the optimization problem was setup to minimize time and do multiple shooting with Runga-Kutta (RK4):

$$\begin{aligned} \min_{X,U,T} \quad & T \\ \text{subject to} \quad & \\ & \text{v\_min} \leq U(0,:) \leq \text{v\_max} \\ & \text{phi\_min} \leq U(1,:) \leq \text{phi\_max} \\ & x_{goal}^{lb} \leq x_{goal} \leq x_{goal}^{ub} \\ & T \geq 0 \end{aligned} \tag{4.3}$$

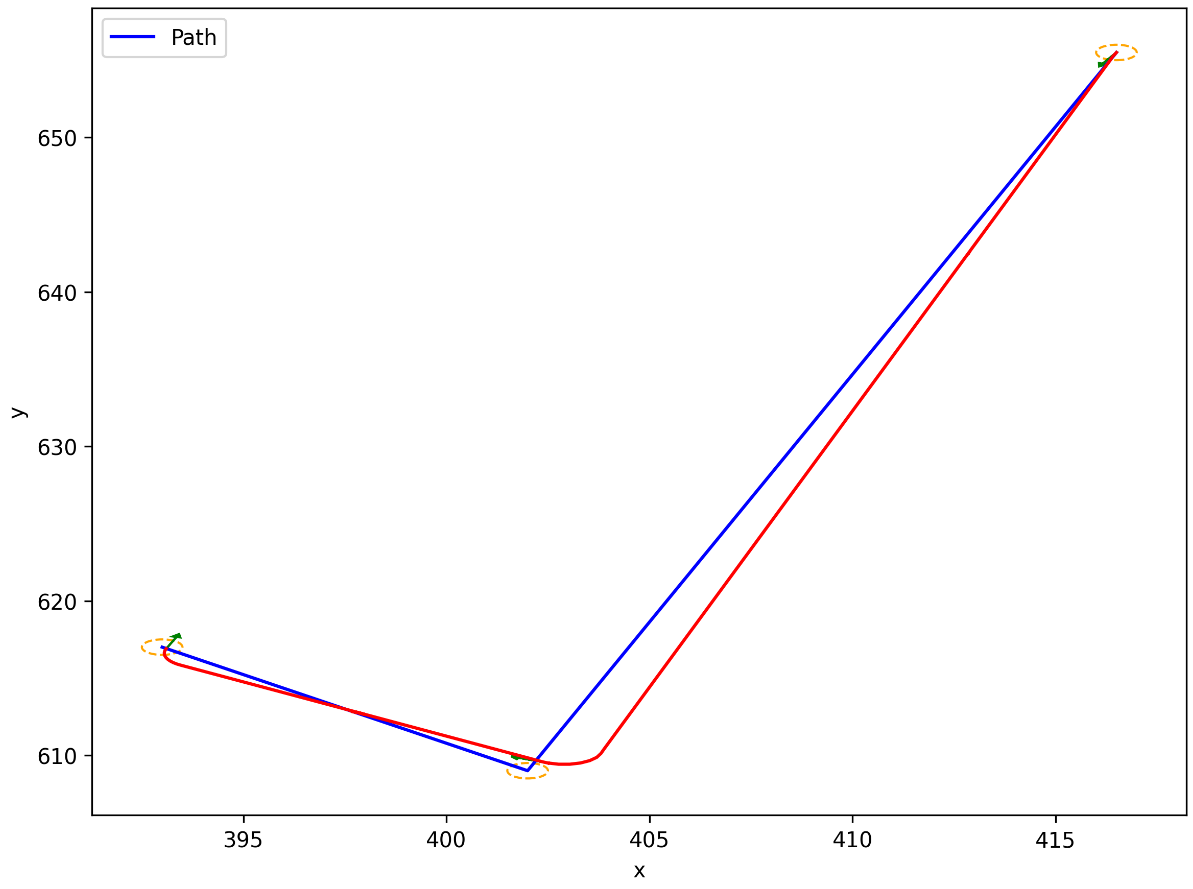


Figure 4.16: Path optimized for time usage with the simple model. Path is feasible with the current waypoints

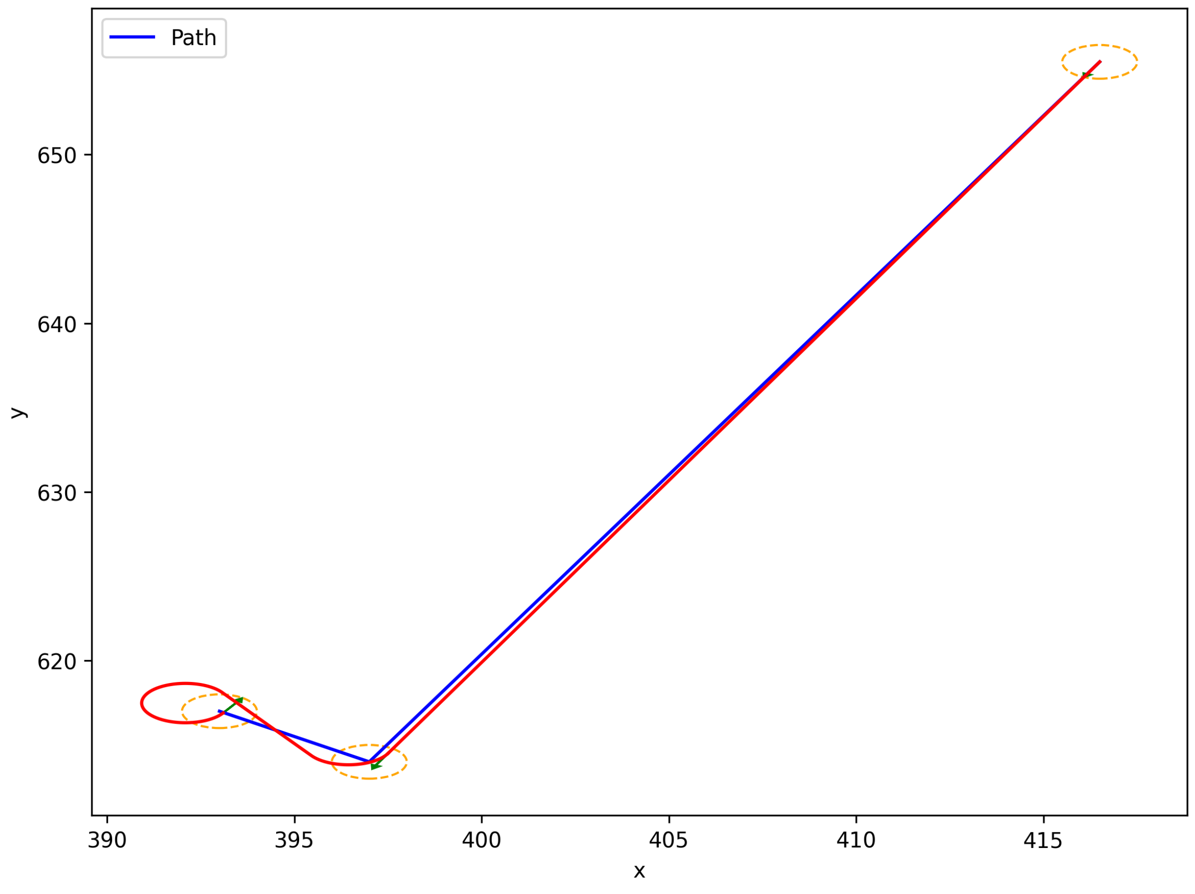


Figure 4.17: Path optimized for time usage with the simple model. The shortest path found by the solver would have gone over land

Chapter 5

Discussion

Since the shapefile extractor is able to use the same NED origin as the rest of the system when converting from GeoDataFrames to NED, the geometries appear at the expected area in relation to the origin coordinates. This makes it possible to extract a rasterized array fairly quickly when a reasonable resolution is selected for the size of the map. In addition, only the relevant area and its included obstacles get rasterized and made into an array, which limits the amount of area that needs to be rasterized.

The safety radius around the vessel and the A* algorithm combined created a slow and ineffective method for searching the space for a feasible path between the start and final waypoint. Even with grid cells in the array that are quite large (e.g. 10 meters per cell), it is a feature that scales poorly. Using a border around obstacles creates a path much quicker as the number of cells it needs to search through is much lower.

The selection of a reasonable epsilon when using Douglas Peucker 4.3.2 is not trivial, especially for environments where it is important to keep the initial shape of the initial path. A robust method of selecting the epsilon is to base it on the resolution of the array produced by the shapefile processor and the border added to the obstacle. How much the reduced path will deviate from the initial one will then be known due to the cell size.

Since the resolution of the array is selected based on the distance from the vessel to the final waypoint, it will limit the size of the array that is created. This has a potential fallback since the selected waypoint might be valid before rasterization, but due to the way cells get determined whether they are occupied or not, the system might find it impossible to find a feasible path to the final waypoint. A simple and naive method for completing the path is to find the nearest unoccupied cell and draw a straight line between it and the endpoint. To ensure that movement near the quay is safe, exteroceptive data from lidar and camera would be required.

Without having had the time to define obstacles in the optimization problem, the solver can find a path in undesired areas. With the geometries extracted to the NED frame in the previous steps, they could be implemented as constraint for the solver, so that it has to find another solution to the problem. The solution would then have to avoid the geometry constraints.

When developing new features and modules for an existing large system, one is essentially creating a black box that processes data. If a script is time-consuming and hard to understand it will most likely reduce the chance of it beeing used or maintained in the future. This is why splitting the data processing code and ROS-relevant code can be especially important to make it more appealing to maintain and use the scripts.

This is really a highlight of the negative sides of ROS's structure (nodes and topics) in larger projects can be seen in the data sent over the topics. The nodes are essentially black boxes, and the data has to be trusted until you're getting close to finishing your own node that will base its measurements or further processing on the received data.

Chapter 6

Conclusion

This thesis has proposed using rasterized map data for navigation and path planning in the quay area to find a feasible path from the vessel's position and to the desired end position. The extraction process takes the same time regardless of resolution, which shows that the extraction time is limited by the utilized packages.

With the GUI functionality for showing the extracted map, found path, and the ability to set end pose in RviZ. It adds functionality that is easy to utilize even with little familiarity with ROS. Since the Revolt codebase revolves around ROS it also limits the spread in different user interfaces and platforms that new developers or students need to familiarize themselves with.

The methodology for this project fits with the DevOps approach for software development, where the principles of continuous integration and iterative improvement are important. While the initial four stages of the DevOps process loop: Plan, Code, Build, and Test, have been accounted for, the full implementation of the other stages: Release, Deploy, Operate, and Monitor, is beyond the practical scope of this thesis due to time constraints.

6.1 Recommended further work

The performance of the map extraction process, while consistent across different resolutions, could be improved in terms of speed and efficiency. The bottleneck appears to be with the utilized packages. Therefore, an investigation into more performant alternatives or optimizing the current ones could be explored.

The optimization problem should be solved by using the ReVolt vessels system matrices as dynamics and the thruster configuration. This implementation can aid in solving for a path and thruster input that gives energy efficiency, a better-suited path for the vessel or minimize the control input required. In addition, a method for defining geometries as constraints in the optimization problem is needed to limit the system from finding infeasible paths.

To improve the evaluation of the vessel's performance, better measurable evaluation criteria should be developed. These should include measures of energy use and the quality of the paths it generates. Also, it should consider factors such as the precision of docking, how the vessel reacts to different environmental conditions, and its compliance with safety rules. This would give a more thorough understanding of how the vessel is performing.

Finally, the application could be extended beyond the quay area to other parts of the maritime transportation system. A broader scope might necessitate addressing challenges such as handling larger and more detailed maps, dealing with dynamic obstacles, and managing longer distances and more complex routes. This would require further investigation and development of the underlying algorithms.

References

[1] Henrik Alfheim and Kjetil Muggerud. *Development of a Dynamic Positioning System for the ReVolt Model Ship*. eng. 2017. URL: http://hdl.handle.net/11250/2452115.

[2] Glenn Bitar, Vegard N Vestad, Anastasios M Lekkas, and Morten Breivik. "Warm-started optimized trajectory planning for ASVs". In: *IFAC-PapersOnLine* 52.21 (2019), pp. 308–314.

[3] David H Douglas and Thomas K Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature". eng. In: *Classics in Cartography*. Chichester, UK: John Wiley & Sons, Ltd, 2011, pp. 15–28. ISBN: 0470681748.

[4] Sigurd Øygarden Flæten. *Dette skipet er utslippsfritt og har ingen mennesker ombord*. (Visited: 14. Desember 2022). URL: https://www.tu.no/artikler/dette-skipet-er-utslippsfritt-og-har-ingen-mennesker-ombord/231695.

[5] T.I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley-Blackwell, 2021.

[6] Sean Gillies et al. *Rasterio: geospatial raster I/O for Python programmers*. Mapbox, 2013–. URL: https://github.com/rasterio/rasterio.

[7] Yewen Gu, Julio Cesar Goez, Mario Guajardo, and Stein W Wallace. "Autonomous vessels: state of the art and potential opportunities in logistics". In: *International Transactions in Operational Research* 28.4 (2021), pp. 1706–1739.

[8] Yewen Gu and Stein W. Wallace. "Operational benefits of autonomous vessels in logistics—A case of autonomous water-taxis in Bergen". In: *Transportation Research Part E: Logistics and Transportation Review* 154 (2021), p. 102456. ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2021.102456. URL: https://www.sciencedirect.com/science/article/pii/S1366554521002209.

[9] C Jallal. "Rolls-Royce and Finferries demonstrate world's first fully autonomous ferry". In: *Maritime Digitalisation & Communications* (2018).

[10] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, Geir Arne Hjelle, Alan D. Snow, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Nick Eubank, maxalbert, Aleksey Bilogur, Sergio Rey, Christopher Ren, Dani Arribas-Bel, Leah Wasser, Levi John Wolf, Martin Journois, Joshua Wilson, Adam Greenhall, Chris Holdgraf, Filipe, and François Leblanc. *geopandas/geopandas: v0.8.1*. Version v0.8.1. July 2020. DOI: 10.5281/zenodo.3946761. URL: https://doi.org/10.5281/zenodo.3946761.

[11] S Kongsberg. *Autonomous ship project, key facts about YARA Birkeland.* 2017.

[12] Andreas Bell Martinsen, Anastasios Lekkas, and Sebastien Gros. "Optimal Model-Based Trajectory Planning With Static Polygonal Constraints". In: *IEEE Transactions on Control Systems Technology* PP (July 2021), pp. 1–12. DOI: 10.1109/TCST.2021.3094617.

[13] Society of Naval Architects, Marine Engineers (U.S.). Technical, and Research Committee. Hydrodynamics Subcommittee. *Nomenclature for Treating the Motion of a Submerged Body Through a Fluid: Report of the American Towing Tank Conference.* Technical and research bulletin. Society of Naval Architects and Marine Engineers, 1950. URL: https://books.google.com.mx/books?id=sZ%5C_bOwAACAAJ.

[14] Simen Sem Øvereng. *Dynamic Positioning using Deep Reinforcement Learning.* 2020.

[15] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Longman Publishing Co., Inc., 1984.

[16] *Red Hat - DevOps.* (Visited: 20. May 2023). URL: https://internship.technovalley.co.in/features/redhat-devops-consulting-kochi.html?fbclid=IwAR3v2CJsLxdA2RQlxpbP

[17] The pandas development team. *pandas-dev/pandas: Pandas.* Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[18] Unity. *Rigidbody component reference.* (Visited: 03. May 2023). URL: https://docs.unity3d.com/Manual/class-Rigidbody.html.

[19] Kjetil Vasstein. *A high fidelity digital twin framework for testing exteroceptive perception of autonomous vessels.* eng. 2021. URL: https://hdl.handle.net/11250/2781031.