Autonomous Maneuvering of Waterborne Vehicles in Close Proximity of Obstacles

Wouter van Unen

Delft University of Technology

Faculty of Mechanical, Maritime and Material Engineering

Department of Cognitive Robotics

Autonomous Maneuvering of a Waterborne Vehicle in Close Proximity of Obstacles

Master Thesis Report

report number: 2020.COR.8456

Author:

Wouter van Unen

Chair:

Prof. Rudy Negenborn

Daily supervisor:

Dr. Ali Haseltalab

Daily supervisor:

Dr. Javier Alonso Mora

Company supervisor:

David Woudenberg

October 1, 2020

Abstract

In the automotive industry, automation is on the rise and it increases safety while decreasing costs. Improved sensor performance and greater computing power steers the future of the shipping industry in the same direction. Avoiding obstacles in close proximity is one of the current challenges. This research provides a generic and open-source design guideline for mapping, path planning and control for autonomous sailing in an environment with obstacles in close proximity. For each of the three modules, several options are discussed and the best one is chosen.

Mapping is applied in the form of mapping with positioning sensors in an occupancy grid map. The original map is devided into cells containing information about the probability that they represent an obstacle. The mapping algorithm is applied in several maps and with noise and compared with a simultanously localization and mapping (SLAM) method. The resulting maps are created within the path planning requirements.

Path planning is performed by inflating the occupancy grid map from the mapping algorithm. The inflated map is sampled into an 8-connected grid matching the grid cell size. A Dijkstra algorithm is applied combined with an quadratic approximation in the costmap. A steepest descent method will return a continous and shortest path which is suitable for control and does not collide with the obstacles in the map.

Proportional and integral is applied to steer the vessel along the path. The proportional controller is tuned based upon the look ahead distance and its maximum error with the original path. The controller is tuned using a block shaped path as a reference and evaluated in a path from the path planner. Both static errors like wind or current and Gaussian noise on the position estimation are included for validation. Finally the controller is able to maneuver the vessel safely from its starting point to its final destination

The functionalities of the modules are indivividually demonstrated, and eventually interconnected into one system. Autonomous maneuvering of the complete system is then demonstrated in various simulated environments and settings, including unknown areas and global planning challenges. The vessel is able to maneuver autonomously from the start to its goal given only the sensor data, its location and its goal position.

Acknowledgement

I am very grateful to Prof Rudy Negenborn and Dr. Javier Alonso-Mora for their support, inspirations, thoughts and insight about the topic. Their great knowledge on maritime robotics inspirated me on the work created, and without their time, resources and knowledge it would not have been possible to fullfil this research. Special thanks goes to Dr. Ali Haseltalab and MSc Vittorio Garofano for creating a comfortable and inspiring working environment including plenty productive conversations.

I would also like to thanks Xomnia for their social and material support. The Xomnia office has become a second home for me during the first part of the project showing how motivating it can be to work with the right people. I would especially thank David Woudenberg for our weekly meetings and good communication between me and Xomnia about the numerous possibilies within a small but great company.

I would like to thank my friend Jan and brother Arjan, for their presence as sparring partners and also as great motivators, Their presence was one of the key factors te keep the work and attitude going.

Last but certainly not least, I would like to thank my beloved parents, younger brother and my girlfriend Estella for their great and everlasting support.

Contents

Acronyms

BFS Breadth first search. 36, 37, 38, 39, 42

DOF Degree of Freedom. 10, 9, 18, 20, 21, 22, 47

EKF Extended Kalman Fiter. 26, 27, 28

EMR Electromagnetic Radiation. 18

FBM Feature Based Map. 16, 32

GBFS Greedy best first search. 36, 37, 39, 42

GNSS Global Navigation Satellite System. 15, 20, 21, 22, 32, 58

GNT Gap Navigation Tree. 16, 32

GPS Global Positioning System. 21, 31, 61

GVD Generalized Voronoi Diagram. 35

IMU Inerial Measurement Unit. 20, 21, 22, 31, 32, 58, 61

LiDAR Light Detection And Ranging. 9, 15, 19, 22, 32, 61, 65

MPC Model Predictive Control. 56, 64

OGM Occupancy Grid Map. 16, 32, 34, 35

PF path following. 46

PRM Probabilistic Roadmap. 35, 36, 44

radar Radio Detection And Ranging. 15

RMS Root Mean Square. 28, 29, 30, 46, 51

RRT Rapidly exploring Random Tree. 35

RTK Real-time Kinematics. 21

SLAM Simultaneous Localization and Mapping. 15, 16, 25, 26, 27, 28, 30, 31, 32, 63, 64

SS Space Skeletonization. 35

USV Unmanned Surface Vessel. 9, 10, 11, 12, 13, 15, 17, 19, 20, 21, 22, 25, 28, 34, 35, 40, 43, 44, 46, 47, 48, 49, 50, 51, 52, 56, 58, 59, 60, 61, 62, 63, 64

Nomenclature

greek symbols	
η	position of USV in map frame
ν	velocity of USV in USV frame
ψ	yaw rate
σ	uncertainty
τ	USV input
arabic symbols	
C	resistance matrix
C_a	added resistance matrix
C_{rb}	rigid body resistance matrix
D	damping matrix
D_l	linear damping matrix
D_{nl}	non-linear damping matrix
E	disturbance
F_x	longitudinal force applied on the USV
F_y	lateral force applied on the USV
R	rotation matrix
M	mass matrix
M_a	added mass matrix
M_{rb}	rigid body mass matrix
P	proportional gain
I	integral gain
T_z	torque applied on the USV
d	distance
e	error
e_x	longitudinal error
e_y	lateral error
e_ψ	rotational error
h	grid map resolution
r	lotational velocity
u	longitudinal velocity in USV frame
v	lateral velocity in USV frame
x	x coordinate of USV in map frame
y	y coordinate of USV in map frame

Chapter 1

Introduction

1.1 Problem statement

Maritime accidents are proven to be harmful to human nature. Well known accidents are the Titanic [1], which caused over 1500 deaths, and the ship collision with the Atlantic Empress [2], which spoiled 280.000 tonnes of oil in the Caribbean. Today still 70% of the shipping accidents is partly caused by human error [3]. In order to reduce the impact of maritime accidents, either humans have to increase their performance by adding assistance systems, or humans must be taken out of the loop. In the latter case a system which outperforms a human should take over control. Removing humans from the loop has multiple beneficial effects besides a major safety increase. First it reduces operational cost significantly, because less crew members are required to sail the ship. Crew costs account on average for 48% [4] of the total operational expenses of a ship. Second it avoids problems caused by future employee shortage. The predicted upcoming shortage is 30.000 crew members worldwide. Third it can reduce the fuel consumption, because an highly automated system is more predictable which can lead to tighter time schedules and lower cruise speed.

Car companies have been working on developing advanced driver assistance systems (ADAS) and semi autonomous vehicles for years now. Today‘s high end car models include features like adaptive cruise control and Tesla even includes a functional and highly used highway autopilot in their models [5]. It is expected that this trend will remain and will finally result in full automation of all road vehicles. Following the development of autonomous cars, ships are being automated as well. In order to make a ship fully autonomous, the ship has to be able to perform all maneuvering on its own. Several systems for partly autonomous maneuvering in the open sees are already available like coarse control [6] or automated coarse planners based on ship data and weather forecast [7].

Open water usually does not contain obstacles and the other vessels can be assumed to send out their position. Therefore no extra sensors are required for environment detection. In inland waters with close proximity of obstacles the situation changes completely. The surrounding cannot be assumed to be free of obstacles and small errors in position estimation might result into large safety risks. Extra sensors have to be implemented for detecting obstacles and increasing the accuracy of the position estimation. Several projects which are currently working on automation in maritime application are described in Section 1.2.

1.2 Related work

Simular research has been performed and is still being performed on autonomous maneuvering of waterborne vehicles. Projects differ from model scale robotic application up till full size vessels. Research oriented projects are performed to discover new methods and applying new principles for autonomous maneuvering. On the other hand, some renowned manufacturers are close to releasing semi-autonomous features for application in real-size vessels.

An example of such a semi-autonomous feature for real-size vessels is Volvo. Ship manufacturer Volvo Penta recently released a video of a self docking system for yachts [8]. The system is completely autonomous with high positional accuracy. However it requires sensor infrastructure at the berth for precise measurements. This means that the system only works for prepared docking spaces equipped with Volvo docking equipment, which is a major disadvantage of the flexibility of the system. Therefor such a system is hard to implement in a fully autonomous vessel. Other project aiming on autonomously maneuvering of full-size vessel are the Norwegian projects Hull2Hull [9] and Yara Birkeland [10]. The Hull2hull project goal is to realize save navigation in close proximity of other vessels and objects. Yara Birkeland is a fully electric and autonomous containership for use in the Norwegian fjords.

Captain AI [11] is cooperating with Port of Rotterdam in order to develop a fully autonomous vessel based on an old harbor vessel. Captain AI uses six cameras in order to get a 360 degree view around the vessel. The ships location is acquired via GPS and environmental conditions are obtained using windspeed sensors and realtime current information measured externally in the port. Currently Captain AI is able to recognize and avoid objects. They have recently fullfilled an autonomous tour in the harbor, but their main issue is to recognise object in close proximity of the vessel due to a big blind spot. Xomnia, an artificial intelligence based company has equipped their ship ,the 'MS-X-AI'[12], with cameras for object detection. The ship can sail autonomously on the Amsterdam canals using an end-to-end approach as used by Nvidia [13]. This means that the boat converts the camera images directly into propeller and rudder input using a deep neural network. The method is proven to perform only in limited condition of low traffic and wide canals.

Some other project are focussing on designing model size vessels for research on autonomous sailing. A well known example is Roboat. Roboat [14] aims for creating a completely autonomous system without required adaptations to the environment. AMS and MIT joined forces to create the Roboat in cooperation with Delft University of Technology. Roboat's aim is to use the Amsterdam waterways to create a multi-agent decentralised floating infrastructure which can cooperate to create temporary bridges or stages but can also commute people or transport goods in the Amsterdam canals. Roboat is using 3D lidar pointclouds and camera images to recognize its environment. The Roboat project focuses on designing a complete new infrastructure rather than upgrading an existing vessel. For this reason their bumping into other challenges than only those one would except when testing autonomous sailing. Therefore the intended release date for the infrastructure is not in the near future. Delft University of Technology is working on autonomous transport vessels. Their Delfia is used for distributed configurations for shipping containers [15]. Delfias are small robots carrying one container at a time which can group together to form one big ship when crossing main waterways. Another project is the Grey Seabax focussing on detecting landmarks for navigation. Due to the numerous state-of-the-art projects on autonomous maritime applications and the availability of knowledge and materials Delft University of Technology is a great place for further research in this field.

As shown above, several projects are being performed on autonomous maritime maneuvering, each have their own shortcomings. However, no generic method is available for designing an autonomous vessel by the knowledge of the author. This research presents a generic guideline for designing autonomous vessels for close proximity applications using open source software. Xomnia has provided their ship for further research on autonomous sailing and Delft University of Technology provides theoretical and pracical knowledge in the field. The Xomnia ship turned out to be unreliable and fragile and therefore is decided to perform only theoretical research. The research is performed based on the maneuvering ship model of the Delfia vessel which is provided by TU Delft. The method is underpinned by showing a working simulation where a vessel is able to safely reach a goal in an unknown environment without colliding into obstacles.

1.3 Research question

The research is lead by answering the following research question:

What navigation and control methods are required to maneuver a boat autonomously in urban waterways?

In order to answer this question, a general method for autonomous maneuvering is applied. This method consist of first mapping, then path planning and finally control. The different parts of the research question are split and explained in following subquestions:

1.3.1 mapping

To begin with, a map is created based on the available sensor data. This mapping part is linked to the first subquestion of this research

What mapping techniques are available and which technique suits best for our application?

In order to answer this subquestion, Two questions need to be answered. Why is mapping desired? And how is mapping performed? These questions are answered in Chapter 3. First a choice is made in what type of map is used. Next the type of sensor is chosen. When the sensor input and map output are defined, several mapping algorithm can be reviewed and evaluated based on their key performance indicators, which will be defined in the chapter. The two best options are implemented and the best algorithm is chosen for the final setup.

1.3.2 Path planning

The path planning module plans a path in the map created by the mapping module. This path represents the desired path for the vessel to reach its goal. In order to implement a good path planning algorithm the following subquestion has to answered:

What path planning techniques are available and which technique suits best for our application?

The question is answered by answering the following questions: Why is path planning desired? And how is path planning performed? These questions are answered in Chapter 4. At first a choice is made whether to include timestamps in the path. Next path planning is defined and several methods are evaluated. The most promising methods are implemented in a simulation with the model of the vessel and finally the best path planning algorithm is chosen for implementation in the final system.

1.3.3 Control

Control refers to path following in this case. Path following is the act of following the path as defined by the path planner. The path is followed by sending signals to the actuators defining their propulsion forces or torques. In order to find a suitable control method the following question has to be answered:

> *What maneuvering control techniques are available and which technique suits best for our application?*

The question is answered in Chapter 5. First, different definitions of path following control are explained. After choosing the right control input, A proportional and an integral control method are implemented, tuned, and evaluated. The tuned controller as found in this chapter is implemented in the final system.

1.3.4 Connecting the modules

Finally, the three modules as defined above are connected into the final system. For evaluation of the final system, the following subquestios has to be answered:

> *Is the final system capable of maneuvering a vessel autonomously?*

For answering this question the final system is implemented in several different situations. Different maps and starting position result in different performance. The system will be evaluated visually as no metric for such a system have been found. The metrics of the separate modules are used as metrics for the evaluation of the final system.

1.4 Contributions

This research has proven that it is possible to make a vessel sail autonomously to a given goal using mapping with positioning sensors, Dijkstra path planning with gradient descent in a quadratically approximated costmap and proportional integral control in simulation. By the authors knowledge, This combination has never been performed before in autonomous sailing combined with the Delfia ship model. All simulation results are performed within the defined bounds for the performance indicators. Additionally, this research provides a generic guideline for autonomous maneuvering in maritime applications. The guideline is generic in the sense that it applies to any ship that fullfills the sensor setup. The grid cell size and the safety margins for path planning have to defined. Tuning the controller is mostly automated by applying a grid search method while optimizing for minimal error. Propiary software is avoided in order to make this repeatable without the need of acces to expensive software packages. Only open source software languages like C^{++}, *Python*, and *ROS* are used. Numerous mapping, path planning and control methods have been reviewed in this research. Several have been applied in the simulation and with that another validation step is taken in the validation of the previous researches.

1.5 Structure of the report

Every subsection is explained in detail in a separate chapter. The remainder of the report is organized as follows: The explanation of the complete system and its contained modules is found in Chapter 2 together with background information about the modeling techniques. The modules mapping, path planning and control are described in Chapters 3 to 5 respectively. For each module the performed task is further described. Several methods are proposed to perform this task and the best method is chosen based on performance in simulation. Chapter 6 combines the best seperate methods into a complete system and evaluates and discusses the results. Chapter 7 finalizes the report by drawing a conclusion list, the contributions and recommend future work.

Chapter 2

Modeling

This chapter describes the system layout schematically. Modeling assumptions are listed and finally the dynamic Unmanned Surface Vessel (USV) model is pointed out.

2.1 System layout

The system as designed for this thesis consists of three main modules: mapping, path planning and control. Chapters 3 to 5 all elaborate on one of the modules. The modules are connected as shown in Figure 2.1. The first module is mapping and requires sensor data from the Light Detection And Ranging (LiDAR)and the positioning sensor for producing a map of the environment. Using this map , the Unmanned Surface Vessel (USV) position and a goal reference, the path planning module generates a path to follow. This path is followed by the control module requiring the USV position. The control module will calculate and return the actuator inputs τ for the USV.

The simulations in this research are performed using Robot Operating System (ROS) Melodic. ROS allows to create modules called *'nodes'* and let them communicate over *'topics'*. The nodes for this research are written in Python 3.6.9. The data is visualized using RVIZ.

2.2 Ship maneuvering model

This research is performed using a model of an USV in a simulated environment. A model is by definition an approximation of the real world. A perfect model would equal the real world, however no perfect simulated model is available. Every model has different benefits and drawback and therefor it is important to choose the right model to obtain a fair approximation of the real world for the specified application. This research uses the Delfia model [16] of which a picture is shown in Figure 2.2. The models associated assumption as defined below:

1. **No severe external disturbances** like waves, wind, current and other environmental variables are assumed. Therefor the outcome of this research cannot be guaranteed to be applicable anywhere else

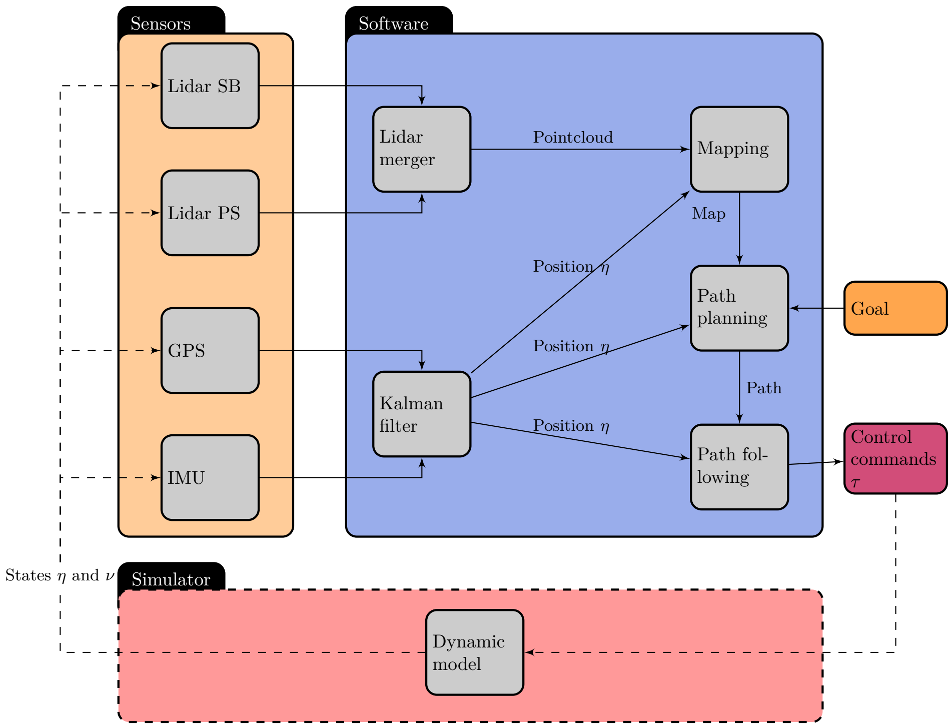


Figure 2.1: Connections between the modules.

rather than flat water. Chapter 5 describes some disturbance rejection, taking into account some small disturbances.

2. **3DOF** are taken into account. Only motion in the horizontal plane is considered. A real USV will move in 6 DOF in the directions shown in Figure 2.3. Pitch, roll and heave motion are caused by waves, wind and acceleration forces which are absent or considered not significant in the scope of this research. Only 3 DOF remain, namely: surge, sway and yaw motion corresponding to displacement in x_{usv}, y_{usv} and ψ respectively.

3. **Holonomic control** can be applied, meaning that every DOF can be controlled independently. An example of an holonomic vehicle is a train, which can only move in surge direction which is controlled. A non-holonomic vehicle is a vehicle which can move in more directions than the number of actuators available. The best known example is a car, which moves in a 3 DOF area with only 2 DOF control, namely steering and throttle.

A Dynamic model is called dynamic because the states of the USV are used for calculating its motion. This means that the motion depends on the velocity and acceleration of the USV. The definition of the state variables is depicted in Figure 2.4. Parameters x and y represent the coordinates of the location of the USV in the fixed map frame as shown in Figure 2.4a. ψ represents the yaw angle (or heading) of the USV relative to the fixed map frame. The USV frame arises by defining a new cartesian coordinate system with its origin at the position of the boat. This frame will later be useful for processing sensor data and defining navigation paths. A point in the USV frame is defined by a x_{usv} and y_{usv} coordinate. As Figure 2.4b shows, velocities are defined similarily but instead $\dot{x}$, $\dot{y}$ and r are used to define the velocities in the map frame. Velocities in the USV frame are noted u and v or also referred to as longitudinal and lateral velocity respectively.

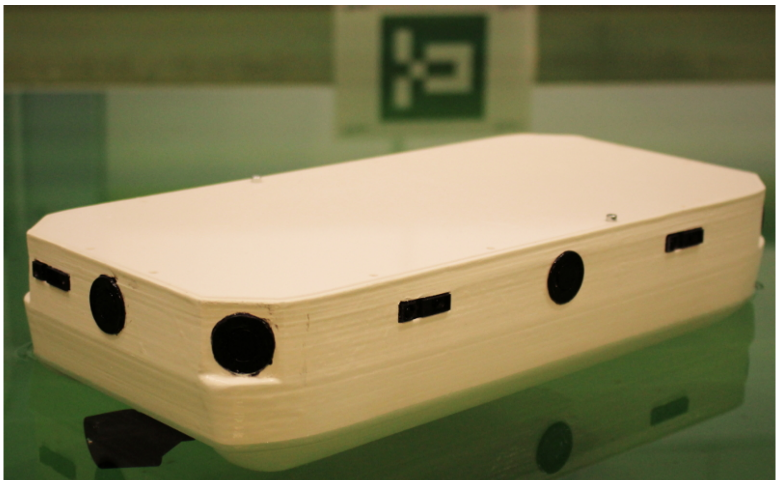


Figure 2.2: Delfia

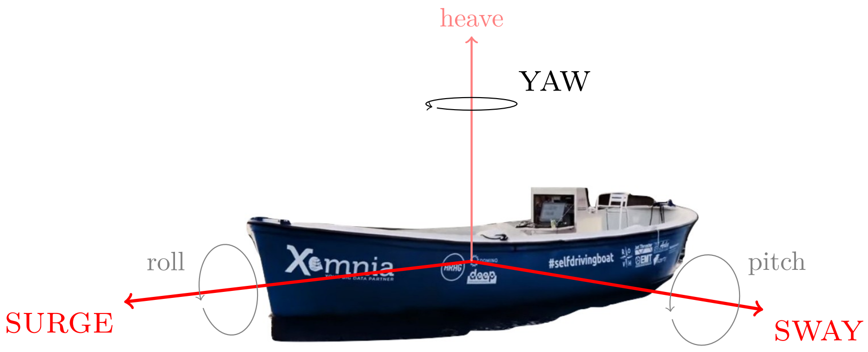


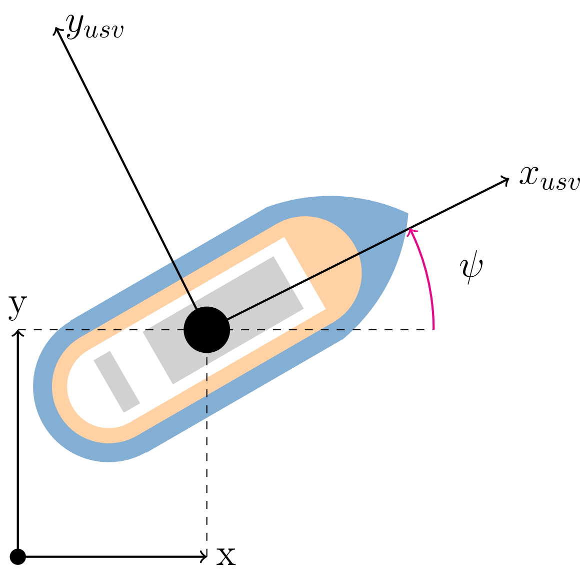
Figure 2.3: 6 DOF motions. The capatilized motions are considered for the 3DOFcoordinate system. The red motions are translational motions and the black motions are rotational

The location of the USV is defined by a x, y and ψ coordinate which could be grouped into a position state vector $\boldsymbol{\eta}$ represented in the map frame. The velocity state vector $\boldsymbol{\nu}$ consists of u, v and r and can be represented similarly

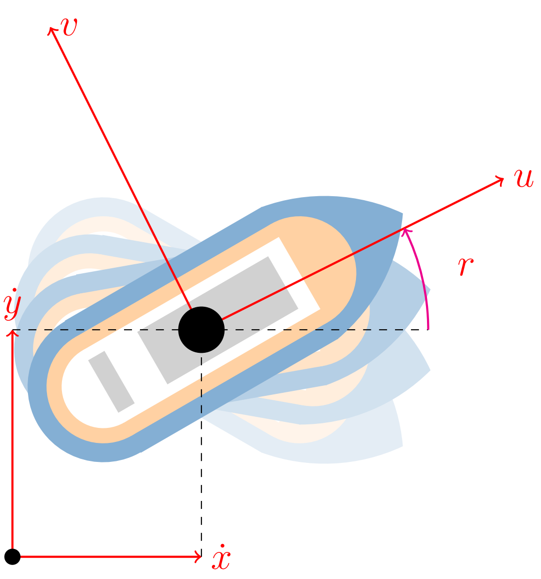
$$\boldsymbol{\eta} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \quad (2.1) \qquad \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ r \end{bmatrix}. \quad (2.2)$$

The translation between the map frame and USV frame can be neglected as the velocity is independent of the position of the USV. The relation between $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ can therefor simply be described by a rotation and a differentation

$$\boldsymbol{\nu} = \mathrm{R}(\boldsymbol{\eta})\dot{\boldsymbol{\eta}} \tag{2.3}$$



(a) position



(b) velocity

Figure 2.4: Definition of (a)location and (b)velocity of the USV

where **R** equals the rotation matrix for the current yaw angle ψ

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

The external forces applied on the USV are denoted by

$$\boldsymbol{\tau} = \begin{bmatrix} F_x \\ F_y \\ T_z \end{bmatrix} \tag{2.5}$$

where the vector entries F_x, F_y and T_z represent the applied force in x_{usv}, y_{usv} and ψ direction respectively applied in the center of rotation of the USV

Having determined the states and external forces of the USV, the actual model can be defined

$$\mathrm{M}\dot{\boldsymbol{\nu}} + \mathrm{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathrm{D}\boldsymbol{\nu} = \boldsymbol{\tau}. \tag{2.6}$$

The first term $\mathrm{M}\dot{\boldsymbol{\nu}}$ represents the resistance due to inertial forces where **M** equals

$$\mathrm{M} = \mathrm{M}_{\boldsymbol{rb}} + \mathrm{M}_{\boldsymbol{a}} \tag{2.7}$$

and $\dot{\boldsymbol{\nu}}$ represent the accelerations in the USV frame. Inertia matrix $\mathbf{M}$ consists of rigid body matrix $\mathbf{M}_{rb}$ and de added mass matrix $\mathbf{M}_a$

$$\mathrm{M}_{\boldsymbol{rb}} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \tag{2.8}$$

$$\mathrm{M}_{\boldsymbol{a}} = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix}. \tag{2.9}$$

$\mathbf{M}_{rb}$ represents the inertia of the USV due to its mass and $\mathbf{M}_a$ refers to the added virtual inertia due to currents and displacement of the water around the USV. The exact values for this added mass are difficult to calculate. The numerical values are obtained for the Delfia ship of Delft university of technology as found in [16] by measuring empirically.. The values or shown in Table 2.1.

The second term $\mathrm{C}(\boldsymbol{\nu})\boldsymbol{\nu}$ of Equation (2.6) refers to the resistance of the USV caused by the coriolis and centrifugal forces, again split in a rigid body and added mass part

$$\mathrm{C} = \mathrm{C}_{\boldsymbol{rb}} + \mathrm{C}_{\boldsymbol{a}} \tag{2.10}$$

where

$$\mathrm{C}_{\boldsymbol{rb}} = \begin{bmatrix} 0 & 0 & -mv \\ 0 & m & mu \\ mv & -mu & 0 \end{bmatrix} \tag{2.11}$$

$$\mathrm{C}_{\boldsymbol{a}} = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ Y_{\dot{v}}v + Y_{\dot{r}}r & X_{\dot{u}}u & -0 \end{bmatrix}. \tag{2.12}$$

Coriolis and centrifugal forces are nonlinear and therefor dependent on $\dot{\boldsymbol{\nu}}$. The third term $\mathrm{D}\boldsymbol{\nu}$ of Equation (2.6) represent the damping caused by wriction between the USVand the water. D consists of a linear part $\mathrm{D}_{\boldsymbol{L}}$ and of a non-linear part $\mathrm{D}_{\boldsymbol{NL}}$

$$\mathrm{D} = \mathrm{D}_{\boldsymbol{L}} + \mathrm{D}_{\boldsymbol{NL}} \tag{2.13}$$

where

$$\mathrm{D}_{\boldsymbol{L}} = \begin{bmatrix} -X_u & 0 & 0 \\ 0 & -Y_v & -Y_r \\ 0 & -N_v & -N_r \end{bmatrix} \tag{2.14}$$

$$\mathrm{D}_{\boldsymbol{NL}} = \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & d_{23} \\ 0 & d_{32} & d_{33} \end{bmatrix}. \tag{2.15}$$

Table 2.1: numerical values of dynamic ship model

Parameter	**Value**
m	$3.345kg$
I	$0.031kgm^2$
d_{11}	-2.7343
d_{22}	$-.6025$
d_{23}	0.79546
d_{32}	0.50439
d_{33}	-0.22243
$X_{\dot{u}}$	-0.2310
$Y_{\dot{v}}$	-1.334
$Y_{\dot{r}}$	0
$N_{\dot{r}}$	-0.110

Chapter 3

Mapping and Localization

Mapping is the task of generating a map. Mapping is performed by connecting the observations of the Unmanned Surface Vessel (USV)'s sensors to the right position on the map. This position could either be acquired using positioning sensors like Global Navigation Satellite System (GNSS) or using Simultaneous Localization and Mapping (SLAM). Localization is determining the position of the vessel within a map. Mapping can be dependent of the location, and the location can be dependent on the map.

The map is required to generate an efficient route or path through the waterways. When the USV has to sail autonomously (Chapters 4 and 5) knowledge of all obstacles is required and therefor mapping is necessary. Obstacles can be static, like harbour walls or a lighthouse, but many obstacles have a changing position. Think about small boats, floating debris or moving shorelines due to change in the water level. These dynamic obstacles could all be subject to collision when not depicted in the map.

The map is dependent on the position and therefore the accuracy of the map is related to the position accuracy. For wheeled robots, positioning accuracy could be improved by using odometry data. Waterborn vehicles cannot benefit from odometry data, because the position of a waterborn vehicle cannot be directly derived from the control inputs (rudder angle and propeller speed) and previous states. The position of the ship is also affected by other factors such as wind, waves, current, load distribution, water depth and water displacement by other ships. Therefor the position of the USV can only be acquired using additional sensor measurements rather than feedforward methods like odometry.

Obstacles are detected using ranging sensors like Radio Detection And Ranging (radar) or Light Detection And Ranging (LiDAR) or visual sensors like cameras each having their own benefits and drawbacks. The properties of several obstacle detection sensors are described in Section 3.2. Localization sensors are described in Section 3.3. Section 3.4 describes the mapping process using position sensors and Section 3.5 describes mapping when acquiring the position from the ranging sensor. The latter is known as SLAM. The performance of mapping and SLAM algorithms will be evaluated by comparing them to the actual map in Sections 3.6 and 3.7. The algorithms acquiring the lowest average closest point distance between a measurement in the map and the ground truth is said to be the best mapping algorithm. Finally Section 3.8 concludes this chapter.

3.1 Map representations

There are different structures to represent the maps produced by mapping. The most important representation according to the author are described in this section.

3.1.1 Feature Based Mapping

Feature Based Map (FBM) [17] as used for feature based SLAM [18] is a finite collection of locations and corresponding probabilities of features at that location. These features are stored in a vector in arbitrary order. A major benefit for this approach is that is computationally less demanding than grid like approaches. This is beneficial in cases where fast response is required or the collected sensor data is contain a high number of data points. In the latter case the high dimensional sensor data could be compressed by selecting the useful data (features) and leaving out the remaining data. However drawbacks for FBM also remain: when a feature is detected by the robot, it has now to be matched to a feature in the vector with a corresponding estimation error. This causes problems because it is challenging to find the right match even without measurement errors due to the arbitrary order of the features in the vector. The order of the features could be changed however this results in a set of features rather than a vector due to the many but finite possibilities. Vector-based navigation approach need then to be replaced by set-based approaches. When some features remain undetected, it will be even harder to match all measurements to correspending features. FBM has been proven to work great in combination with particle filtering and low demanding applications. Due to a continously increasing computing power FBM has lost in popularity in favor of the grid mapping approaches.

3.1.2 Gap Navigation Tree

a Gap Navigation Tree (GNT) [19] is a 2D depth based approach. The robot applies depth measurement from increasing angle relative to its heading angle resulting in 2D pointclouds. Discontinuities in the sensor measurements are called *gaps* and are used as a navigation references. The robot can localize itself based on its surrounding gaps. The GNT is designed for efficiently solving different visibility-based robotic tasks including SLAM in unknown planar environments. A GNT is mostly used to avoid common grid type SLAM problems like complete map building and exact localization. Other approaches mostly assume a perfect geometric model to be available. A GNT is designed to minimize the information required by directly use sensor information instead of process the sensor data with internal models. Possibly, more reliable and less costly solutions could be found for challenging visibility-based robotic problems using a GNT.

3.1.3 Occupancy Grid Maps

An Occupancy Grid Map (OGM) [20] is the most used map representation, for example by Tomono [21]. An OGM is, opposed to the FBM and GNT a metric map represention method used to identify per cell (2D or 3D) whether it is occupied, free, or undiscovered. A higher resolution will result into smaller pixels and thus a higher precision of the map. However more computations have to be executed in order to explore the same area.

According to [22], the grid size depends on the accuracy of the sensors and the computational limits of the hardware. The accuracy of the sensors (partly) depend on environmental influences (lighting conditions, fog, etc.), and principle of the sensor (radars are less precise then lidars due to different wavelength for example). Too large pixel size in the grid decompose the high dimensional mapping problem into seperate single pixel one-dimensional estimation problems which are handled independently [23].

3.2 Obstacle detection

The USV will have to detect obstacles using its sensors. Several sensors for detecting obstacles are discussed in this section.

3.2.1 Ultrasonic sensor

Advantages	Disadvantages
○ independent of lighting conditions	○ pressure dependent
○ cheap	○ not reliable when foam, vapors, powder, dust, or tilted surfaces comes into the equation

Ultrasonic sensors emit ultrasonic waves. These waves are reflected by the surroundings and then received by the receiver of the sensor. Ultrasonic waves travel at a speed of 340m/s (speed of sound in air). When the speed of the wave is known, the distance can be calculated by

$$\text{distance} = \frac{\text{Speed of light} \cdot \Delta\text{time}}{2}. \tag{3.1}$$

Ultrasonic sensors can sense in the darkness, but are sensitive for obstacles like dust or rain drops. These particles will reflect the ultrasonic waves. Flat areas under an angle like the hull of a boat or non-reflective areas like foam will also fool the sensor. These surface will not reflect the waves back in the desired directions which makes it hard for the sensor to detect reflected waves. [24]

3.2.2 Mono camera

Advantages	Disadvantages
○ extracts color information which makes object recognition easier	○ can only measure distances when sizes are known and vice versa
○ high accuracy in lateral and height dimension	○ color dependent (difficulties with changing seasons or shadows)

A camera captures photons normally in visible range by photodiode sensors configured in an array (pixels). These sensors convert the photons to electrons. Applying color filters in front of each photodiode produces separate R-G-B pixel values. Camera sensors are affected by lighting conditions. Object in the shadow will be harder to sense and an autonomous boat which is trained with summer data will have harder times with navigating in winter. Sailing in the darkness seems impossible when the system is mainly relying on cameras. Cameras can estimate dimensions if other dimensions are known. For example the distance to a car can be estimated when knowing the approximate with of an average car because objects which are further away appear smaller in the picture.

3.2.3 Stereo camera

Advantages	Disadvantages
○ extracts color information which makes object recognition easier ○ high accuracy in lateral and height dimension	○ color dependent (difficulties with changing seasons or shadows)

A stereo camera sensor is a mount with two cameras spaced with a known distance. The working principle is the same as a stereo camera besides the fact that a stereo camera can estimate distances without prior knowledge of the object on a picture. The basic principle is based on the idea that an object in front shifts more when comparing to pictures from slightly different positions than an object in the background [25].

3.2.4 Radar

Advantages	Disadvantages
○ resistant to changes in weather conditions ○ can also measure longitudinal velocity	○ cannot detect low dielectric materials ○ not reliable at narrow bridges and tunnels due to echo

RAdio **D**etection **A**nd **R**anging (radar) works according to the same principle as an ultrasonic proximity sensor. The only difference is that EMR waves are used instead of ultrasonic sound waves. These waves propogate at the speed of light (300,000m/s) instead of the speed of sound(340m/s). This makes the radar significantly faster than an ultrasonic sensor. As higher frequency waves tend less to diverge, radar waves (24-77GHz) are easier to aim than ultrasonic waves (50-500kHz). This means that the range of a radar (≈200m) is significantly higher than the range of an ultrasonic sensor(≈4m). The higher the frequency of a radar system, the more it is affected by weather conditions such as rain or clouds. But the higher the transmitted frequency, the better is the accuracy of the radar system. Short-range radars (up to 30m) operate usually at 24GHz and have a wide beam ($\approx 80°$). Long-range radars operate usually at 76-77GHz and have a narrow beam($< 20°$). Radar can directly measure the speed of the target from the Doppler shift effect (FMCW) whereas camera and laser systems calculate speed as a derivative of range. Low frequency radars were used in the second world war but as higher frequency radars became available, lower frequency radars extincted. These days low frequency radars are in a comeback because they can even detect the shapes of stealth vehicles. It might be worth considering using a low frequency radar in order to detect boats, because boats are also hard to detect from the front of the hull due to its shape.

When mounting a radar on a rotating platform an extra DOF is created. These 2D radars are available in two main types: pulse radars and the upcoming broadband radar. Pulse radars send out a short EMR pulses and wait for a reflection afterwards. The main drawback for these systems is the waiting time between radiating the pulse and receiving the reflection of the pulse because the transmitter and receiver cannot operate simultanously. The receiver has to be turned off when the transmitter transmits the pulse because it will break due to the high power of the pulse. After radiating the pulse a small switching delay is present causing a typical minimum range of around 50 meter for rotating pulse radars. The counterpart of the pulse radars are de continuous radars. These radar can transmit and receive simultanously allowing them to measure at shorter minimum ranges and with low power waves. For example simrad's broadband radar transmits 10 times less radiation then a mobile phone while pulse radars ar produce several times as muchs radiation as a mobile phone. Continuous radars calculate distances by sending and receiving frequency shifted EMR waves and calculating the frequency shift rather than the time of flight.

[26]

3.2.5 2D Lidar

Advantages	Disadvantages
○ allrounder	○ cannot observe transparent areas
○ can also measure longitudinal velocity	

LIght **D**etection **A**nd **R**anging (lidar), sometimes also referred to as laser scanner, is based on the same principles as radar except for the wave frequency or wavelength. The single plane lidar or 2D lidar is a lidar operating with just one laser. The main issue with lidars is that they operate in the visible range and therefor need to be safe for the human eye. Lidar are said to be eye safe when emitting less than 1W into the human eye. Most commonly used Lidars work nowadays at the 700nm wavelength. The newer models work with 1550nm wavelength which is not focused by the human eye since it is out of the visible range. Lidars at 1550nm can work at higher power levels and therefor sense further. The strength of the reflected waves (lidar intensity) depends on the reflective properties of the composition of the object surface.

3.2.6 3D Lidar

Advantages	Disadvantages
○ allrounder	○ cannot observe transparent areas
○ can also measure longitudinal velocity	○ expensive
○ can recognize objects	

A 3D lidar works the same as the 2D lidar, but instead of a single laser, it has multiple lasers mounted on top of each other and with that multiple plains in the resulting point cloud. Nowadays lidar with up to 64 planes are manufactured. These 3D point clouds give a better view of the surrounding and could be used for object recognition, preferably in combination with a camera. The main downside is that this technique is still in the early stage of its development and therefor very expensive. Prices over €20k are no exceptions for the high end models. The use of lidars in combination with computer vision has been proved before [27]

Table 3.1: Comparison of sensors for object detection.

	Camera	Stereo camera	Radar	2D lidar	3D lidar
Lateral accuracy	✓✓	✓✓	××	✓	✓
Longitudinal accuracy	××	×	✓✓	✓	✓
Object recognition	✓	✓	×	×	×
Viewing angle	×	×	×	✓	✓
Costs	✓✓	×	✓	×	×××

For this research several sensors have been mounted. Two planar (2D) LiDAR are mounted at the starboard and port side of the bow as depicted in Figure 3.1a. These LiDAR are used for detecting the surroundings and other vessels. The LiDAR are limited by range which is approximately 15m for fully reflecting white surfaces. Range reduces when the weather turns foggy or the sun disturbes the reflcections of the laser. Black surfaces can only be detected at reduces range due to limited laser reflecting capabilities. Another downside of the 2D LiDAR is its single plane of view as depicted in Figure 3.1b. In this figure the LiDAR is able to detect the wall (dark red) but it will not return any detection of the jetty (orange) between the USV and the wall.

Around the center of the USV a GNSS and Inerial Measurement Unit (IMU) sensor are placed. The GNSS sensor is used to obtain the global location of the USV. This obtained position has a precision of around 2.5m. Just the location is not enough to define the position of the USV and therefor a magneto meter is added. The magnetometer is embedded in the IMU and measures the magnetic field of the earth to obtain the orientation of the boat relative to the direction of the north pole. The magnetometer has an precision which is higher than the angular precision of the lidar. The orientation can therefor be obtained with sufficient precision by the magneto meter. However the position accuracy of 2.5m is too low for the application of this research. In order to improve the position accuracy an IMU is added. This IMU consist of a 3DOF gyroscope, 3DOF accelerometer and 3DOF magnetometer. The IMU excels at detecting changes in the position but suffers from drift. Combining the characteristics of both the GNSS and IMU, when properly merged together using Kalman Filtering, will increase the position accuracy. A position accuracy of 0.3m is expected to be feasible.

3.3 Localization

Several different sensor could be used for localizing the USV. This section describes odometry, the IMU and GNSS sensor.

3.3.1 Odometry

The simplest example of odometry is measuring wheel speed and integrate this for calculating the total distance travelled. This works well for wheeled robots which do not slip and steer a lot. However always an integration error will occure. Therefor the odometry measurements will somehow have to be filtered and compenstated for drift. Measuring wheel speeds is not possible at waterborn vehicles but measuring throttle and rudder outputs is possible. However, the connection between an USV and the surface is not as direct as the connection between a wheeled vehicle and the ground. When a very good ship model is implemented, odometry could be applied by calculating the motion of the USV as function of its currents states and the throttle and rudder input. Another option is visual odometry. Visual odometry matches camera images to find a difference in position of the images taken. This difference represents the movement of the USVbetween the two images taken. Visual odometry does not drift due to an integration error and it has been shown to have the capabilites to reach for an accuracy up till 0.067m [28]. However it cannot be used when the surroundings are not visually changing when moving or when it is dark. For example, visual odometry cannot be applied when sailing next to a straight uniformly colored wall or next to moving objects.

3.3.2 IMU

An IMU is an unit containing different position measurement sensors. Some 6 DOF versions contain a 3 DOF accelerometer and a 3 DOF gyroscope. A high end IMUis extended with a 3 DOF magnetometer. The accelerometer measures the acceleration in all three translational direction. Accelerometers produce noisy data but do not suffer from drift. This means that filtering of the accelerometer data could provide accurate estimations of the translations accelerations over long time. Due to the fixed downward vertical direction of the gravity an accelerometer is also suitable for obtaining pitch and roll angle when applying some simple geometrics. A gyroscope measures angular velocities around all 3 axes with high precision but suffers from drift. Fast changes in rotational velocity can be precisely detected but on the long term no fair estimate of the orientation can be made due to integration errors. Combining the properties of an accelerometer and gyroscope using for example kalman filtering wil result in an unit able to detect fast changes in orientation which will not suffer to drift due to the accurate position measurements of the accelerometer. A magnetometer measures the magnetic field and can therefor obtain the yaw angle like a compass. A magnetometer can be added to the IMU to compensate for the drift in yaw rate measurements of the gyroscope using the magnetic field of the earth as a reference for the yaw angle. A combinition of an accelerometer, gyroscope and magnetometer is often called an IMU.

3.3.3 GNSS

GNSS is a satellite based positioning system. Several GNSS are available of which the most known and utilised is the American Global Positioning System (GPS). Russia has its own GNSS called glonass and Europe is planning on fully releasing their system this year (2020). GNSS systems are limited to an accuracy of approximately $10m$. Other than with accelerometers or velocity meters which rely on integration of the signal to accomplish the position, GNSS does not suffer from drift due to integration error because the position is the direct output of the sensor. Due to its low accuracy the GNSS signal will mostly not be used as raw data, but some processing will be done in order to increase the accuracy.

Sensor data merging

The precision of the GNSS position estimate can be increased by merging sensor data. A well known example is Kalman filtering of IMU data and GNSS data to acquire a precise estimate for the position by relying on the IMU for quick changes in velocity and relying on the GNSS data to compensate for the integration error of the IMU. In some cases even the dynamics and kinematics of the specific USV will be included as well resulting in a USV specific position estimate of high accuracy referred to as Real-time Kinematics (RTK).

GNSS Augmentation

GNSS accuracy is limited to ≈ 10 meters due to the atmosperic influences, satellite constellation, orbital error, satellite clock error and other physical limititations. Compensating for these errors by supplying external knowledge is also known as GNSS augmentation. This could be done by providing the speed of light in the ionic sphere or the orbital error for example. Another well known example is adding a base station with a known position as Differential GPS does. In this case an extra station is added which is less affected by physical disturbances since it is significantly closer to the sensor. However a complete coverage of base stations is required which is currently only the case is some densely populated areas, including the Netherlands.

Sensor configuration

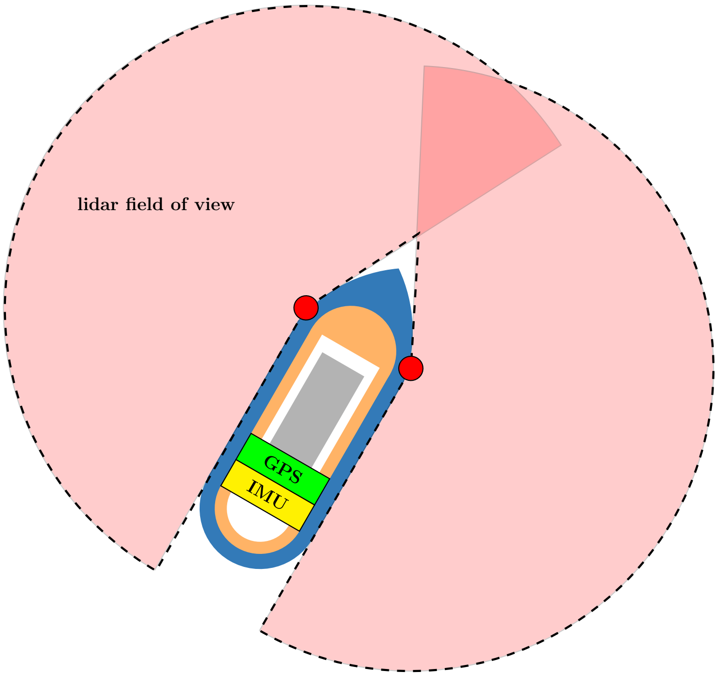
Based on the above sensor information the sensor configuration as shown in Figure 3.1 has been chosen. Two planar lidars are mounted at both sides of the bow. this will result in a field of view as depicted by the red area. The planar lidars have been applied because of its high precision and resolution measurements. Drawbacks are only its planar field of view resulting in possible errors when obstacle or out of sensing plane. Also major errors will occure when the USV is subject to heavy pitch and roll motions. Therefor the assumption for research is made that no substantial waves exist in the water the USV is sailing. The reason for utilize 3D LiDAR instead of 2D LiDAR is financial. Together with the two lidars, A GNSS and IMU sensor are mounted for global position estimation. In simulation all object are assumed to be in height range of the lidar. However, due to planar limitation of the 2D LiDAR it was not possible to use the data acquired by the real sensor using the model ship outdoors because it misses a lot of object or only senses water when tilted in the roll direction. The simulation does only assume 3 DOF which means that the roll motion is neglected. The real sensor could be used for acquiring data on the real size Xomnia vessel because the quays in Amsterdam are usually straight, high and vertical meaning that roll motion does that change the data much.

3.4 Mapping using positioning sensors

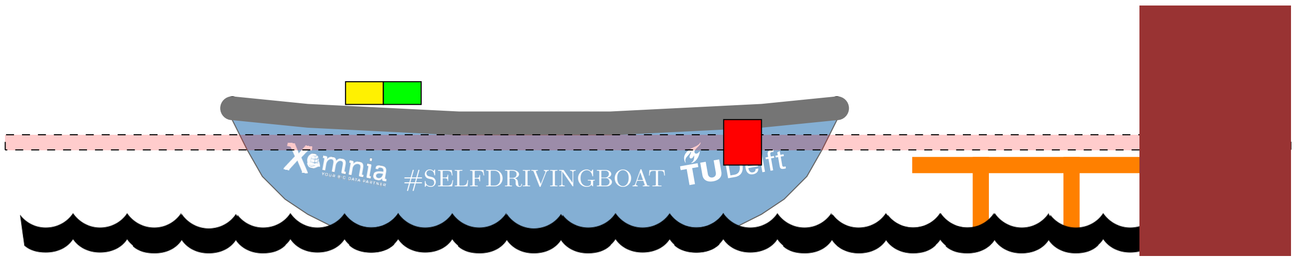
The position of a USV could either be expressed in a global coordinate system like GPS coordinates or in local coordinates relative to the first position measurement as an origin. Measurements are done with sensors attached to the USV and are therefor expressend in the USV frame. Assuming the position of the USV is known, either globally or locally, The acquired sensor data could be plotted by connecting the measurement to the USV on the map [29]. This will create a perfect map in case the position and sensor data are provided without uncertainty. However every measurement has a variance. The sum of variances for mapping can be defined as

$$\sigma_{map} = \sigma_{position} + \sigma_{sensor}. \tag{3.2}$$

In order to create a useful map the variance should be less or equal than the precision required for maneuvering. And still the sensors data will contain outliers. Faulty detections, or maybe reflections of dust could already result in a map with large errors. In order to create the validity of the map, the map will contain of probabilities

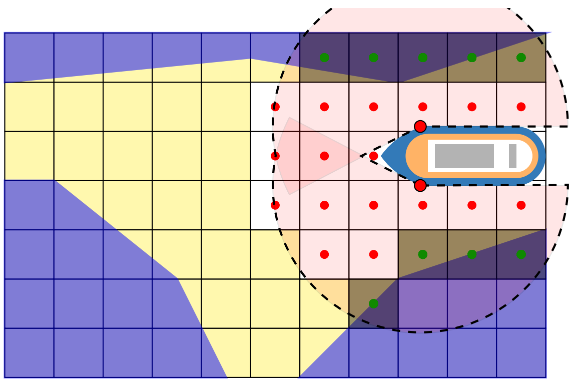


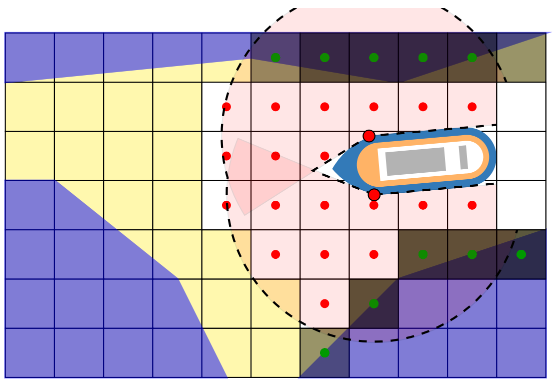
(a) top view

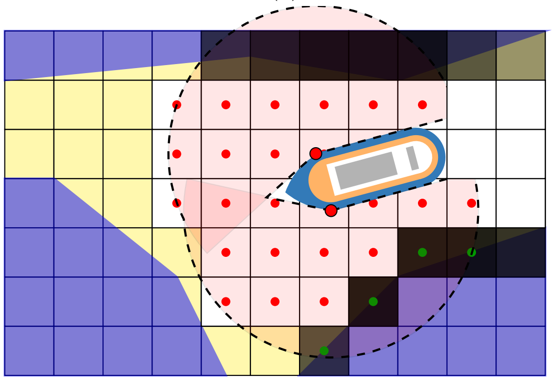


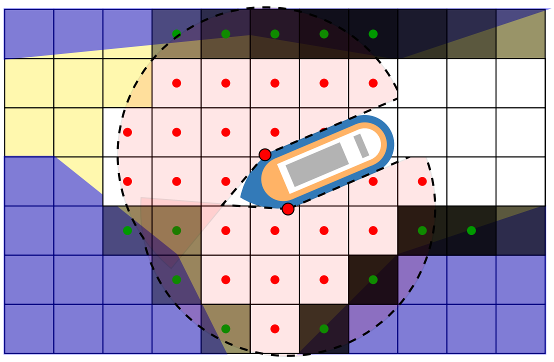
(b) side view

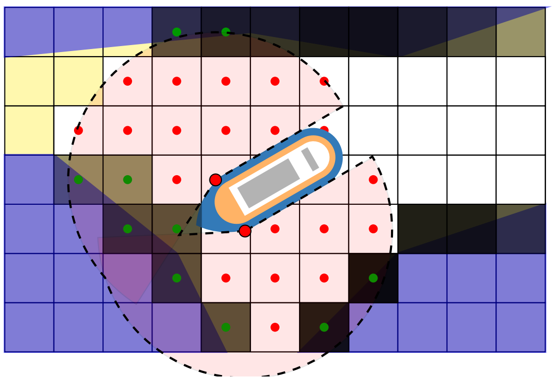
Figure 3.1: Sensor setup. Two 2D lidars (red) are mounted at the bow and a GNSS (green) and IMU (yellow) are mounted at center of rotation. The pink area shows the lidar field of view


(a) 1


(b) 2


(c) 3


(d) 4


(e) 5

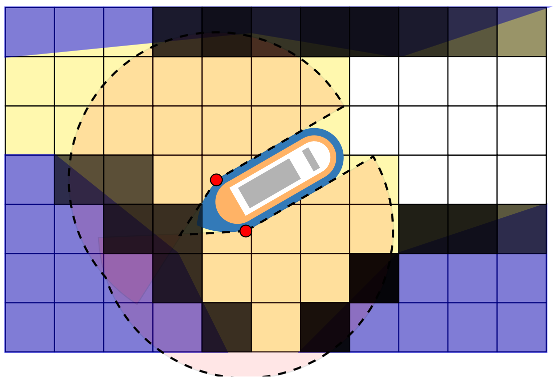

(f) 6

Figure 3.2: The occupancy probability of the green nodes are increased as they are within sensor range and at the location of an obstacle. The occupancy of the red nodes decrease because they represent no obstacle and are within range of the ranging sensor. The black cells represent the occupancy probability of the cell. Darker black means higher occupancy probability

of an cell of the map being occupied rather than just being a sum of every measurment so far. The occupancy probability of a cell will increase when a sensor measurement is detects an obstacle within the range of the sensor as shown in Figure 3.2 The occupancy probability of a cell will decrease when the cell is within range of the sensor, but no object is detected. When a cell has never been in range of the sensor, or is blocked from view by other objects, this cell will be marked as unknown. The algorithms is defined by:

1. **Initialize** a map of predefined size with every cell marked as unoccupied. unoccupied is referred upon as the numeric value -1. 0 probability is referred as 0 and 100% percent change of the presence of an obstacle is noted with 100. The initialized map is a matrix with a number of columns and rows corresponding the map width and height respectively. Every entry of the matrix consist initially of a 0. This step has only to be performed at the beginning of the algorithm as has only to be performed only once.
2. **Locate** the USVin the map, and map the sensor data to the map frame at this position. This step will make sure that the sensor data will correspond to the correct cell in the map. This step has to be performed everytime when a new position estimate is available.
3. **Increase** the occupancy probability for those cells which are occupied according to the most recent measurement. The occupancy probability will in this case be increased by a value $p_{increase}$ between 0 and 100. This means that 5 sequentially measurements of the same occupied cell will return into a 100% probability of this cell to be occupied when a value of 20 is used.
4. **Decrease** the occupancy probability for every cell between the USVposition and detected cell. The value of the decrease differs related on the current occupancy probability p of the cell according to:

$$p_{new} = max(p_{current} - \frac{p_{increase}}{50} \cdot (100 - p_{current}), 0) \tag{3.3}$$

This means that a cell of probability 100 will remain 100. A cell with a low value will be decreased more than a cell with a high value. The value of $p_{increase}$ is evaluated in Section 3.6. Decreasing the probability related on its value is implemented in order to make sure that once detected obstacles will remain in the map when they have been detected several times.

3.5 Simultanuously localization and mapping

Simultanuously Localization and Mapping (SLAM) has two main applications. The first one is SLAM for map acquisition [30] which is used to map an environment for later use. It could be used for example to map the docking environment just requiring a known reference position. This reference position could be acquired by placing reference positions in the environment with known locations, or odometry for wheeled robots for example. The data could be recorded and processed offline on a different machine when the data acquisition has been finished in other to generate a map of the unknown environment. This generated map could then later be used to determine the boats location by comparing the sensor inputs to the mapped image. Just like a roadmap basically. SLAM for map acquisition, compared to real-time mapping has as its main benefit that computationally more demanding algorithms could be used due to absence of the real-time operating requirement. More demanding algorithms, will require more computational power, but will generally also perform better in the sense of accuracy and resolution of the generated map.

In another more well-known application, SLAM is applied to locate an autonomous robot online in an unknown environment with or without prior knowledge of its location. Online SLAM algorithms for localization are available in a wide spectrum of different techniques, mostly due to varying sensors and different environments. As stated in [22], SLAM techniques could generally be ordered in three different classes: *visual SLAM* , *lidar SLAM* and *sensor fusion SLAM* . While visual and lidar slam could also contain many types like monocular camera, stereo vision, lidar, radar sensors or a combination of these sensors. An example a vision based SLAM is provided by [31], An example for an Feature based 3D SLAM using sensor fusion using a depth sensor to find regions of interest in the camera images is provided by Jamiruddin [32]. 360°radar based SLAM would uses similar concepts als lidar based SLAM. However radar sensors are more noisy than lidars and therefor required more noise reduction techniques and more robust algorithms.

This section explains two well-known SLAM algorithms into further detail: Particle Filtering and Extended Kalman filtering

3.5.1 Particle filtering for SLAM

Particle filtering or Sequential Monte Carlo [33] is a localization algorithm which does not necessarily require a complete map for localization. Therefor this algorithm could also be used for SLAM applications. The particle filters approaches the actual location of the robot by virtually placing particles randomly across the available map. When the robot moves, sensor data is updated and several particles might be excluded due to non-matching sensor data with the map. Finally only one possible position remains, being the actual position of the robot. Noise is introduced to create several particles moving like a swarm around the robot. For each particle the likelihood of being the actual position is calculated. When a particle deviates to far off the robot position, the particle is removed and a new particle is placed at the robots actual location.

3.5.2 Extended Kalman filter

Another method for localization is Extended Kalman Fiter (EKF). EKF can be used as a noise smoothing filtering method when noisy measurements (I.E. GPS) are involved. Besides noise smoothing a state estimation is produced by the EKF. Considering linear dynamic system:

$$\mathbf{x}(k+1) = \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)\mathbf{u}(k) \tag{3.4}$$

$$\mathbf{y}(k) = \mathbf{H}(k)\mathbf{x}(k) \tag{3.5}$$

- $\mathbf{x}(k)$ is the state vector
- $\mathbf{u}(k)$ is the input vector
- $\mathbf{y}(k)$ is the output vector
- $\mathbf{F}(k), \mathbf{G}(k)$ and $\mathbf{H}(k)$ are the known system matrices

The main problem in solving the set of equations of the dynamic system from Equations (3.4) and (3.5) is that the state vector $\mathbf{x}$ is not directly measurable but has to approximated. The Kalman filter produces a state vector estimation by following the following steps:

Table 3.2: Root means square error and maximum error in meters for mapping with positioning sensors using different probability increases without noise

probability increment	**coverage**	**RMS error** [m] 1	2	3	**avg**	**max**[m] 1	2	3	**avg**
2	35%	0.0242	0.0226	0.0236	**0.0235**	0.06	0.06	0.06	**0.06**
5	58%	0.0253	0.0265	0.0284	**0.0267**	0.06	0.08	0.08	**0.07**
10	69%	0.0279	0.0302	0.0322	**0.0301**	0.06	0.08	0.10	**0.08**
15	75%	0.0285	0.0316	0.0346	**0.0316**	0.08	0.10	0.12	**0.10**
20	79%	0.0287	0.0328	0.0365	**0.0327**	0.10	0.09	0.12	**0.11**

Table 3.3: Root means square error and maximum error in meters for mapping with positioning sensors using different probability increases with noise with $0.1m$ lateral and $0.01°$ accuracy

probability increment	**coverage**	**RMS error** [m] 1	2	3	**avg**	**max**[m] 1	2	3	**avg**
2	22%	0.0245	0.0219	0.0249	**0.0238**	0.06	0.06	0.06	**0.06**
5	62%	0.0267	0.0268	0.0283	**0.0273**	0.08	0.09	0.08	**0.08**
10	78%	0.0303	0.0335	0.0320	**0.0319**	0.09	0.11	0.09	**0.10**
15	86%	0.0323	0.0375	0.0341	**0.0346**	0.10	0.10	0.12	**0.10**
20	90%	0.0335	0.0403	0.0358	**0.0354**	0.14	0.10	0.10	**0.11**

1. make a prediction based on the last estimate:

$$\hat{x}(k+1|k) = \mathbf{F}(k)\hat{x}(k) + \mathbf{G}(k)\mathbf{u}(k) \tag{3.6}$$

$$\hat{y}(k) = \mathbf{H}(k)\hat{x}(k+1|k) \tag{3.7}$$

2. calculate correction based on current measurement and the prediction defined in the previous step:

$$\Delta\mathbf{x} = f(\mathbf{y}(k+1), \hat{x}(k+1|k)) \tag{3.8}$$

3. update prediction based on correction:

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + \Delta\mathbf{x} \tag{3.9}$$

The performance of this EKF changes with the observer as presented in Equation (3.8). The function f could be stated as a simple linear function but also more complex function could be executed in this algorithm. Rather than finding an approximate solution using a complex model like the particle filter does, the EKF, finds an exact solution using a simplified model. Kalman filters can only keep track of a single position. When the initial position is chosen wrongly in the map, it is hard to recover from this mistake using EKF. However when the map has a high number of lookalike point and symmetric paths an EKF is more likely to keep track of the current location than a particle filter would.

Table 3.4: Root means square error and maximum error in meters for SLAM

errornous maps	**coverage**	**RMS error [m]**					**max[m]**				
		1	2	3	4	**avg**	1	2	3	4	**avg**
exluded	66%	0.0217	0.0180	0.0190		**0.0235**	0.06	0.06	0.06		**0.06**
inluded	60%	0.0217	0.0180	0.0190	0.1898	**0.0488**	0.06	0.06	0.06	0.41	**0.15**

Table 3.5: Root means square error and maximum error in meters for mapping with positioning sensors and SLAM for different maps

map	**coverage**	**RMS error [m]**				**max[m]**			
		1	2	3	**avg**	1	2	3	**avg**
map1 mapping	80%	0.039	0.033	0.033	**0.035**	0.14	0.10	0.10	**0.11**
map2 mapping	25%	0.024	0.028	0.022	**0.025**	0.09	0.16	0.12	**0.12**
map1 SLAM	75%	0.022	0.018	0.019	**0.020**	0.06	0.06	0.06	**0.06**

3.6 Results

The mapping algorithm from Section 3.4 and the Particle filtering for SLAM from Section 3.5 have been applied for further insights. Two different original maps are used for evaluation. The first map is a manually generated map consisting of only horizontal and vertical lines. Dense areas with tight spaces are present but also more spacious areas consist. This map is mostly used to evaluate the specification in dense areas with constant close presence of obstacles. This is the map which is mainly used is this section. The second map is more open and represents the areas in front of the AMS building in amsterdam. It is based on a printscreen of Google maps. This map is mainly used to evaluate the performance of the mapping and SLAM in real environments.

Now the map is uploaded to the simulation environment so the simulated sensors will return data regarding the right map. The simulated USV will maneuver manually through the simulated environment according to the routes as shown in Figures 3.4a and 3.5a. The acquired data will be turned into mapping by either performing the mapping or SLAM algorithm. For the SLAM algorithms the particle filtering is preferred above the EKF. The EKF might have a chance to be more accurate when implemented correctly but it will always suffer when a mislocalization occures. Particle filtering will recover more easily from mislocalization and is therefor more likely to create a complete map albeit with less accuracy than EKF would do. The particle filtering for SLAM is compared with the mapping algorithm using positioning sensors. For every occupied cell in the original map, the corresponding occupancy in the created map is checked. If the created map shows an occupied cell here as well, this is counted as a true positive. If the created map shows an unoccupied cell at an occupied location of the original map, this cell is marked as an false negative. The coverage is calculated according to:

$$\text{coverage} = \frac{\#\text{true positives in created map}}{\#\text{occupied cells in the original map}} \tag{3.10}$$

The coverage is shown in Table 3.2, Table 3.3 and Table 3.4. Next, for every occupied cell in the created map, the closest cell in the original map is calculated. The Root Mean Square (RMS)error is calculated according to:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\text{error}_j^2} \tag{3.11}$$

For N errors. The RMS error is comparable to the mean error but greater errors are penalized more widely used in comparing images. The RMS errors and maximum errors are shown in Table 3.2, Table 3.3 and Table 3.4 per image and averaged over the 3 images. For the mapping with position sensors a the value of $p_{increase}$ is gradually increased between 2% and 20% increase of probability per detection. The resulting maps are shown in Figure 3.3. All simulations are run 3 times with a Gaussian noise of with a standard deviation of $0.01m$ lateral and 0.01° rotational. On large size vessels, the positioning sensor lateral noise could be neglected as modern GNSS system could reach up to an accuracy of $0.05m$ and ship lenght could easily increase over $100m$. These number will result in to a relative error below 0.05%. However this thesis is performed on small scall vessel and therefor some noise is taken into account. $0.01m$ refers to a tenth of the vessel width. 0.01° rotational accuracy refers to the accuracy of a highend compass.

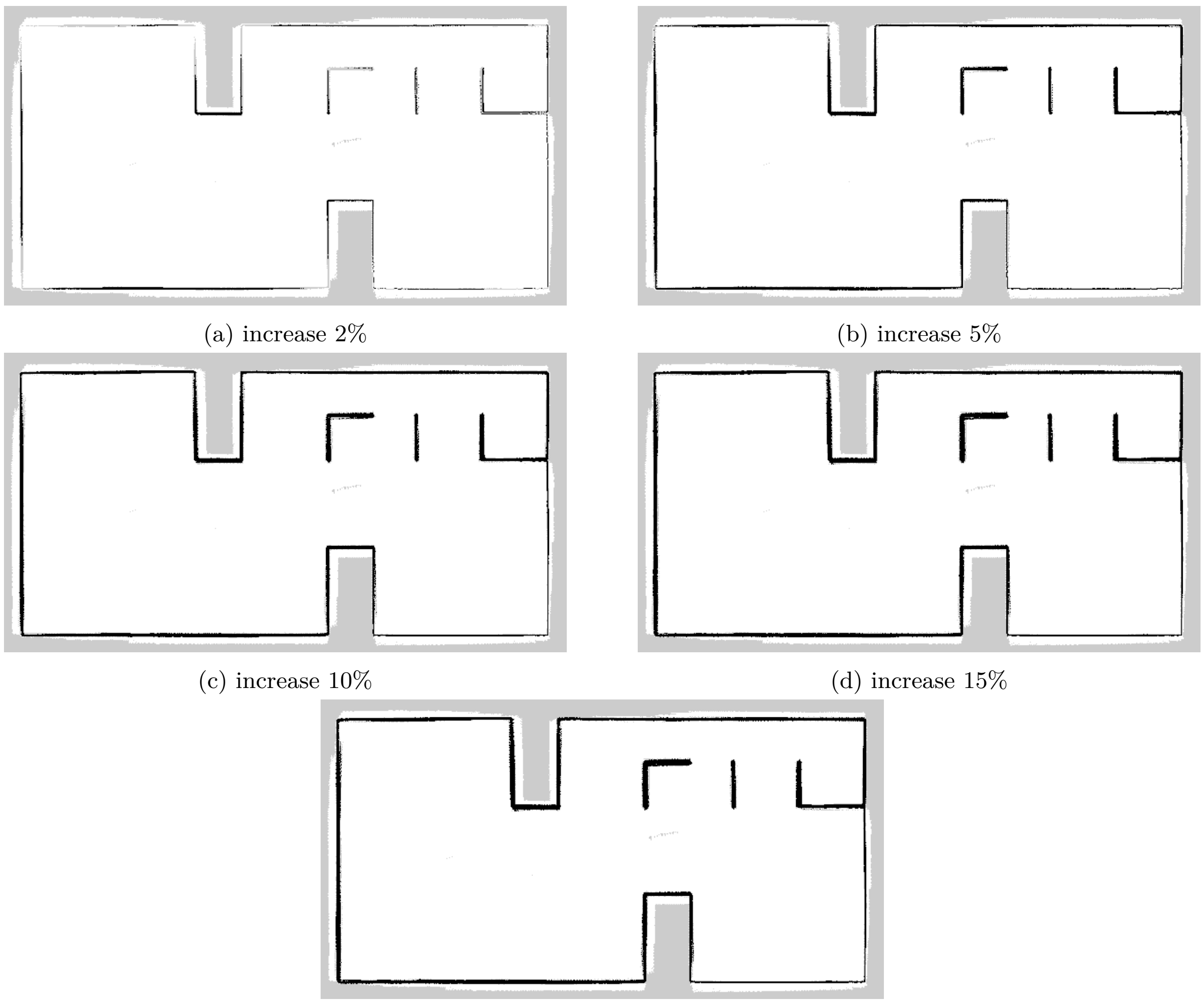
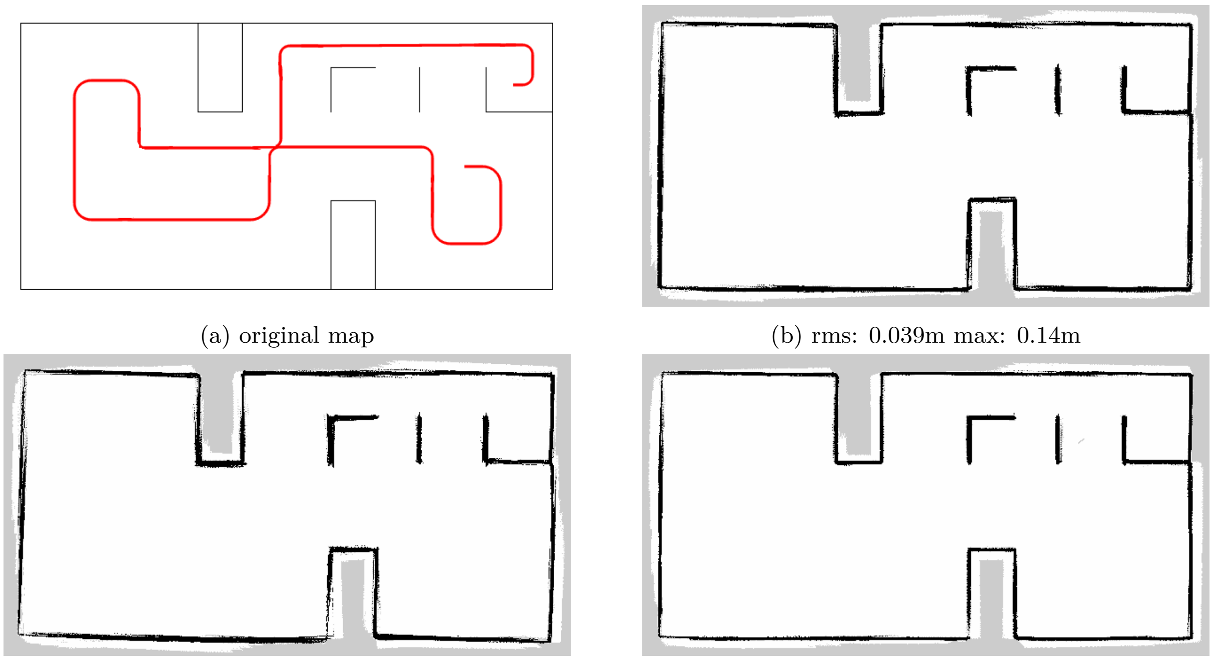

(a) increase 2%
(b) increase 5%
(c) increase 10%
(d) increase 15%
(e) increase 20%

Figure 3.3: Mapping with positioning sensors with different numbers for the probability increase when a cell is detected to be occupied.

are shown in Figures 3.4 to 3.6.



(a) original map

(b) rms: 0.039m max: 0.14m

(c) rms: 0.033m max: 0.10m

(d) rms: 0.033m max: 0.10m

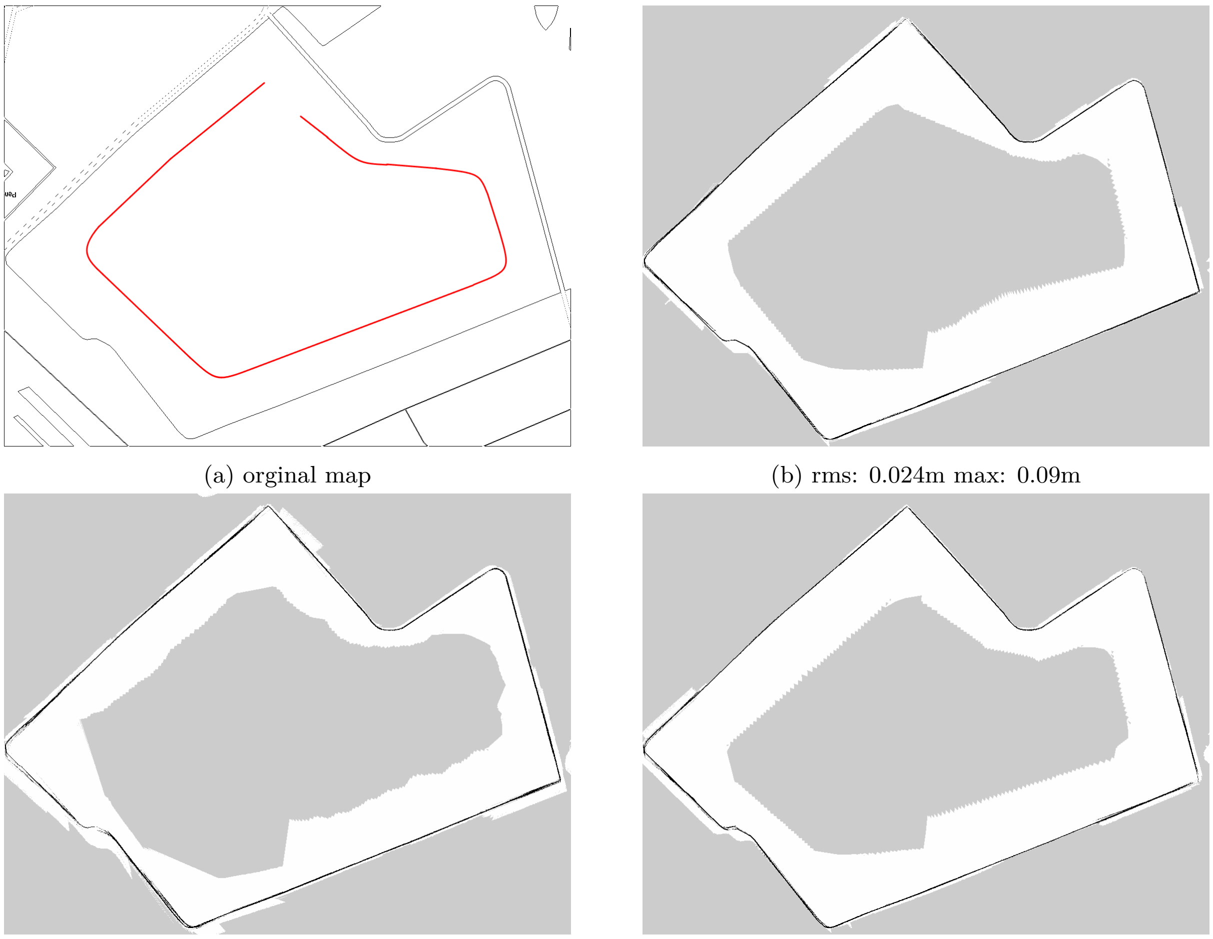
Figure 3.4: Mapping with orginal map (a) following the red path. The mapping algorithm have been executed 3 times with a $p_{increase}$ of 20%. The root mean square error and maximum error between the attempts (b,c,d) are denoted.

Figure 3.7 shows a failed map created by the SLAM algorithm. Due to faulty position estimation, the SLAM lost track of the position of placed measurements at the wrong pixel of the map.

3.7 Discussion

The RMS error can only tell us anything about the accuracy of the system when a high coverage is achieved. High coverage is achieved when the occupation probability raises fast with every detection. This means that $p_{increase}$ should be high. The optimal value will be a trade-off between coverage and RMS error. Generally, the error will increase when less measurement are required for sensing obstacles, but obstacels will be sensed. Therefor no scientific bounds can be given for tuning the $p_{increase}$. However, the RMS values are significantly lower than the safety bound which will be applied on the map in Chapter 4 and therefor the values with the highest coverage is chosen. $p_{increase}$ is set to be 0.2 (20%). This also benefits the act of detecting objects which are meesered with only a few measerements. This is beneficial because false negatives a more a thread to the safety of the system than false positives. The mapping with positioning sensors has been performed with an average RMS error of 0.0327m and an average maximum error of 0.011m in the first environment as shown in Figure 3.4. The mapping with positioning sensors has also been performed with noise on the positioning as shown in Table 3.3. The coverage of the map increases as more points are contained in the map with noise. For the same reasen the RMS error also slightly increase when noise is involved.

Table 3.5 reports small error for the second environment. The difference in both values most likely result from


(a) orginal map

(b) rms: 0.024m max: 0.09m

(c) rms: 0.028m max: 0.16m

(d) rms: 0.022m max:0.12m

Figure 3.5: Mapping with orginal map (a) following the red path. The mapping algorithm have been executed 3 times with a $p_{increase}$ of 20%. The root mean square error and maximum error between the attempts (b,c,d) are denoted.

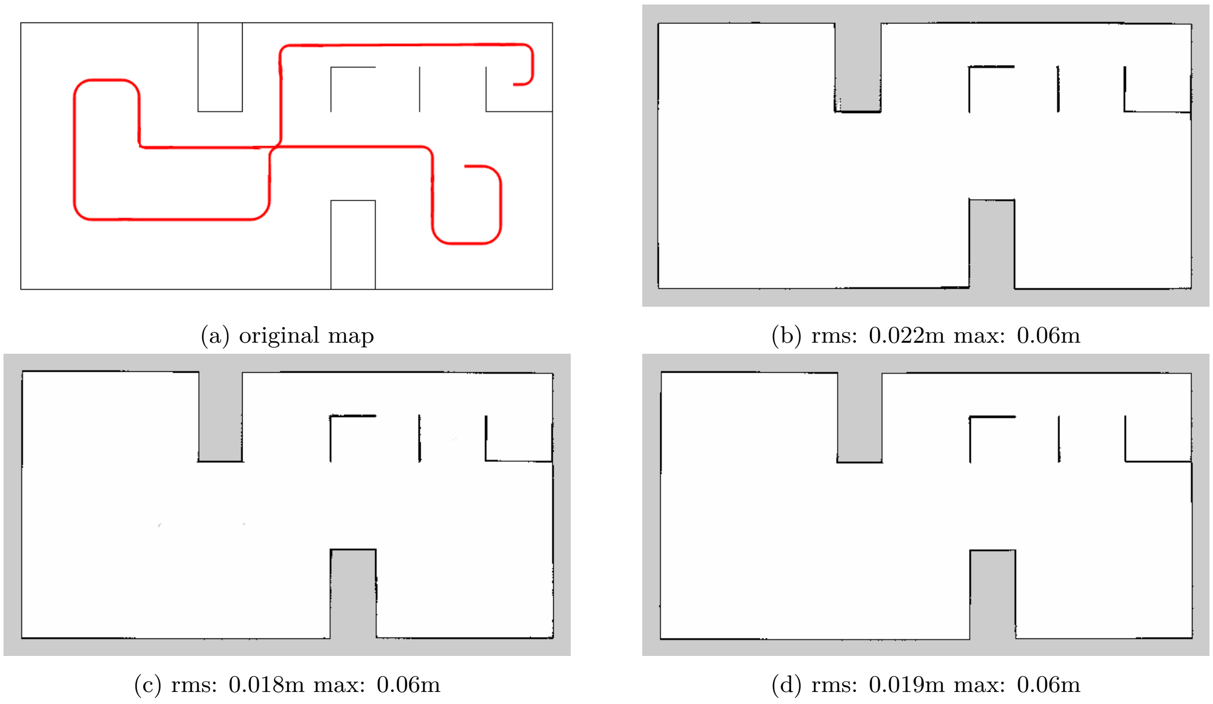
the difference of the paths to be followed. The path from the first environment Figure 3.4a contains relatively more and sharper turns per unit lenght of path than the path used for the second environment Figure 3.5a. Fast rotations could result in large error. For example: a 90 degree corner is performed in 10 seconds. A rotational velocity of

$$r = \frac{\text{angle}}{\text{time}} = \frac{0.5\pi}{10} = \frac{\pi}{20} \tag{3.12}$$

and an object measured at 15m at 0.2s delay will result in an error of

$$\text{error} = \frac{\text{delay}}{\text{distance} \cdot r} = \frac{0.2}{15 \cdot \frac{\pi}{20}} = 0.08m. \tag{3.13}$$

These could even increase when time delay between GPS/IMU data and lidar increases or the mapping algorithm required more time due to weaker hardware. Straight paths account for almost zero error due to zero rotational velocity and therefor the second environment can be mapped more accurately.


(a) original map

(b) rms: 0.022m max: 0.06m

(c) rms: 0.018m max: 0.06m

(d) rms: 0.019m max: 0.06m

Figure 3.6: SLAM with orginal map following the red path (a). The mapping algorithm have been executed 3 times with a $p_{increase}$ of 20%. The root mean square error and maximum error between the attempts (b,c,d) are denoted.

The SLAM algorithm reports an average root means square error of 0.024m and an avererage maximum error of 0.06m for the first map. This is lower than the mapping algorithm and therefor it can be said that the SLAM does perform best out of the three runs based on the first map based on the errors. However several comments have to be taken into account. The SLAM algorithm was not be able to make a map out of the second environment and returned an errornous map as shown in Figure 3.7. The SLAM algorithm requires an environment which can differ between small time steps. This is possible when objects are around and move relatively to each other. However, when straight lines are present without further marks or movement, the SLAM algorithm cannot allign the lidar images properly. The SLAM performance for the first environment was not flawless either. Only the completed maps have been taken into account. An example for an uncompleted map produced by the SLAM algorithm is shown in Figure 3.7. The mapping is preferred for its robustness and will therefor be applied in the final system.

3.8 Conclusion

Mapping and localization takes a fair share in the task of autonomous maneuvering on the water. Before mapping can be applied a suitable map representations has to be defined. In this case the OGM is preferred over the FBM and GNT for its widespread applications and straightforward implementation. Because OGM is the standard in robotics, an OGM is required for the best known path planning applications. Next the sensors for mapping has to be defined. High resolution data is required for mapping in both longitudinal and lateral direction combined with a range of at least the size of the vessel. Camera type sensors do not fullfill the range

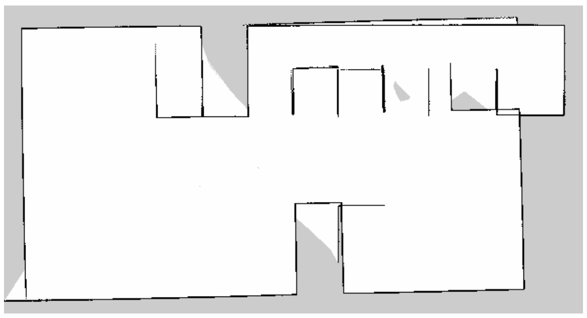


Figure 3.7: SLAM map with error

requirements and the other sensors cannot omit data with desired resolution except for the LiDAR. The 2D LiDAR is preferred over a 3D LiDAR for financial reasons. Localization is performed using GNSS/IMU sensor fusion for its high accuracy and global reference frame. Two different methods for mapping can be performed. SLAM has de ability to outperform mapping using positioning sensors in accuracy of the map. However higher robustness and the presence of a global reference frame which does not suffer from drift favours the mapping using positional sensors. The mapping with positional sensors does handle noise in the positioning well due to its high update rate.

Chapter 4

Path Planning

Path planning is the act of calculating a path between a starting point and a goal given a map of the environment. Chapter 3 discussed how to obtain an Occupancy Grid Map (OGM) that contains information about which of the cells represent obstacles, which cells are free of obstacles and which cells are unseen and therefore unknown. In this chapter, this map will be used to perform the path planning, such that the system can later determine the required control inputs.

Path planning requires a known initial position of the Unmanned Surface Vessel (USV) and a given goal position. It will then use the OGM to optimise an interconnection such that obstacles are avoided. For this interconnection, there is a choice whether to include time, and this is the difference between two different route types: a path or a trajectory. A path is a sequence of positions of lenght k

$$\text{Path} = \left\{ \begin{bmatrix} x_0 \\ y_0 \\ \psi_0 \end{bmatrix}, \begin{bmatrix} x_i \\ y_i \\ \psi_i \end{bmatrix}, \dots, \begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix} \right\} \text{for } i \text{ in } [0:k]. \tag{4.1}$$

Every position contains an x and y coordinate and a heading ψ. Connecting the entries of a path sequentially will show the complete path. Paths are used in time-invariant situations where the arrival time at the goal is irrelevant. A trajectory is a path where every waypoint is stamped with an arrival time.

$$\text{Trajectory} = \left\{ \begin{bmatrix} x_0 \\ y_0 \\ \psi_0 \\ t_0 \end{bmatrix}, \begin{bmatrix} x_i \\ y_i \\ \psi_i \\ t_i \end{bmatrix}, \dots, \begin{bmatrix} x_k \\ y_k \\ \psi_k \\ t_0 \end{bmatrix} \right\} \text{for } i \text{ in } [0:k]. \tag{4.2}$$

This means that besides desired positions also desired velocities could be derived. A trajectory is useful when the trajectory of other vehicles is known or when the tides change the occupancy of the map. Moving vehicles with known trajectory make some areas in the map only accesible at specific times and therefore the path has to be given a time stamp. This thesis does not cover other moving object with known trajectory or specific arrival time at the goal.

Although there are different methods to carry out the path planning itself, all approaches share the following structure:

1. **Node distribution** First, the map is populated with nodes (unoccupied and reachable) including feasible edges that interconnect these nodes. Here, feasible relates to occupancy of the map .System dynamics are not in scope until Chapter 5.
2. **Shortest Path** Given the node distribution, the next step is about finding the optimal sequence of nodes that connect the initial position and the goal. Optimality here simply means shortest distance.

This chapter covers the two subproblems and the most important solutions in the first two sections, namely Section 4.1 and Section 4.2. Next, Section 4.3 covers *map inflation*, a technique to account for the size of the boat when avoiding obstacles. The results, the discussion and the conclusion are presented in Sections 4.5 to 4.7.

4.1 Node distribution

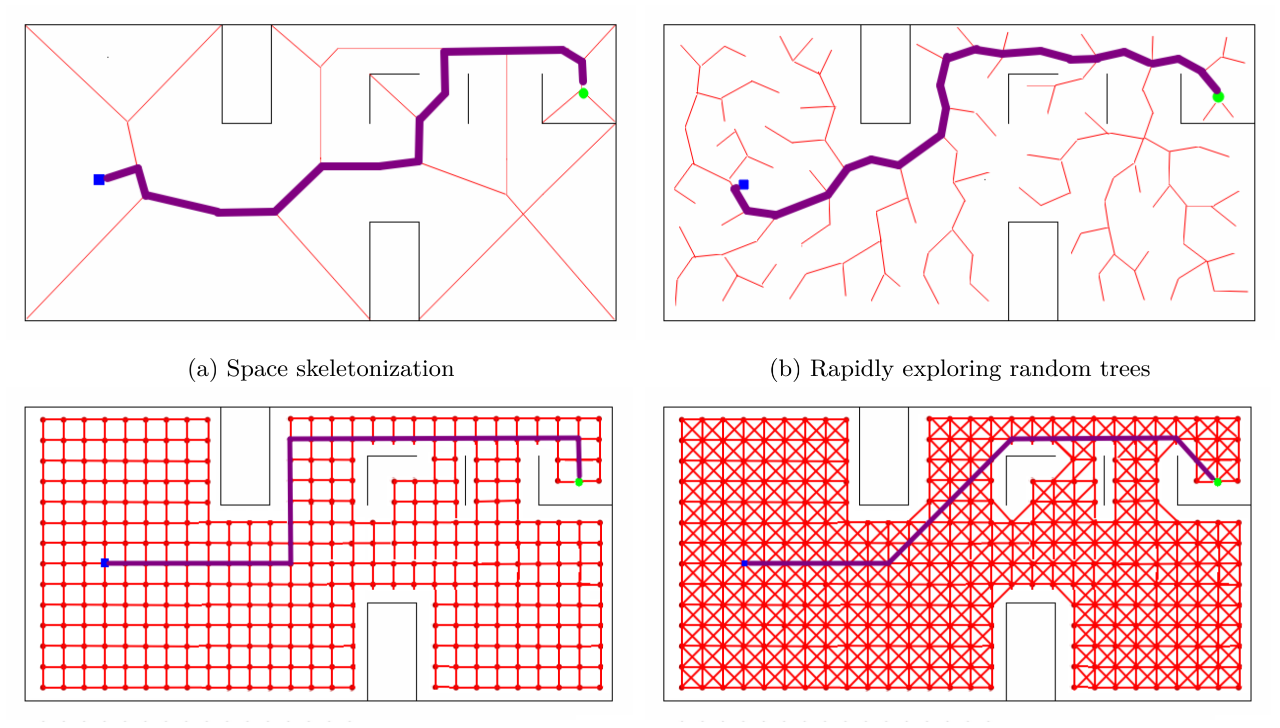
Several node distribution methods are available, each having their advantages and drawbacks. This section describes Space Skeletonization (SS), Rapidly exploring Random Tree (RRT) and Probabilistic Roadmap (PRM).

4.1.1 Space Skeletonization

SS finds an optimal route in a "skeleton", which is a continuous line which has an equal distance to the two closest obstacles. One way of deriving such a skeleton is using a Generalized Voronoi Diagram (GVD) [34]. In this case the obstacles are "inflated", meaning that the outer border of the object will be expanded at all sides by the same amount. During this inflation, the intersection between two different inflated obstacles is stored in the skeleton. The inflation will end when all free space points are covered by an inflated obstacle. The GVD is complete. However, chances are low that the initial and final pose are exactly on the GVD. Typically, the closest points from the initial and final pose to the GVD are computed, in this case respectively p_{in} and p_{out}. The final route will be the sum of the route from p_{init}, p_{in}, from p_{in} to p_{out} and from p_{out} to p_{final}. SS performs well in dense areas, where other techniques can struggle to find narrow openings between obstacles. The SS will simply find any opening. On the other hand, the performance of SS in low density maps is disappointing due to the large distances between the USV and the obstacles, which will cause large detours. An example of SS is shown in Figure 4.1a

4.1.2 Rapidly exploring random trees

The RRT [35] constructs a tree structure and each unoccupied point in the OGM is connected to the closest tree node as shown in Figure 4.1b. The initial point of the tree structure is an arbitrary unoccupied cell. Commonly used is the robots position because this cell is known to be unoccupied by obstacles. In order te create the tree, a random point p_{rand} in the map is selected and the closest point p_{near} to p_{rand} in the tree is defined. A new point p_{new} is defined on the line connecting p_{rand} and p_{near} at a distance in the range $[0, D_{max}]$ from



(a) Space skeletonization

(b) Rapidly exploring random trees

(c) Probalistic road map with uniform sampling in a 4-connected grid

(d) Probalistic road map with uniform sampling in a 8-connected grid

Figure 4.1: A visualisation of the three major node distribution methods. The green dot represents the initial position p_{init} and the blue square represents the goal p_{goal}

p_{near}. If p_{new} lies in unoccupied space of the map, it is added to the tree and another iteration step takes place until the final location can be connected to the tree. An advantage of the RRT is the possibility to easily add non-holonomic constraints [36] making it physically possible for non-holonomic ($\#DOF_{map} > \#DOF_{robot}$) robots like cars and boats without bow and/or stern propellers the follow the desired path. Different variants are available like RRT* [37] and Multiple-RRT.

4.1.3 Probabilistic road maps

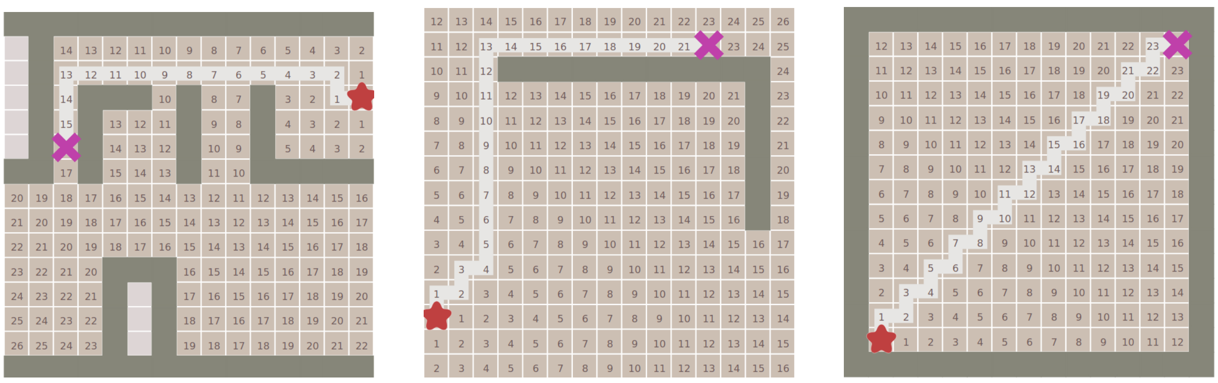
The PRM [38] method defines nodes as to be distributed around the map using spatial sampling techniques. The most straightforward method is *uniform spatial sampling* resulting in a grid of nodes at the free spaces. Other methods are *random space sampling*, *space sampling with Halton sequences* and *uniform incremental space sampling*. Since the map is a grid map, it is a natural and convenient choice to distribute the nodes in a grid as well, so this study adheres to uniform spatial sampling. When the nodes are distributed, collision free connections between the neighbors need to be determined. There is a choice to include only horizontal and vertical neighbours (at most four), as depicted in Figure 4.1c or to include diagonals (at most eight neighbours) as depicted in Figure 4.1d. Since the latter will generally lead to shorter paths, it is preferred. Even more connection could be created but this will lead into highly sophisticated maps where computational power will increase heavily. A smoothing approach like explained in Section 4.4 is preferred over increasing the number of nodes even more.

4.2 Finding the shortest path

Next the shortest path in the node graph from Section 4.1 has to be calculated. This section will focus on four different path planning algorithms: Breadth first search (BFS), Dijkstra, Greedy best first search (GBFS) and A*. All described algorithms use the frontier which is the collection of nodes which are adjacent to an unvisited node but which have been visited themselves. Every algorithm will return a structure which tells for every node which was the previously visited node. This structure can be turned into a path using the strategy as shown by the following pseudo-code.

```
current = goal
while current not is start:
  path.append(current)
  current = came_from(current)
```

4.2.1 Breadth First Search



(a) L: 16 D: 153 (b) L: 22 D: 206 (c) L: 24 D: 196

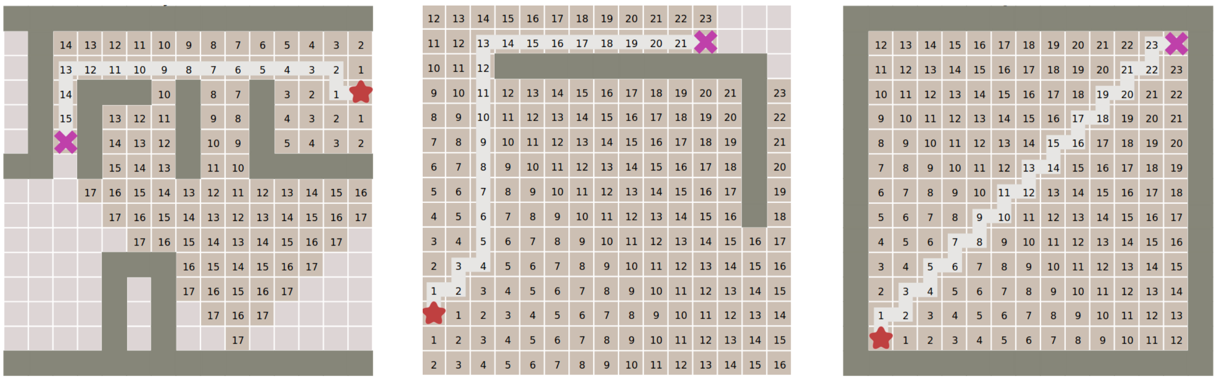
Figure 4.2: BFS. the number in the graph represent the lenghth of the path to the start from the node at the location of the number. L denotes the length of the path and D denotes the total number of discovered cells. Every path is a shortest path because BFSin guarenteed to find a shortest path.

BFS is the simplest path planning algorithm of the four algorithms discussed here. Dijkstra, GBFS and A* are all extensions based on the BFS principle. The BFS algorithms starts at the starting point on the map and expand from there like a wave front until the whole map is covered. BFS will pick out the node from the frontier which has been in the frontier longest and call this the current node. When adding the new nodes to the tail of the frontier, the node which is first in line will be the node which has been in the frontier longest. BFS first finds the neighbors of the current node. If these nodes have never been the current node before, they will be added to the frontier. Also they are given the current node as its source. The final path can be found by the pseudo-code as shown in the introduction of this section. Due to the grid map, the distance between the nodes is constant. The length of the paths equals the number of nodes in the paths multiplied by the distance between the nodes. Therefor only four-neighbor patterns can be applied in a grid map. Eight-neighbor grids result in different distances between nodes. BFS guarantees to find the shortest path. The pseudo-code for BFS is shown below.

```
frontier = {origin}
came_from = {}
```

```
while length(frontier) not is 0:
  current = get_and_remove_first_value(frontier)
  for neighbor in get_neighbors(current):
    if neighbor not in came_from:
      append_to_frontier(neighbor)
      came_from[neighbor] = current
```

4.2.2 Dijkstra



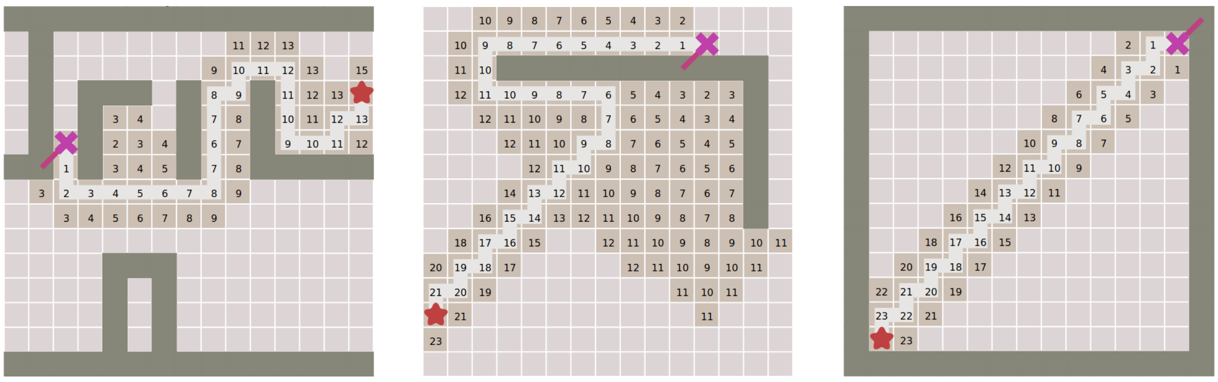
(a) L: 16 D: 106 (b) L: 22 D: 199 (c) L: 24 D: 196

Figure 4.3: Dijkstra. the number in the graph represent the cost of the node at the location of the number. L denotes the length of the path and D denotes the total number of discovered cells. Dijkstra is shown to show an optimal path in all three cases.

Dijkstra's algorithm is an extension to the BFS as abovementioned. BFS calculates the length of the path by summing the number of nodes between a path and multiply this by a constant distance between the nodes. Dijkstra does not assume a uniform spacing of the nodes. Where BFS investigates the nodes with the shortest path length, Dijkstra investigates the node with the lowest cost. For example for road navigation Dijkstra cost could be taken as the length of the road divided by the speed limit. In this case the fastest route can be calculated instead of the shortest. This means that the shortest route might not be the optimal one. Also patterns with variating node distances like eight neighboring grids could be used when Dijkstra is applied. In the specific case applied in Figure 4.3 Dijkstra's algorithm is implemented to stop when the goal is reached where BFS is implemented to complete the whole map. The number in the graph represent the cost at the location of the number. The cost is calculated with a constant distance of 1 between the nodes. Dijkstra's algorithm guarantees a shortest path. Like BFS, Dijkstra will return a cost map which could be used for gradient path planning rather than grid path planning. Gradient path planning is performed by following the slope a the cost map whereas grid path planning is limited to connecting the nodes of the map. Therefore Dijkstra is capable of planning smoother paths than algorithms which lack a costmap.

```
frontier = {origin}
came_from = {}
path_cost = {}
while length(frontier) not is 0:
  current = get_and_remove_lowest_cost_value(frontier)
  for neighbor in get_neighbors(current):
    if neighbor not in came_from:
      path_cost[neighbor] = path_cost[current] + cost_between_nodes(current, neighbor)
      append_to_frontier(neighbor)
      came_from[neighbor] = current
```

4.2.3 Greedy Best First Search



(a) L: 25 D: 55　(b) L: 32 D: 117　(c) L: 24 D: 48

Figure 4.4: GBFS. the number in the graph represent the heuristic of the node at the location of the number. L denotes the length of the path and D denotes the total number of discovered cells. GBFSis shown to be able to find the shortest route very fast like in Figure 4.4c. However optimality is not guarenteed as shown in Figures 4.4a and 4.4b

GBFS is at first sight almost identical to Dijkstra's algorithm. The only difference is that GBFS first expands the node closest with the lowest heuristic where Dijkstra first expands the node will the lowest cost. The heuristic is the distance to the goal, not taking obstacles into account. GBFS can be really fast and efficient as Figure 4.4c shows. GBFS requires almost always less time then Dijkstra and BFS because less nodes are being visited. BFS and Dijkstra expand like a wave front while GBFS expands straight in the direction of the goal. This fast method comes with a drawback however. When the algorithm is expanding rapidly in the direction and there seems to be an obstacle in the way, the path has to be detoured like in Figure 4.4b. This result in a route which is not guaranteed to be optimal.

```
frontier = {origin}
came_from = {}
while length(frontier) not is 0:
  current = get_and_remove_lowest_heuristic_value(frontier)
  for neighbor in get_neighbors(current):
    if neighbor not in came_from:
      insert_to_frontier(neighbor)
      came_from[neighbor] = current
```

4.2.4 A*

A* combines both the benefits of Dijkstra and GBFS. A* is guaranteed to find an optimal path while exploring less nodes than Dijkstra does. A* first expands the node in the frontier with the lowest value of priority

$$\text{priority} = \text{heuristic} + \text{cost} \tag{4.3}$$

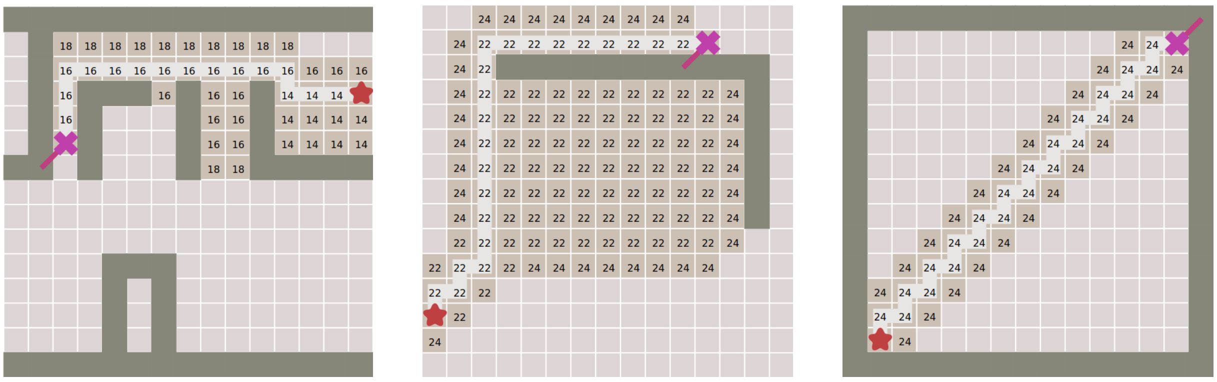

(a) L: 16 D: 46 (b) L: 22 D:124 (c) L: 24 D: 48

Figure 4.5: A*. the number in the graph represent the sum of cost and heuristic at the location of the number and the distance from the goal at the location of the number. L denotes the length of the path and D denotes the total number of discovered cells. Every path is a shortest path because A* in guarenteed to find a shortest path.

A* is proven to find an optimal path without exploring the complete node graph. This is beneficial when computing power is limited.

```
frontier = {origin}
came_from = {}
path_cost = {}
while length(frontier) not is 0:
  current = get_and_remove_lowest_priority_value(frontier)
  for neighbor in get_neighbors(current):
    if neighbor not in came_from:
      path_cost[neighbor] = path_cost[current] + cost_between_nodes(current, neighbor)
      insert_to_frontier(neighbor)
      came_from[neighbor] = current
```

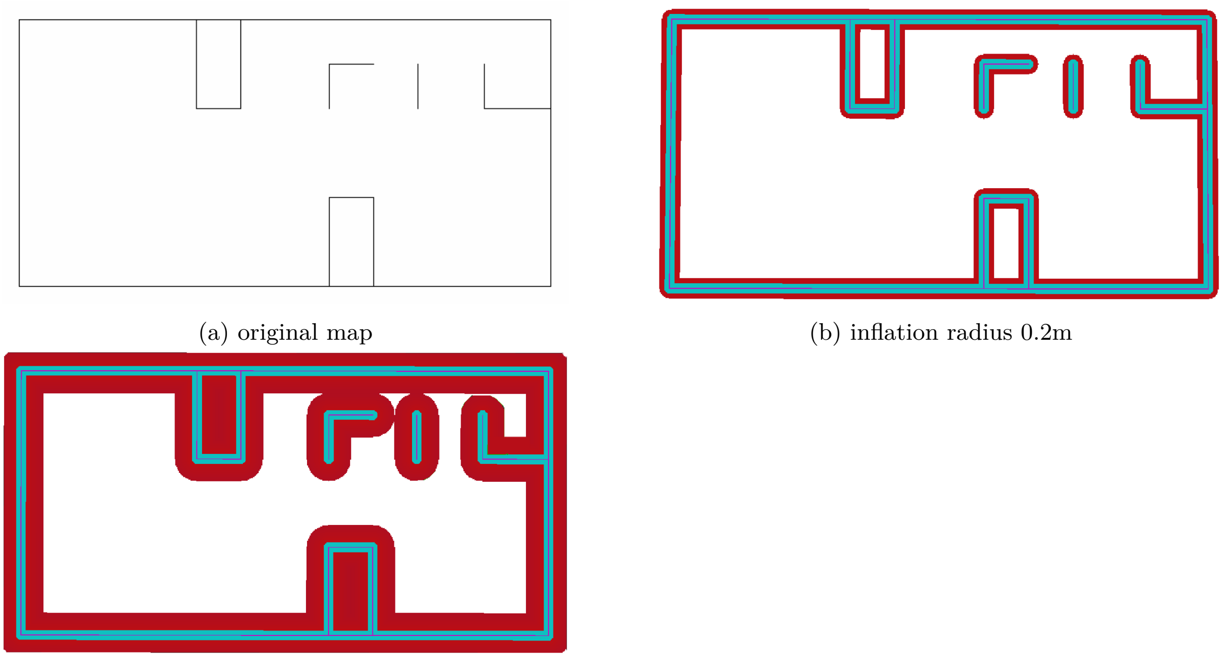
4.3 Map inflation

Map inflation is performed in order to create a safety area around the obstacles. The cells close to an object are given a higher cost to cells further away. This is done to compensate for the size of the USV. The USV will follow the path with its center. This means that the minimum distance between the path and an obstacle should be at least

$$\text{distance}_{min} = \text{size}_{USV} + \text{safety margin}. \tag{4.4}$$

The size of the USV could be interpreted in different ways. For circular USV the size would be the radius, but the shape of our USV is orientation dependent. The size is assumed to be the width of the USV as the USV is generally assumed to move in longitudinal direction. Figure 4.6 shows an example of an infated map. The free area (white) of the inflated map is smaller the the orginal map.

The inflation radius influences the planned path. Figure 4.7 shows two different paths constructed by the same Dijkstra algorithm but in maps which are inflated by a different inflation radius. Some parts will become inaccesible but following the path will less likely result in collision due to larger distances between the path and


(a) original map

(b) inflation radius 0.2m

(c) inflation radius 0.5m

Figure 4.6: (a) original and (b) inflated map. The purple inner area represents the obstacles in the map. The light blue middle area represents the extension of the obstacles in which the USV is still colliding with the obstacles, and the outer red area represents the safety boundary.

the obstacles. In this thesis an inflation radius of 0.2m is chosen referring to the length of the vessel. This is said to be a safe passing distance according to [39].

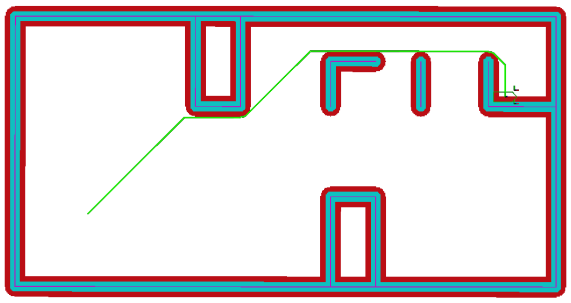
4.4 Gradient path

grid path approaches as shown in the previous sections will result into unsmooth paths because their direction are limited to either vertical, horizontal or diagonal direction. The path could become even shorter when it is not limited to the eight directions. A way of providing shorter paths is to use the costmap as created by the planning algorithm for gradient descent path planning [40]. The costmap can be transformed into the continous domain usign quadratic approximations as shown by [41].

The costmap transformation is performed using interpolation. For a grid cell the neighbor's cost $cost_A$, $cost_B$, $cost_C$ and $cost_D$ are defined as shown in Figure 4.8. Point A and B share an axis and point C and D share the other axis. The object is to find an interpolated cost for the current grid cell based on the surrounding cost values. The following assumption are made without losing generality

$$cost_A < cost_B \quad \text{and} \quad cost_C < cost_D \quad \text{and} cost_A < cost_C. \tag{4.5}$$

The next step is to calculate whether it is shorter to traverse through the grid cell or travel around. Traversing through the cell is only shorter when


(a) inflation radius 0.2m

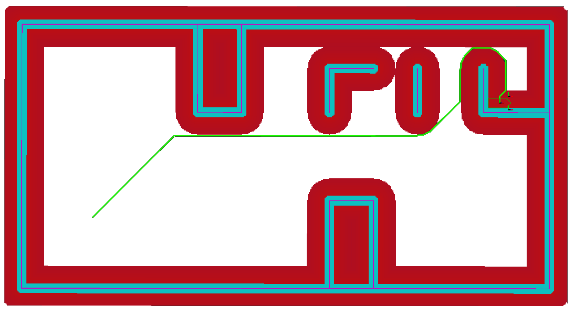

(b) inflation radius 0.5m

Figure 4.7: A path planned using Dijkstra in a map with (a) 0.2m inflation radius and (b) 0.5m inflation radius leads to a different path planning outcome

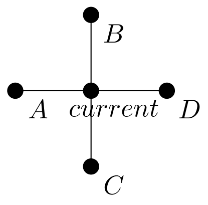


Figure 4.8: Definition of neighbor's cost for quadratic approximation

$$cost_{old} < cost_A + h \tag{4.6}$$

where h refers to the size of the grid cell. If Equation (4.6) holds, $cost_{new}$ can be calculated by solving

$$(cost_{new} - cost_A)^2 + (cost_{new} - cost_C)^2 = h^2. \tag{4.7}$$

The cost is calculated instead of the original cost function as used by the Dijkstra algorithm. In practice this equation will be solved by a taylor approximation to avoid the computationally expensive square root which is required for solving Equation (4.7). Gradient descent path planning in the continuous domain generally results into shorter paths than post-planning path smoothing because its movement choices are made at a more fundamental level. An example of a gradient path is shown in Figure 4.11.

Table 4.1: Path length and number of discovered cells for 4-connected grids for BFS, Dijkstra, GBFS and A*

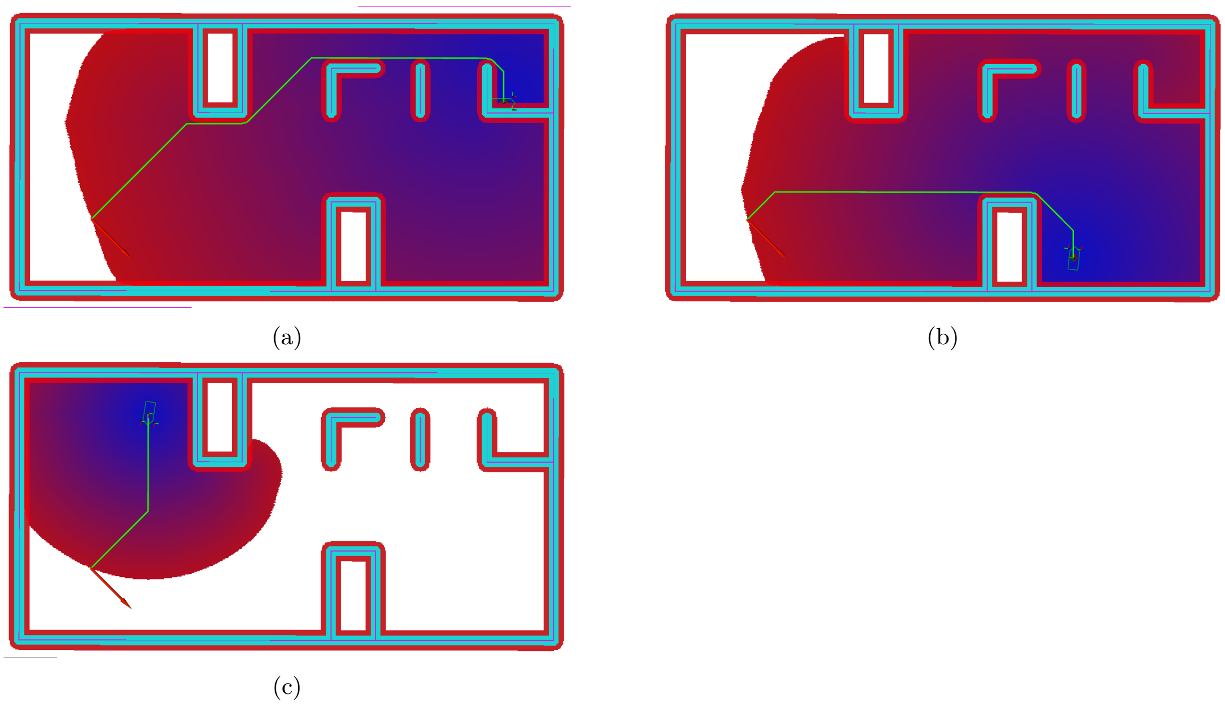
	Path length				**Discovered cells**			
	map 1	map 2	map 3	average	map 1	map 2	map 3	average
BFS	16	22	24	21	153	206	196	185
Dijkstra	16	22	24	21	106	199	196	167
GBFS	24	32	24	27	55	117	48	73
A*	16	22	24	21	46	124	48	73

Table 4.2: Path length for Dijkstra, A* and gradient descent in a smoothed costmap from Dijkstra

	Path length			
	case 1	case 2	case 3	**average**
Dijkstra	11.32	8.22	3.87	**7.81**
A*	11.32	8.22	3.87	**7.81**
Gradient	10.96	7.99	3.67	**7.54**

4.5 Results

Several path planning methods have been proposed in this chapter. The results from Figures 4.2 to 4.5 are summarized in Section 4.5. The length of path shows that BFS, Dijkstra and A* show that produce equal paths which is as expected as they are proven to provide optimal results. GBFS generates an optimal path for the third case only. BFS discovers the most cells shortly followed by Dijkstra. GBFS does not always discover the least number of cells. In the first map A* outperforms GBFS in the sense of number of discovered cells.



(a) (b) (c)

Figure 4.9: Path planning performed using Dijkstra performed from 3 different starting positions in a 8-connected grid. The gradually coloured area refers to the costmap produced by Dijkstra's algorithm.

Figure 4.9 Shows different maps produced by applying Dijkstra's algorithm in a map from different starting position towards the same goal. The colored area inside the map represents the costmap of processed points representing the distance to reach that point starting from the USV position. Figure 4.10 Shows the same situation but now applying A* for path planning. The main difference one could observe is that the costmap for A* contains significantly less point than the costmap constructed by Dijkstra. Figure 4.11 shows again the same starting points and the Dijkstra algorithm results. However a gradient path in the potential field is applied for path planning instead of the grid path as used before. This results in a smooth path which is not limited to horizontal, vertical of 45° angle segments like the 8-connected grid paths. The path is easier to follow, and even

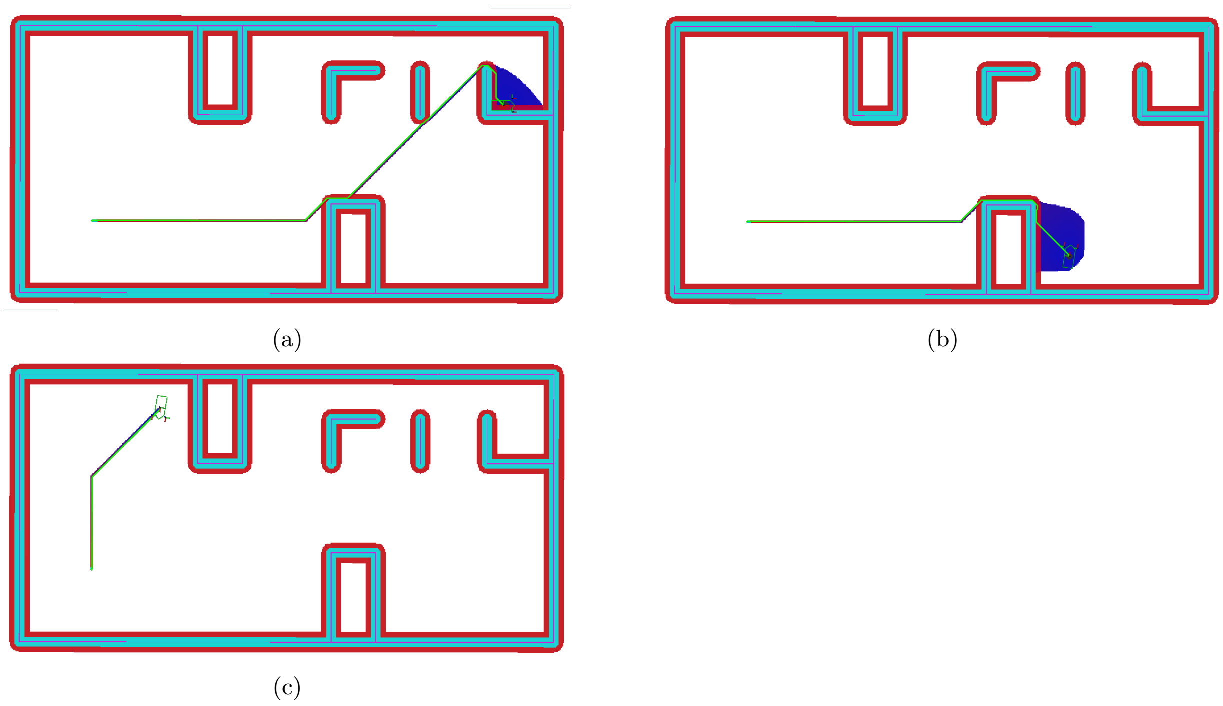


Figure 4.10: Path planning performed using A* performed from 3 different starting positions. The gradually coloured area refers to the costmap of the heuristic. The costmap appears around the starting point and around the path but is not as extended as the grid map from Dijkstra

shorter.

4.6 Discussion

In order to apply path planning a sailable area has to be defined by inflation of the map which is acquired by the mapping algorithm. The minimum inflation radius equals the width of the USV. When this condition is not met, the USV will collide. For extra safety and maneuverability a safety margin is added to the inflation radius. The size of the safety margin determines the sailable area and is in this case set to 0.2m. For path planning in this sailable area either Dijkstra and A* look promising. Both algorithms are proven to return an optimal path. A* will calculate the path faster and with less computational resources. Dijkstra will return a complete costmap applicable for planning a gradient path rather than a grid path. A gradient path can only be applied when Dijkstra's algorithm is used because a complete costmap or potential field is required. A* does not return a complete costmap or potential field which makes it unapplicable for a gradient path. The gradient path method produces the shortest path and is therefor prefered.

4.7 Conclusion

Nodes have to be distributed around the map in order to plan a path. Different node sampling techniques are available of which PRM is chosen due to is widespread applicability. As computation power does not seem

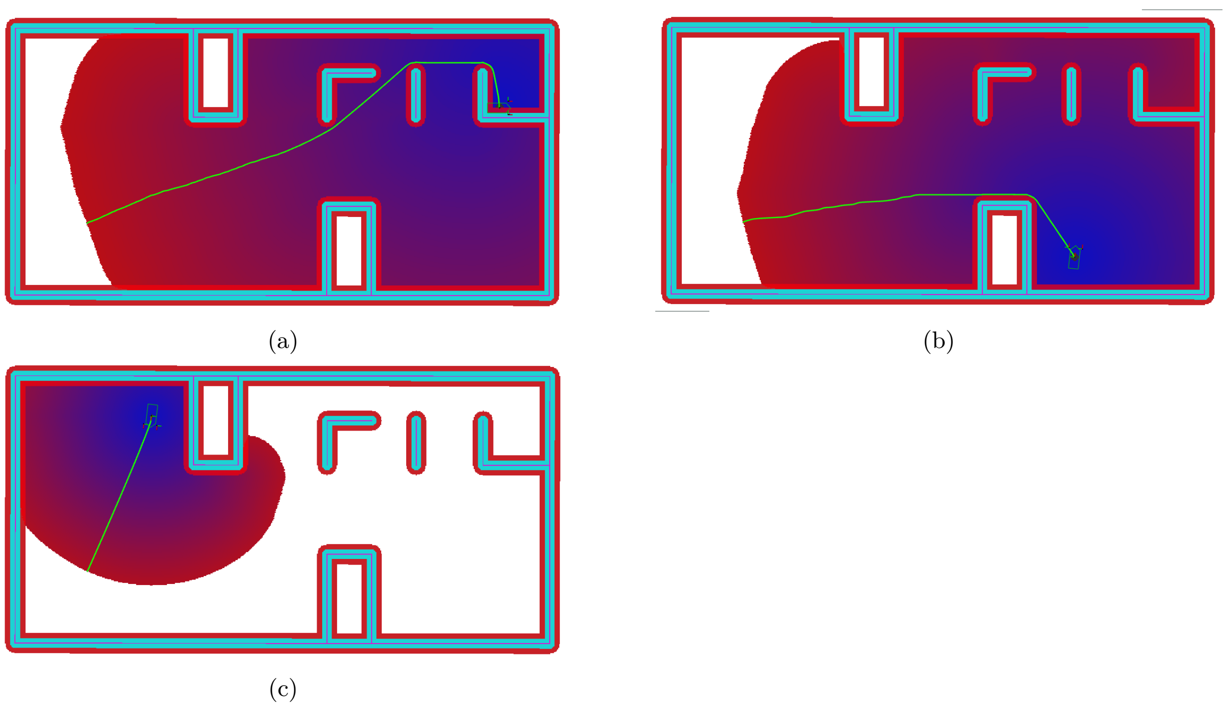


Figure 4.11: Path planning performed using Dijkstra performed from 3 different starting positions. This time the path is obtained by following the steepest descend in the gradient field or costmap. This is only possibly for Dijkstra due to its extensive costmap.

to be the limiting factor in the path planning algorithm, Dijkstra is chosen as the path planner. Its costmap is suitable for gradient planning rather than grid planning, which might increase the performance of the final system. The gradient path is smoother and therefore easier to follow and shorter than the original because it is not limited to the 8-connected grid with its only horizontal, vertical and 45° angle segments.

Chapter 5

Control

Chapter 4 describes how to plan a path through a map with obstacles. The next step is to establish a path following (PF) algorithm that determines the actuation of the Unmanned Surface Vessel (USV) such that it will follow the path. PF can be seen as the substition for the human driver who follows the route from a navigation system. Instead of the human, now the PF is steering the rudder and handling the throttle. PF control is a research field on its own, where, depending on the desired bandwidth and possible nonlinearities, rather advanced algorithms have emerged like adaptive control [16] and model-predictive control [42]. Many more examples exists. For our USV, we may argue a proportional control law is sufficient. However, our system is expected to be subject to either systematic errors or external disturbances, such as (imbalanced) load, rudder offset or wind. The proportional controller is therefore extended with integral action, such that it can cope with steady-state error.

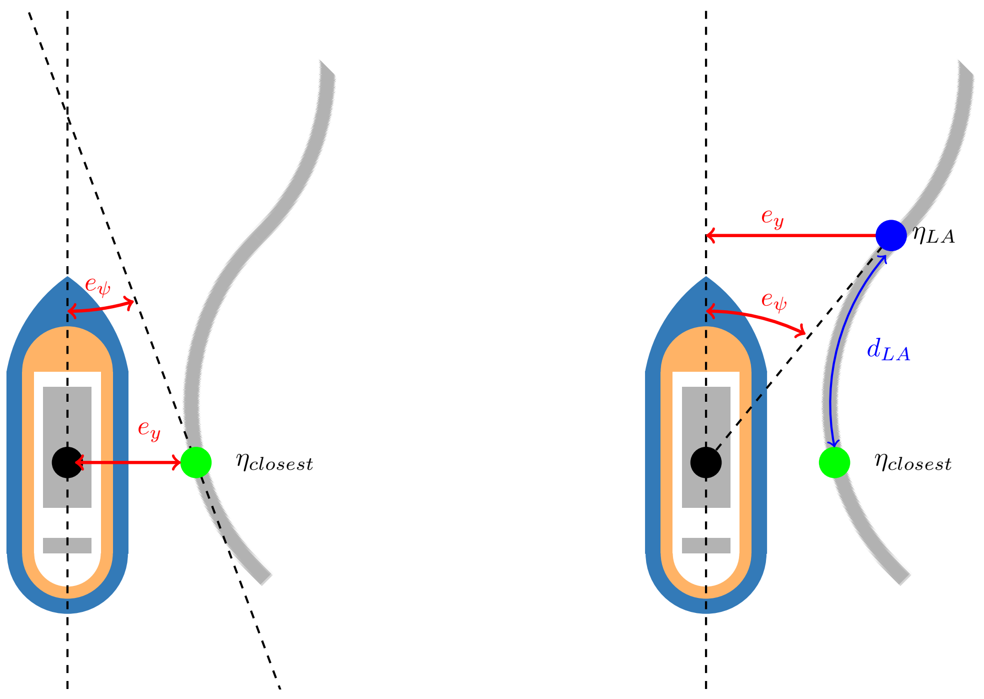
This chapter will first define the error signal, and subsequently discuss the design of a PI controller for an artificial, block-shaped path. The 'Results' section provides an example with an actual path from the path planner as introduced in Chapter 4. The control is said to perform well if the Root Mean Square (RMS) error is less than the vessel size according to [39].

5.1 Positional control approaches

A controller controls a robot or model by applying a control law based on the given error. Before a controller can be applied, this error has to be defined. Two options are investigated: *Position control* and *Look ahead control*. For both alternatives, longitudinal error is not accounted, but velocity is controlled based on a reference velocity. The lateral and rotational error drive the steering controller.

5.1.1 Position Error

Position error is the simplest method for determining the error between the position of the USV and the path. The point on the path which is closest to the USV is determined by a minimization algorithm, or (in this work) an exhaustive search. The distance to the USV is computed for every point on the path and the point with the



(a) The error used for position error control is the difference between the position of the USV and the closest point on the path.

(b) The error used for look ahead control is the angle between the heading of the USV and the heading required to reach the *look ahead* point in a straight line from the current position.

Figure 5.1: Definitions of position error (**a**) and look ahead error (**b**) which may be used for positional control and/or path following.

lowest error is labeled as closest point $\eta_{closest}$. The errors e_y and e_ψ are defined as the distance between this closest path point and the USV position as depicted in Figure 5.1a:

$$\boldsymbol{e} = \begin{bmatrix} e_x \\ e_y \\ e_\psi \end{bmatrix} = \boldsymbol{\nu_{closest}} - \boldsymbol{\nu_{USV}}. \tag{5.1}$$

Position error control works well on a fully actuated USV with enough power to actuate all DOF. In practice position error control is mostly used for simpler/slower reference tracking situations such as dynamic positioning. Vessels designed for longitudinal motion demand greater lateral motion actuation power, due to higher lateral hydrodynamic drag. Here, position control is applied when the USV has (almost) reached the goal and has to stay at the goal.

Table 5.1: Root mean square error for following the block path for different look ahead distances

look ahead [m]	$error_y$ [m]	$error_\psi$ [rad]
0.07	0.12	0.45
0.15	0.11	0.42
0.25	0.05	0.29
0.35	0.04	0.26
0.50	0.14	0.42

5.1.2 Look ahead error

Path following uses future information of the path to anticipate on tight corners, and can therefore better cope with the relatively high inertia of the USV. The look ahead error approach as depicted in Figure 5.1b looks ahead with a distance d_{LA} at the section of the path to come. For look ahead error control again a closest point has to be determined similarly to the position error approach. From $\eta_{closest}$ a look ahead point η_{LA} is defined at a given look ahead distance d_{LA} (on the path) from $\eta_{closest}$. The angular error between the heading of the USV and the heading required to reach this look ahead point is taken as the rotational reference error, such that the complete error vector becomes

$$\boldsymbol{e} = \begin{bmatrix} e_x \\ e_y \\ e_\psi \end{bmatrix} = \begin{bmatrix} x_{LA} - x_{USV} \\ y_{LA} - y_{USV} \\ \cos^{-1}\left(\frac{y_{LA}-y_{USV}}{d_{LA}}\right) \end{bmatrix}. \tag{5.2}$$

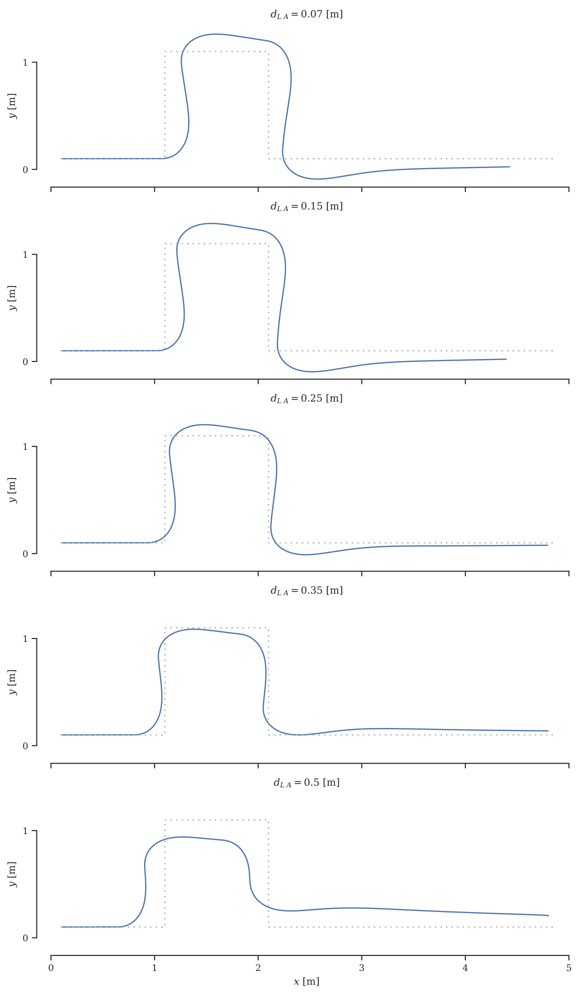
The lateral control input is the lateral error between the USV and η_{LA} in the USV frame. Look ahead control applies a torque τ_ψ to steer the USV towards the path and applies lateral force τ_y to compensate for small errors in e_y. Because of the suitability for path following, look ahaed error control is applied for the path following applications in this thesis.

5.1.3 Look ahead distance

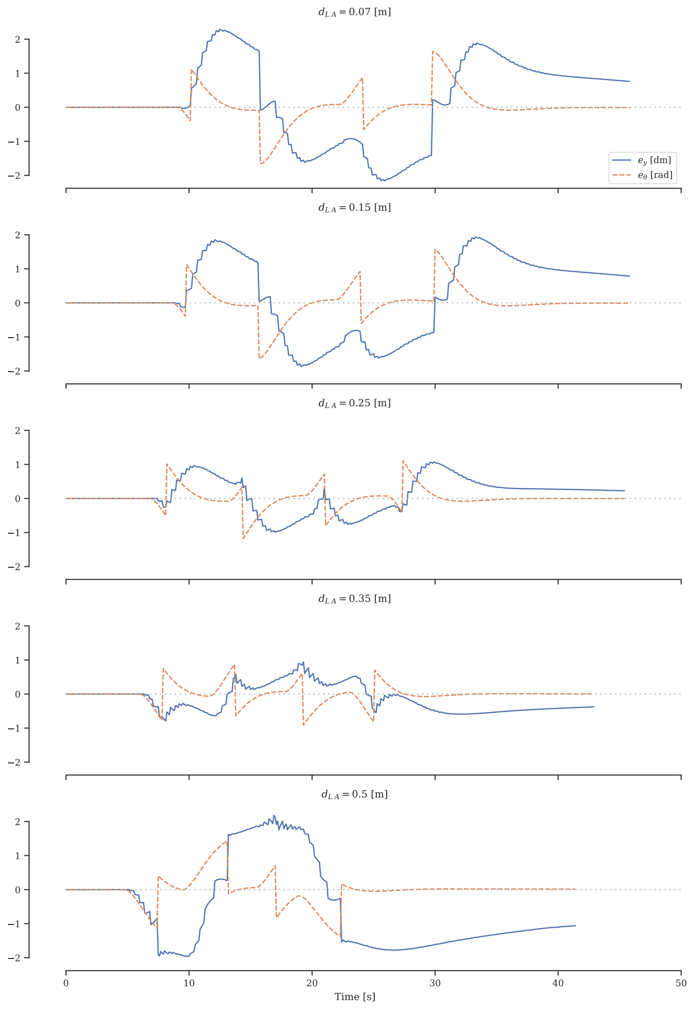
If the controller is based on the future path the look ahead distance d_{LA} has to be determined. Control results will vary based on the look ahead distance, as it changes the magnitude of the error by definition. Low d_{LA} will result in little cutting corners but large overshoot while high d_{LA} results in the opposite. An empirical trade-off is to be made, and this requires some control law already. In this study, the proportional controller and corresponding d_{LA} have been synthesized in a couple of iterations. The controller used to determine the best look ahead distance will be discussed in Section 5.2.

As a running example, the control design experiments use a block shaped reference path. This path is designed such that the controller needs to react on left and right corners following each other. The angles are 90°, something that shouldn't occur in the eventual setup-up, but is now useful to mimic a step reference and analyse overshoot and overdamping. The block shape ends with a long straight to evaluate how well the system converges from error.

Figure 5.2a shows the sailed path for different values of d_{LA}. In this case a look ahead distance of $0.25m$ is preferred, because it results in very low error and best convergence after all corners are taken. The look ahead



(a) Sailed paths



(b) Error signals

Figure 5.2: Sailed path and error signals for different setting of look ahead distance d_{LA}. The dashed line represents the reference trajectory, which is unfeasible for the sake of this discussion.

distance of $0.35m$ also results in very low errors but does perform worse on converge to goal at the final part and it produces more aggresive occilations. For evaluation purposes, the positional error signals (as defined in Figure 5.1a) are depicted in Figure 5.2b.

The corresponding control signals for the same five d_{LA} settings are depicted in Figure 5.3. Note that these are not proportional to the positional errors (Figure 5.2b), but instead based on the look ahead error. Along with the other quantities, the control signal converges best for $d_{LA} = 0.25$. The jagged line is an artefact of coarse path descretization. The controller with lower d_{LA} is more sensitive to coarse path discretization than the controller with high d_{LA} because of a trigonometric effect: the errors have a different magnitude because of the different length. One can compensate for this effect by scaling the proportional gain. Here, we just assess the timing (anticipation), and use a fixed gain. It is observed that an increased d_{LA} results in a shift of the graph to left as the same position on the path is earlier detected. For lower d_{LA}, the USV starts steering too late. From here on we continue using $d_{LA} = 0.25$.

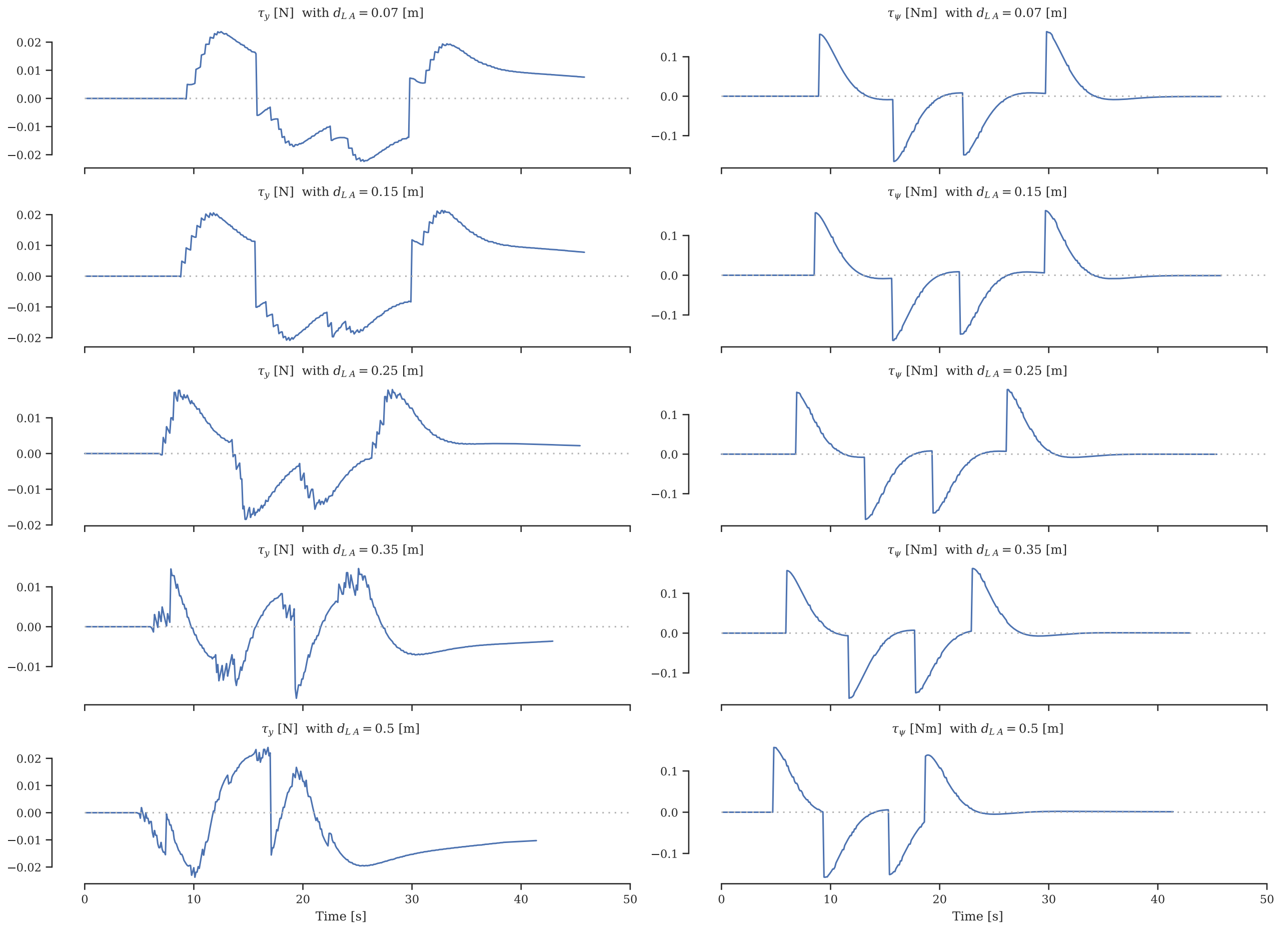


Figure 5.3: Control inputs for the different settings of look ahead distance d_{LA} as depicted in Figure 5.2a.

Table 5.2: Maximum applicable forces and torques of the USV.

F_x	N/A
F_y	1.5 N
T_z	0.53 Nm

5.2 Proportional control

The proportional controller commonly known as P controller calculates the USV actuator input as a function of the error between the USV position and its desired path according to Section 5.1.2. The control input τ is obtained by

$$\boldsymbol{\tau} = \boldsymbol{P}\boldsymbol{e}. \tag{5.3}$$

where P represents the control gain and τ the applied force on the USV:

Table 5.3: Root mean square error in meters for following the block path for different values of P_y and P_ψ

P_ψ / P_y	0.05	0.10	0.20
0.01	0.18	0.06	0.13
0.10	0.18	0.04	0.11
1.00	0.09	0.03	0.07

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_\psi \end{bmatrix} \tag{5.4}$$

$$\boldsymbol{\tau} = \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix}. \tag{5.5}$$

Here F_x and F_y are the forces applied in N in the surge (longitudinal) and sway (lateral) direction in the USV frame respectively. M_z is the torque applied in Nm around the vertical axes of the USV. The control input will thus increase as the error increases, but only until the maximum force or torque which is physically applicable by the actuators is reached. These actuator limits are summarized in Table 5.2.

Too strong proportional gain will result in agressive fluctuation in the input τ which might be physically unachievable by the actuators. Also oscillatory motion can then occur. Weak proportional gain will result in larger errors due to insufficient actuation. Tuning is therefore crucial. The controller is tuned by evaluating the error as defined in Figure 5.1a and USV input τ when following the block shaped path.

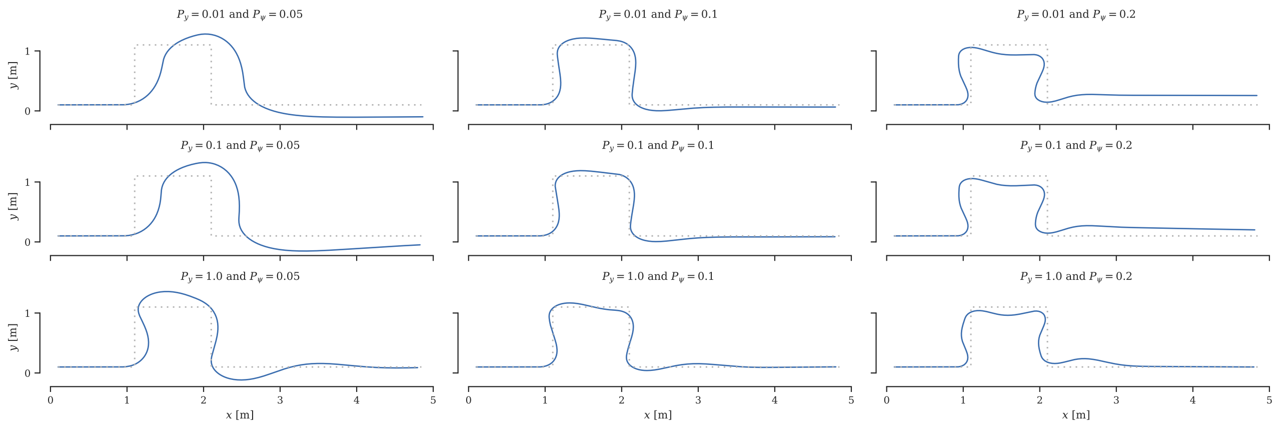


Figure 5.4: Sailed trajectories for proportional control with different lateral gains P_y and rotational gains P_{psi}.

Figure 5.4 shows the sailed path of an USV trying to following a block shaped path using a proportional controller. The middle figure represents the best controller. Lowering the P_y results in larger error whereas increasing the P_y results in overshoot and high lateral input. The effect of tuning P_t is likewise, but stronger because the model is more sensitive to rotational error than lateral error. Table 5.3 shows the RMSerros for the different settings of the proportional gains corresponding to the figures. The mid bottom produces the minimal error but however results into overshoot. The difference in RMSerror is minimal but the middle setting result into less overshoot and is therefor preferred.

The errors between the given path and the sailed trajectory are depicted in Figure 5.5.

The input signal τ_y is depicted in Figure 5.6 and the input signal τ_ψ is depicted in Figure 5.7. Observed is that

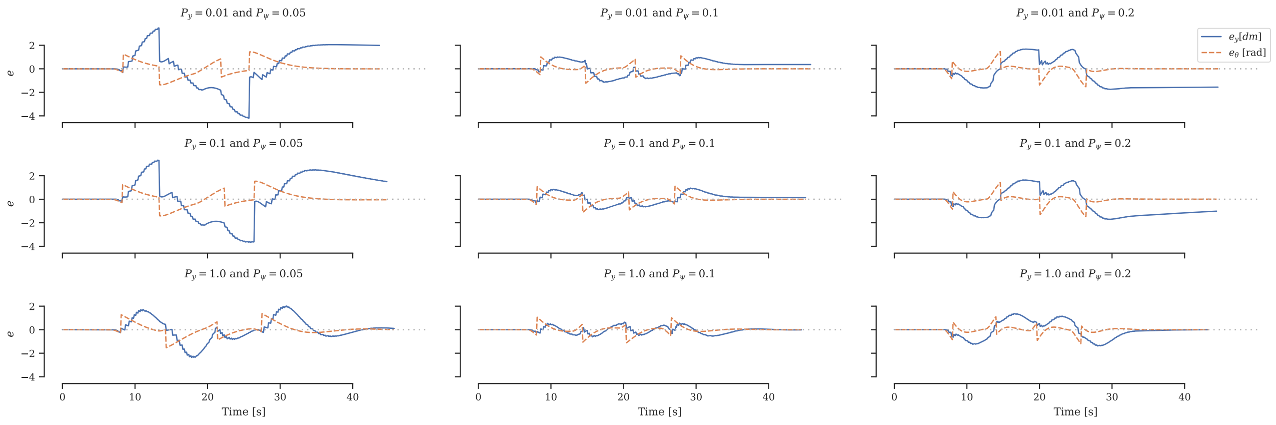


Figure 5.5: Positional errors for variations on lateral gain P_y and rotational gain P_t.

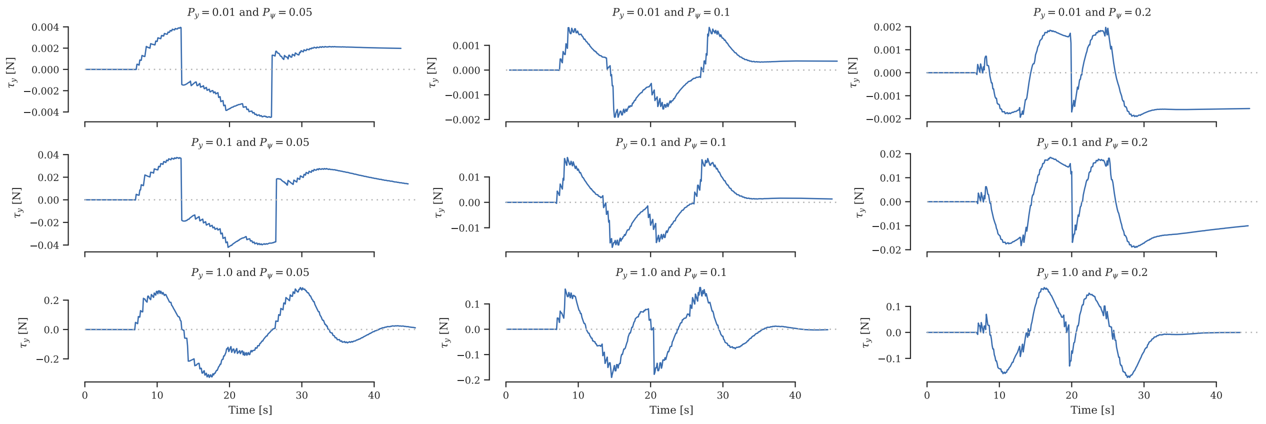


Figure 5.6: Control signal τ_y for different choices of proportional gains. Note that the y-axis has a different scale in each plot.

the shape of τ mainly depends on P_ψ and that an increase in P_y will lead into a scaling of this shape for τ_y. τ_ψ seems to barely depend on P_y.

5.3 Integral control

The integral I part of the controller controls the USV based on the cumulative error. Integral control is mostly useful for systematic disturbances that cause steady state errors. Examples of such disturbances are an unbalanced weight distribution, current or wind. The control law of a proportional controller extended with an integral part is defined by

$$\boldsymbol{\tau} = \boldsymbol{P}\boldsymbol{e} + \boldsymbol{I}\int \boldsymbol{e}dt. \tag{5.6}$$

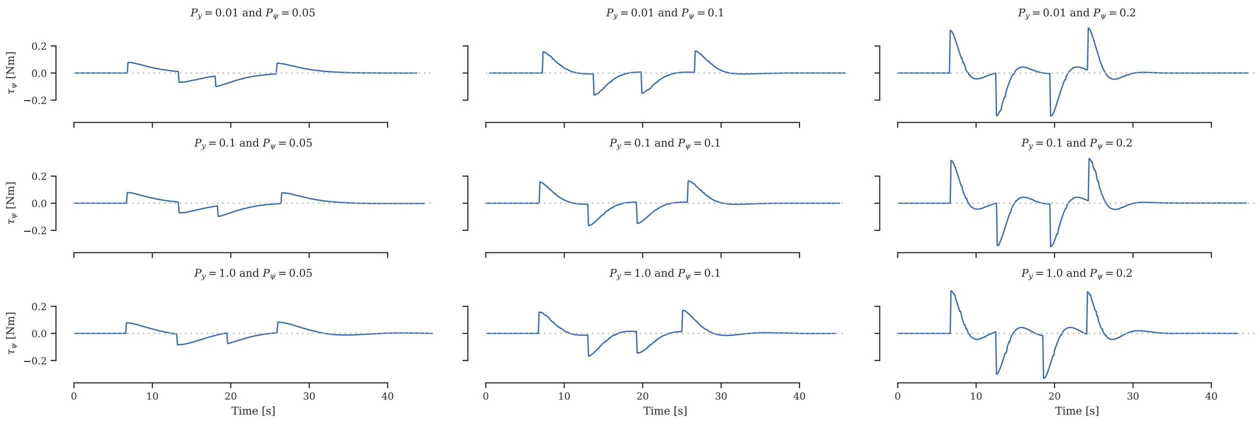


Figure 5.7: Control signal τ_ψ for different choices of proportional gains.

where

$$\boldsymbol{I} = \begin{bmatrix} I_x \\ I_y \\ I_\psi \end{bmatrix}. \tag{5.7}$$

There are various ways to find the right I values. For simple cases, as demonstrated here, the gain can slowly be raised until the performance is satisfactory. This tuning process is similar to what was discussed in the Section 5.2. For low I values, little happends compared to regular P-control. For greater I, the integral action can harm the initial performance and might lead to unstability. As is illustrated in Figure 5.8, an example without disturbance, greater integral gain will decrease reference tracking performance. To keep the discussion short, the evaluation of τ and the errors are not reported for this case.

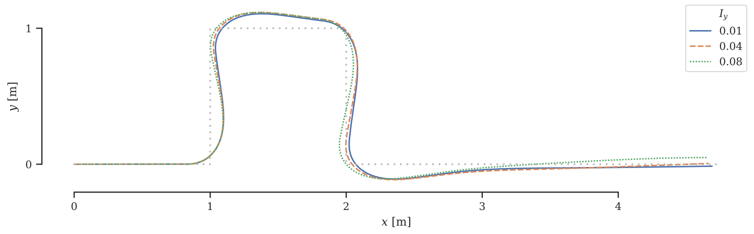


Figure 5.8: Sailed trajectories for different integral gains in a situation without disturbance.

Let us now look at disturbance rejection. A disturbance E is added in the boat model by modifying the boat model from Equation (2.6) to

$$\mathrm{M}\dot{\boldsymbol{\nu}} + \mathrm{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathrm{D}\boldsymbol{\nu} + \boldsymbol{E} = \boldsymbol{\tau}. \tag{5.8}$$

with

$$\boldsymbol{E} = \begin{bmatrix} E_x \\ E_y \\ E_\psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0.005N \\ 0 \end{bmatrix}. \tag{5.9}$$

In Figure 5.9, the top plot shows the performance of a P controller when constant external disturbance E applies. The bottom plot shows the same situation controlled by a PI controller with integral action in the lateral dimension. It is observed that the compensation needs to build up, and eventually rejects the disturbance. This is seen by the fact that the two lines in the top plot diverge when approaching the goal whereas the two lines converge in the bottom plot.

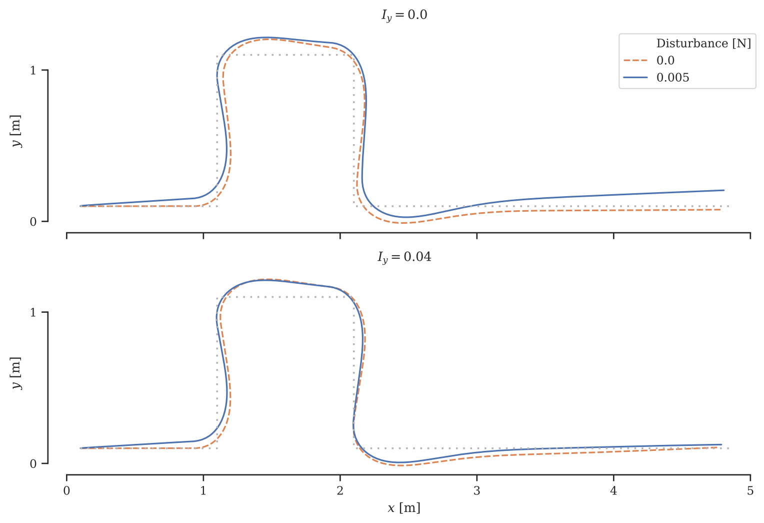


Figure 5.9: Sailed trajectories for the original controller (top) and for the same controller with integral action (bottom). The dashed (orange) line corresponds to no disturbance, and the solid (blue) line is for the boat subject to a lateral disturbance force.

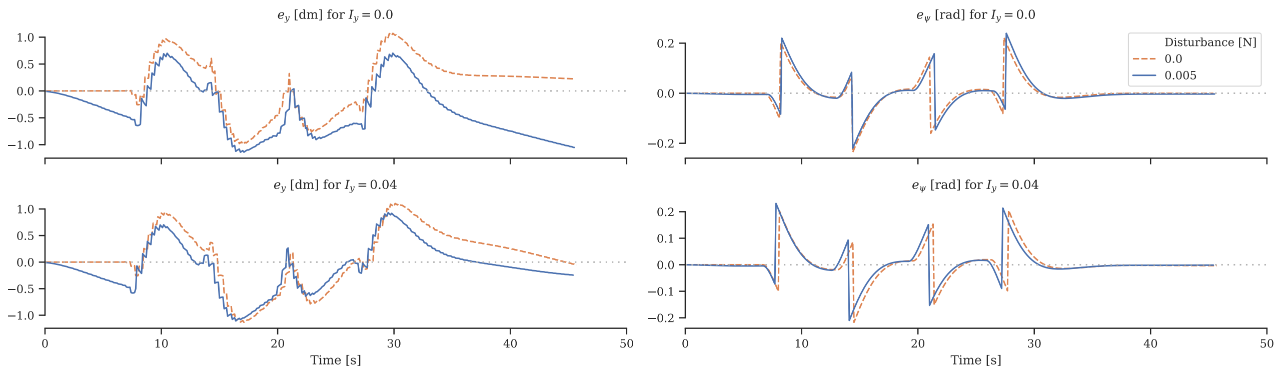


Figure 5.10: Positional errors for the controller with and without integral action.

The input signals τ are depicted in Figure 5.11. τ_ψ is hardly affected, which could be explained by the fact that the disturbance is here applied on the lateral direction. If the disturbance would also be applied in the rotational direction simular behavior as for the lateral direction is expected. The lateral input clearly compensates for the constant lateral force by continuously applying a counterforce, as observed by the vertical shift.

The disturbance rejection is here discussed for the lateral dimension, but the idea carries over to the rudder position, where it can be applied in the same way. For the rest of this thesis a PI controller is used with the gains found in this chapter as summarized in Table 5.4.

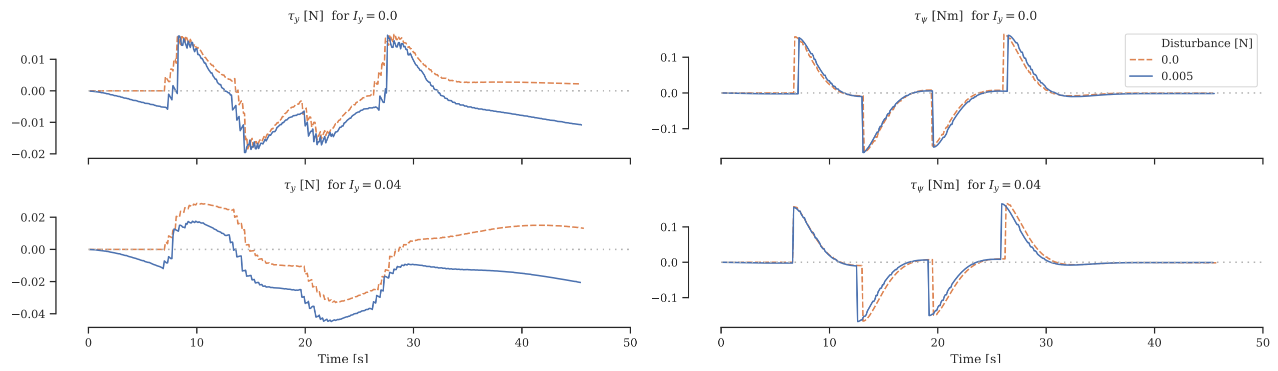


Figure 5.11: Control signals τ for different choices of integral gains and disturbance. Note that the y-axis has a different scale in each plot.

Table 5.4: Controller gains for PI controller

P_y	0.1
P_ψ	0.1
I_y	0.04
I_ψ	0.0

5.4 Results

In this section the above work is put together and applied to a path from the path planner instead of an artificial path like the block path in the previous sections. Figure 5.12 shows the sailed path which looks fairly equal to the path to follow. At the top right corner of the path, the sailed path intersects with the red outer layer of the map. However this is a buffer zone so no collision has taken place here.

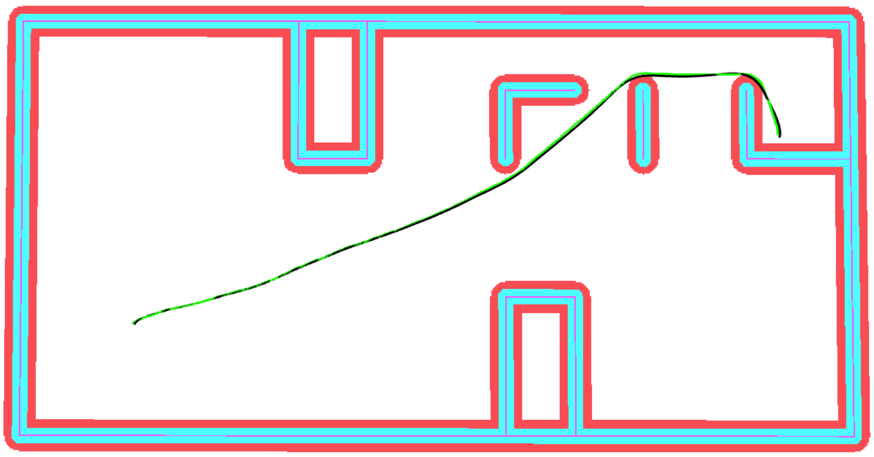


Figure 5.12: Reference path (green) and sailed path (black) when following a path from the path planner from Chapter 4.

The errors between the given path and the sailed trajectory are depicted in Figure 5.13. The shaky line is a result of coarse discretization of the path. The path follower finds the look ahead point as one of the points on the path to follow. Because these points are distributed with gaps in between, the error jumps slightly when a

Table 5.5: Root mean square error for following the path planner path with and without a Gaussian noise of $0.01m$ lateral and $0.01°$ rotational standard deviation

noise	$error_y$ [m]	$error_\psi$ [rad]
no	0.031	0.09
yes	0.032	0.10

new point is defined as the closest point. Similar logic applies to the inputs signals in Figure 5.14. One could see that the maximum lateral error is $0.05m$. The system is also run with noise on the position estimation to visualize the effect of noise for the control algorithm. The noise is Gaussian noise with a standard deviation of $0.01m$ lateral error and $0.01°$ rotational error. This is the same error as the position error applied for the mapping algorithm

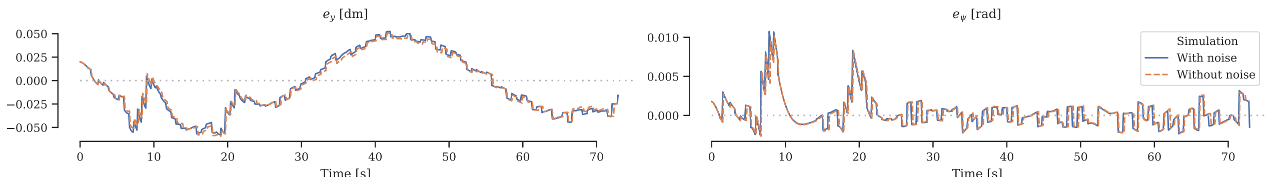


Figure 5.13: Positional errors for a static reference path.

The input signals τ are depicted in Figure 5.14. The two bumps in the beginning represent the two tighter corners in the beginning. The input signals are now smoother and smaller than in the previous discussion, because the path is more smooth as well.

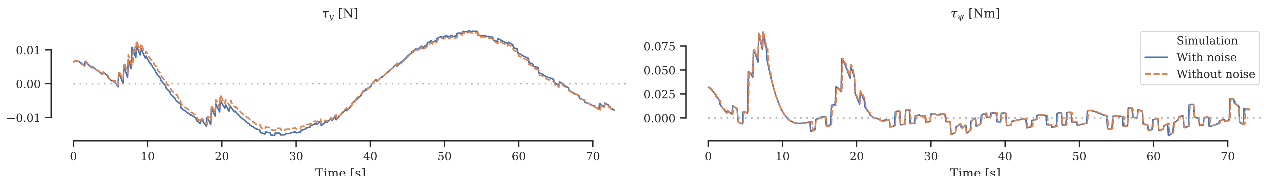


Figure 5.14: Control signals τ for a static reference path.

5.5 Discussion

Following the path from Figure 5.12 is performed within a maximum error of $0.10m$ with noise. The safety margin applied in the path planning equals the length of the USVwhich is $20cm$. The mapping is performed with an maximum error of $0.10m$ as well resulting in exactly $0.20m$ of error for the control and mapping in the extreme case. This chapter focused on PI control, thus without derivative action. A differential gain is often added for controlling the velocity error for compensating for oscilatory motions or overshoot. Neither oscilatory motion nor excessive overshoot harm the current performance and therefore differential control is not researched here. The same applies for adaptive control and Model Predictive Control (MPC). These techniques have shown to decrease error for severe disturbances, or when the system is subject to constraints with are easily and often violated.

5.6 Conclusion

This chapter has proposed an elegant way to define the error required for the controller. A look ahead approach is applied for path following, and for dynamic positioning at the goal the system uses the direct error between its position and the reference. A proportional controller with $0.25m$ look ahead distance extended with an integral gain properly controls the USV. Both for simulated block path and for a path generated from the path planner module, the performance of this controller is evaluated and approved. The next step is to link the mapping, path planning and path following modules together into a working system. This is shown in Chapter 6.

Chapter 6

Results & Discussion

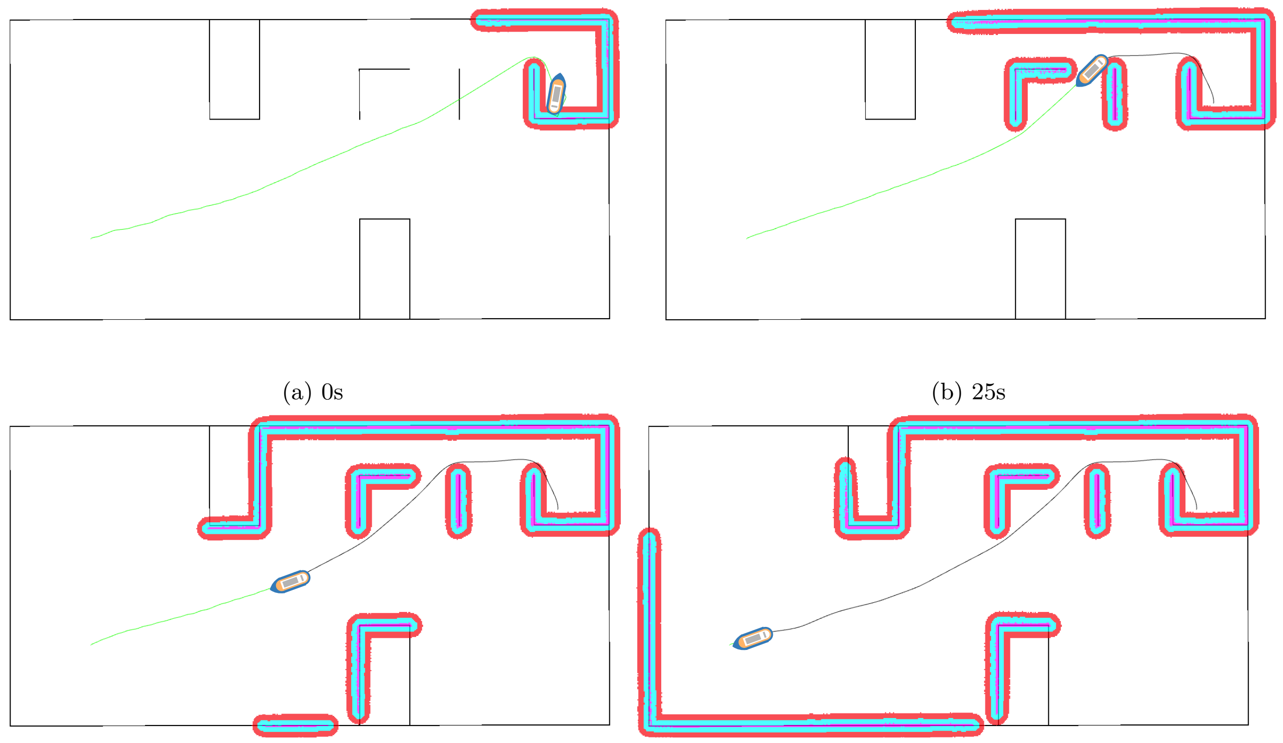
As final part of the technical work of this thesis, the modules as designed in Chapters 3 to 5 are connected to cooperate as a complete navigation system actuating the Unmanned Surface Vessel (USV) according to the schematic in Figure 2.1. The system requires lidar and Global Navigation Satellite System (GNSS)/Inerial Measurement Unit (IMU) sensor data and a goal reference for maneuvering the USV. All other parameters are variable and could be varied according to the vessel used. The result in Section 6.1 are acqured using a simulated map based on straight lines. The shapes of the environment in Sections 6.2 and 6.3 are equal to the real world. Of course this simulation cannot replace an outdoor test in the open water. Outdoor tests have been performed with the vessel provide by Xomnia but broken machinery, low end sensors and an increasing number of error sources finally resulted in a simulation based research. Simulation based research also made it easier to isolate separate modules for evaluation.

6.1 Simulated map

At first all modules are linked together to make the USV find the goal from its initial position without any knowledge of the environment. The results are shown in Figure 6.1. The GNSS/IMU sensor will return the location with known probability. The lidar sensor will send its acquired data to the mapping node. The mapping node will turn all the combined sensor data into a map, which is shown as the colored parts in the figures. The map expands over time as more grid cells have been explored. The mapping algorithm is run at $10Hz$. The path planning algorithm updates the path at a frequency of $10Hz$ as well. Every map iteration the path to follow is updated. New obstacles might have occured and the starting position (the current USV position) has differed as well. The path planner changes the plan over time but does not know anything about what is in the

Table 6.1: Root mean square error and maximum error for mapping and minimum distance between vessel and closest obstacles

map	RMS error mapping [m]	max mapping error [m]	closest obstacle [m]
simulated map	0.033	0.11	0.35
Amsterdam	0.029	0.08	0.41
Global planner	0.026	0.06	0.25


(a) 0s

(b) 25s

(c) 50s

(d) 75s

Figure 6.1: Recording at times 0(a), 25(b), 50(c) and 75(d) seconds of a simulation in a virtual environment applying the complete system from this thesis

undiscovered area. The undiscovered grid cells are assumed to be accesible until measurements state otherwise. The path follower is also running a $10Hz$ and performing well. The path follower only looks at the point at a distance d_{LA} from the USV on the path.

The mapping algorithm seems to perform well because no false positives or false negatives are detected outside of the uncertainty bounds. The performance of the mapping is similar to the performance in Chapter 3 because the situation is very similar. The same mapping algorithm is performed based on the same map. The only difference is the route the USV is traversing. For a good mapping algorithm the route sailed for discovering the area should not matter significantly when the accuracy of the location estimation remains equal over the whole map. The error in mapping is noted in Section 6.1. The planner might show unefficient behavior due to turning into dead ends because of the absence of a global planner. More about possible path planner challenges is shown in Section 6.3. In this case the path planner is performing as expected because the planned paths are quite smooth and no collision between the planned path and the actual map have occured. The USV is entering the red area (outer bounds of the map) which represent a buffer area. This area would rather be avoided but collision would only occure when the USV hits the purple area (inner bounds of the map). Quantifiable data can not be omitted for the path follower because the path is adapted to the position of the USV and therefore the whole system is performed in a feedback loop. This means that there is no path available as a reference for evaluation of the system. The performance of this simulation is summarized in Section 6.1. The minimum distance between the vessel and closest obstacle is $0.35m$. $0.20m$ is for compensation of the vessel size so another $0.15m$ is available to compensate for the mapping error. The mapping error is at maximum $0.11m$ and therefore the vessel will not even be colliding in the most extreme case with maximum error of all components.

6.2 Real map

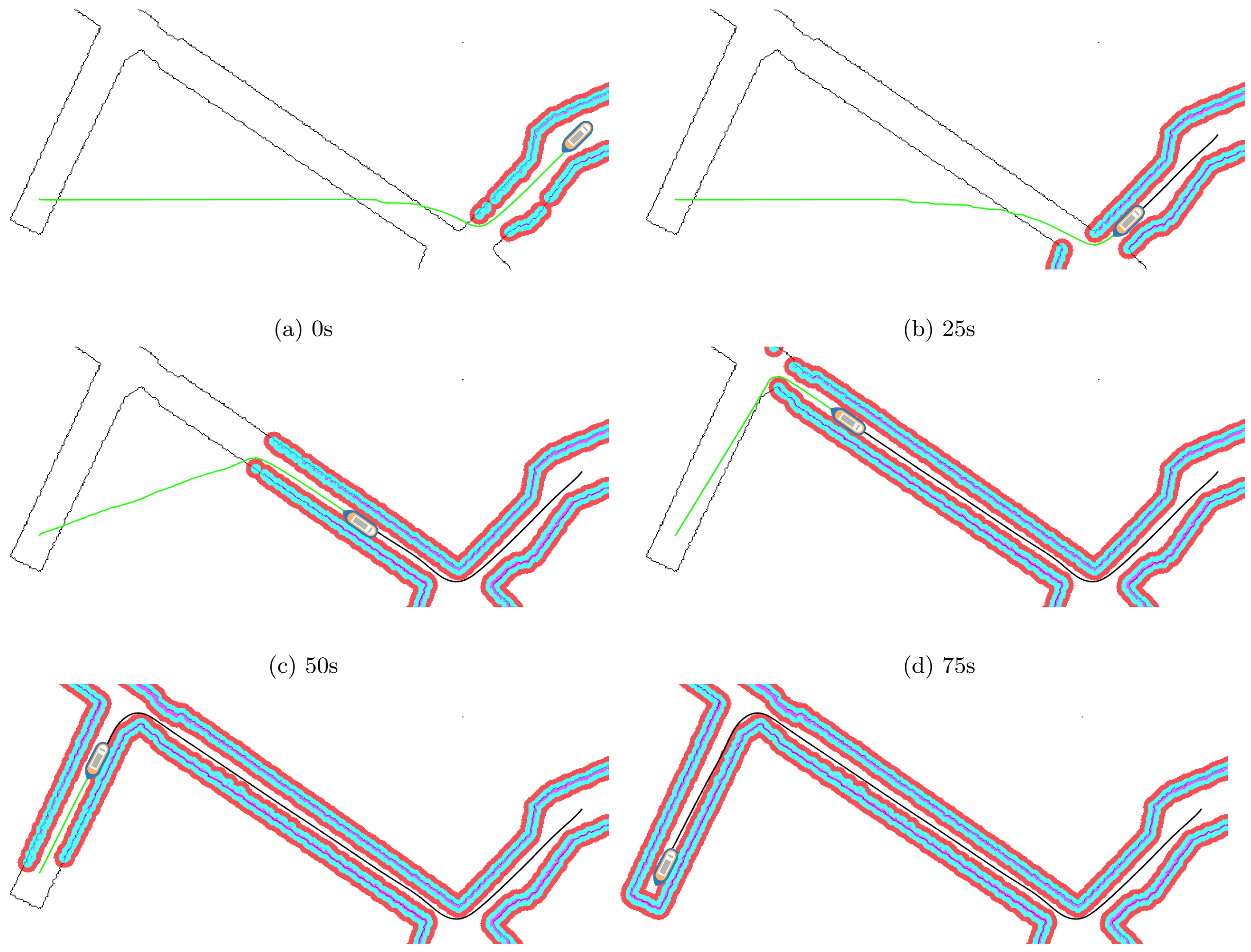
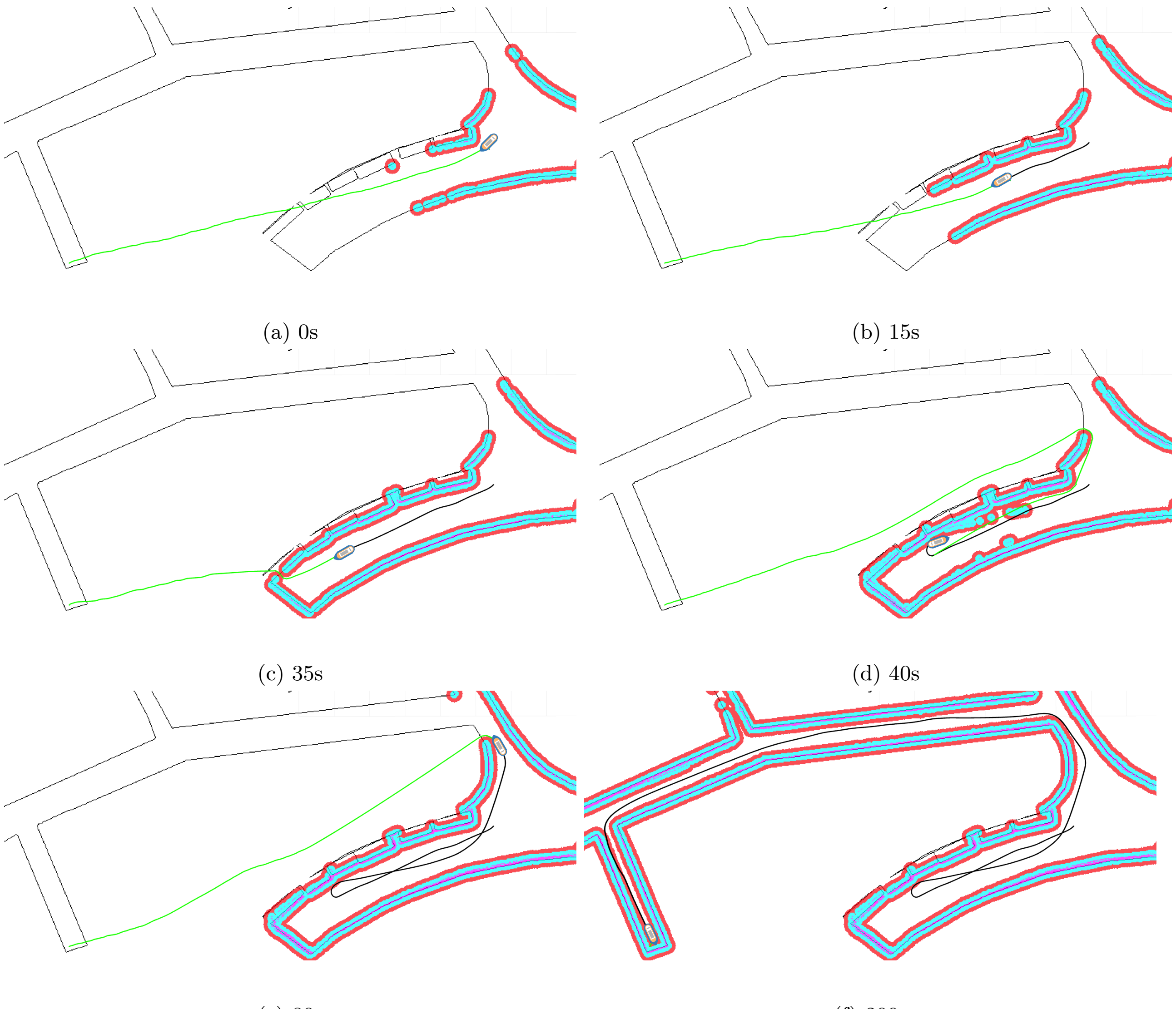

(a) 0s (b) 25s (c) 50s (d) 75s (e) 100s (f) 125s

Figure 6.2: Recording at times 0(a), 25(b), 50(c), 75(d), 100(e) and 125(f) seconds of a simulation in a environment based on the Amsterdam canals applying the complete system from this thesis

Again the same system as shown before in Section 6.1 is applied but this time in a map based on a real situation. A map based on a PNG image from Amsterdam representing the route from the Xomnia office (top right) to the docking space (bottom left) is used for navigation this time. Please note that the USV model is based on a scaled version of an USV which is around $30cm$ in lenght. Therefore a scaled map of Amsterdam with in this case a width of 15m is used. Figure 6.2 shows the sailed path on created map for the Amsterdam situation. Even more than with the previous example the green line is visible in unexplored area. At $0s$ the path to follow is mainly straight except for the beginning where the map is discovered and expanding. Over time the discovered map is expanding and the path to following is turning into its final shape. Again the USV is able to follow the path with only slightly touching the red safety zone. Section 6.1 shows the errors for mapping and the complete system. The minimum distance between the vessel and closest obstacle is $0.40m$. $0.20m$ is for compensation of the vessel size so another $0.20m$ is available to compensate for the mapping error. The mapping error is at maximum $0.08m$ and therefore another $0.12m$ margin is available.

The simulation in the Amsterdam map proves that the system is not only capable to maneuver autonomously in a simulated map like in Section 6.1 but also in a real world based map. As mentioned before, the real world based map cannot replace outdoor test, but it is a step closer towards outdoor testing.

6.3 Global planning


(a) 0s

(b) 15s

(c) 35s

(d) 40s

(e) 80s

(f) 200s

Figure 6.3: Recording at times 0(a), 15(b), 35(c), 40(d), 80(e) and 200(f) seconds of a simulation in a environment based on the Amsterdam canals applying the complete maneuvering system to show drawbacks of the path planner

This section points out one of the challenges of the system. As stated before the USV only requires Light

Detection And Ranging (LiDAR) and Global Positioning System (GPS)/IMU data to perform the autonomous maneuvering. No prior knowledge like a map is required. However, due to the lack of knowledge the path planning cannot foresee what is in the undicoverd area of the map. The path planner might plan a path which turns out to be unfeasible because an undiscovered obstacle is blocking the way. An example of such a sitation is shown in Figure 6.3. The USV is heading from the starting point in the bottom right to its goal position in the bottom left of the figure. At first the USV will find its way to the left because that is the shortest route to the goal when the obstacles are not yet discovered. Everything goes fine until the road block is detected in Figure 6.3d. The path is recalculated and heads the opposite direction now. The USV will have to make a sharp turn and the route is continued by following the detour. This will finally lead to the goal. This maneuver is made without collision which shows that the system is capable of solving such situations as well. This situation could be handled even better by applying a global planner based on maps of the area. A global route can be followed and an alternative route can be created in those cases where the internal map does not overlap with the map of the area. The minimum distance between the vessel and closest obstacle can be found in Section 6.1 and is $0.27m$. $0.20m$ is for compensation of the vessel size so another $0.07m$ is available to compensate for the mapping error. The mapping error is at maximum $0.06m$ and therefore the vessel will not even be colliding in the most extreme case with maximum error of all components. A note has to be made that the margin is really small however in this most extreme case.

Another interesting feature of the mapping algorithm is pointed out in Figure 6.3d. These are the points which are detected and drawn in the internal map, but which do not appear in the actual map. The false positives are visable in the figure by comparing Figures 6.3c and 6.3d. The added points are mostly false positives and are detected in the map due to a fast rotation of the USV. The mapping algorithm cannot handle this fast motion due to limited update rate of the USV position and therefore the lidar point are plotted with very low accuracy. Since these false positives are only detected once, they are marked in the map as low probability of occupancy. After the rotation is finished, the USV will move normally again and the new lidar measurment do not show the false positives in their data. The false positives will fade away from the map as their probability decreases and only the high probability obstacles remain in the map.

6.4 Conclusion

Concluding, one could state that the USV is able to perform completely autonomously maneuvering given a map and a goal. Three different map with goals have been evaluated and every simulation was able to safely return the USV to its goal. The simulated map is the easiest to maneuvering in. The USV stays the furtherst away from the obstacles and also the path planning algorithm is not suddenly suprised by obstacles hidden behind other obstacles due to the square shapes. The Amsterdam map is a bit harder for maneuvering but still performs wel. The last map shows a draw back of the system as designed in this thesis. Due to a lack of global information about planning routes, the path planner might not plan the best path straight away. However, based on the knowledge the system have, it does the best possible. Overall one could state that the system performs well.

Chapter 7

Conclusion

This research is performed in order to answer the following research questions as stated in the introduction:

> **What navigation and control methods are required to maneuver a boat autonomously in urban waterways?**

This section provides the answer to the research question by summarizing the conclusion from the separate chapters. This research presents a generic guideline for designing autonomous vessels for close proximity applications using open source software. The generic guideline is desired to make it easier to apply automation on vessels. Automated vessels have potential to decrease the number of maritime accidents significantly and might be the solution for an upcoming shortage of crew members on board.

The goal of designing an autonomous Unmanned Surface Vessel (USV) is to make an USV maneuver autonomously between its initial position and a reference point. The research done in this thesis has been performed in simulation using a dynamic model of the Delfia by TU Delft. In order to perform this research three separate modules are defined and later coupled: mapping, path planning and path following.

The mapping module creates a map based on the sensor input. Localization has been performed either with positioning sensors and Simultaneous Localization and Mapping (SLAM) algorithms. SLAM has the capability to outperform positioning sensors in accuracy of the map. However higher robustness and the presence of a global reference frame which does not suffer from drift favours the use of positional sensors. A combination of positioning sensors and SLAM algorithm might also be an option, but is not evaluated in this thesis for its extended complexity.

The middle module, path planning, creates a path to follow out of the map from the mapping algorithm. Several algorithms have been evaluated of which Dijkstra seems to be the best option. Dijkstra is guaranteed to find an optimal path and simultanously provides a cost map of the environment. The cost map is used for smoothing the path which makes is more comfortable to follow the path. The smoothed path is easier to follow and shorter than the orginial because it is not limited to the 8-connected grid with its only horizontal, vertical and 45° angle segments.

Path following is the final module. Path following is applied for calculating actuator inputs based on the error between the USV currents position and the path from the path planner. A proporional-integral controller is implemented which looks a certain distance ahead in order to account for the future path.

Finally the modules are connected in order to cooperate as one system. This system is tested in three different cases. At first in a simulated environment showing satisfactory results. Secondly a map of a commonly traversed route is uploaded and again the maneuvering is performed well. Finally a dead end is added to the map to reveal the problems the path planner has with planning a path without prior knowledge of the environment. However the system is able to adapt to the situation, plan a detour and continue its journey until the goal is reached.

7.1 Contributions

This research has proven that it is possible to make a vessel sail autonomously to a given goal using mapping with positioning sensors, Dijkstra path planning with gradient descent in a quadratically approximated costmap and propertional integral control in simulation. By the authors knowledge, This combination has never been performed before in autonomous sailing combined with the Delfia ship model. All simulation results are performed within the defined bounds for the performance indicators. Additionally, this research provides a generic guideline for autonomous maneuvering in maritime applications. The guideline is generic in the sense that it applies to any ship fullfills the sensor setup. The grid cell size and the safety margins for path planning have to defined. Tuning the controller is mostly automated by applying a grid search method while optimizing for minimal error. The use of propiary software is avoided in order to make this research repeatable without the need of acces to expensive software packages. Only open source software languages like C^{++}, *Python*, and *ROS* are used. Numerous mapping, path planning and control methods have been reviewed in this research. Several have been applied in the simulation and with that another validation step is taken in the validation of the previous researches.

7.2 Future work

- **Localization** - In this research a choice for a localization in a mapping algorithm has been made between the use of positioning sensor and SLAM. However a next step might be to combine them and implement a SLAM algorithm using positioning sensor data as well. Such an algorithm has already been succcesfully applied by [43] but was out of the scope of this research. An increase in map accuracy while maintaining robustness is expected due to the combination the advantages of both SLAM and positioning with sensors.
- **Global planner** - Extending the path planning module with a global planner with prior knowledge about the environment will make the system applicable in those situation where dead ends are present in the area. When information about the area is inserted local traffic rules and preferred routes could be implemented.
- **Model predictive control** - Control could be extended by applying Model Predictive Control (MPC) for the path follower. The performance of the path follower will likely increase due to anticipation based on

the physics of the USV. Several works for simulated applications have been published [42] on MPC, but the practical implementation seems more challenging.

- **Outdoor verification** - The system as provided is not verified in an outdoor environment. For the evaluation of the systems robustness outdoor testing will be required.
- **3D lidar or gimbal based lidar** - The 2D lidar is tested outdoors, but turned out to be of too low quality for maneuvering of a full size vessel outdoors mostly due to limited range, and planar view. A gimbal mounted 2D Light Detection And Ranging (LiDAR) or a 3D LiDAR could sharply increase the results and might be a major step towards a full size working system.

Bibliography

[1] B. S. Frey, D. A. Savage, and B. Torgler, “Behavior under extreme conditions: The titanic disaster,” *Journal of Economic Perspectives*, vol. 25, pp. 209–22, March 2011.

[2] S. A. Horn and C. P. Neal, “The atlantic empress sinking - a large spill without environmental disaster,” *International Oil Spill Conference Proceedings*, vol. 1981, no. 1, pp. 429–435, 1981.

[3] R. Negenborn, “Infographic on autonomous shipping.” http://www.negenborn.net/pubs/all/how-will-autonomous-ships-work.pdf, accessed: 2019-04-29.

[4] E. Demirel and D. Bayer, “a study on cost optimization in the ship management,” *PROCEEDINGS BOOK*, p. 67, 2016.

[5] M. Dikmen and C. M. Burns, “Autonomous driving in the real world: Experiences with tesla autopilot and summon,” in *Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pp. 225–228, ACM, 2016.

[6] T. Lauvdal and T. I. Fossen, “Robust adaptive ship autopilot with wave filter and integral action,” *International Journal of Adaptive Control and Signal Processing*, vol. 12, no. 8, pp. 605–622, 1998.

[7] O. Levander, “Autonomous ships on the high seas,” *IEEE Spectrum*, vol. 54, no. 2, pp. 26–31, 2017.

[8] VolvoPenta, “Autodocking.” https://www.volvopenta.com/marineleisure/en-en/news/2018/jun/volvo-penta-unveils-pioneering-self-docking-yacht-technology.html, accessed: 2018-11-21.

[9] SINTEF, “Hull2hull.” https://www.sintef.no/projectweb/hull-to-hull/, accessed: 2018-11-21.

[10] Yara, “Yara birkeland.” https://www.yara.com/knowledge-grows/game-changer-for-the-environment/, accessed: 2018-11-21.

[11] CaptainAI, “Captainai.” www.captainai.com, accessed: 2018-11-21.

[12] Xomnia, “Self driving boat.” https:/self-driving-boat.com/, accessed: 2018-11-21.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” *CoRR*, vol. abs/1604.07316, 2016.

[14] AMS/MIT, “Roboat.” www.roboat.org, accessed: 2018-11-16.

[15] L. Chen, A. Haseltalab, V. Garofano, and R. R. Negenborn, “Eco-vtf: Fuel-efficient vessel train formations for all-electric autonomous ships,” in *2019 18th European Control Conference (ECC)*, pp. 2543–2550, 2019.

[16] A. Haseltalab and R. R. Negenborn, “Adaptive control for autonomous ships with uncertain model and unknown propeller dynamics,” *Control Engineering Practice*, vol. 91, p. 104116, 2019.

[17] S. Thrun, “Particle filters in robotics,” in *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[18] J. Mullane, B. Vo, M. D. Adams, and B. Vo, “A random-finite-set approach to bayesian slam,” *IEEE Transactions on Robotics*, vol. 27, pp. 268–282, April 2011.

[19] B. Tovar, L. Guilamo, and S. M. Lavalle, “Gap navigation trees: Minimal representation for visibility-based tasks,” in *In Proc. Workshop on the Algorithmic Foundations of Robotics*, pp. 11–26, 2004.

[20] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 116–121, March 1985.

[21] M. Tomono, “Robust 3d slam with a stereo camera based on an edge-point icp algorithm,” in *2009 IEEE International Conference on Robotics and Automation*, pp. 4306–4311, May 2009.

[22] Z. Lu, Z. Hu, and K. Uchimura, “Slam estimation in dynamic outdoor environments: A review,” in *Intelligent Robotics and Applications* (M. Xie, Y. Xiong, C. Xiong, H. Liu, and Z. Hu, eds.), (Berlin, Heidelberg), pp. 255–267, Springer Berlin Heidelberg, 2009.

[23] S. Thrun, “Learning occupancy grid maps with forward sensor models,” *Autonomous Robots*, vol. 15, pp. 111–127, Sep 2003.

[24] P. A. Shirley, “An introduction to ultrasonic sensing,” *Sensors*, vol. 6, no. 11, pp. 10–17, 1989.

[25] T. Hentsberger and H. Aghazarian, “Stereo vision-based navigation for autonomous surface vessels,” *Journal of Field Robotics*, 2011.

[26] M. Schuster, M. Blaich, and J. Reuter, “Collision avoidance for vessels using a low-cost radar sensor,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9673 – 9678, 2014. 19th IFAC World Congress.

[27] L. Huang and M. Barth, “Tightly-coupled lidar and computer vision integration for vehicle detection,” pp. 604–609, June 2009.

[28] T. B. O. H. M. C. T. Kriechbaumer, K. Blackburn, “Quantitative evaluation of stereo visual odometry for autonomous vessel localisation in inland waterway sensing applications,” 2015.

[29] D. Hahnel, “mapping with mobile robots,” 2004.

[30] U. Frese, R. Wagner, and T. Röfer, “A slam overview from a user’s perspective,” *KI - Künstliche Intelligenz*, vol. 24, pp. 191–198, Sep 2010.

[31] A. Subramanian, X. Gong, C. L. Wyatt, and D. Stilwell, “Shoreline detection in images for autonomous boat navigation,” *Institute of Electrical and Electronics Engineers (IEEE)*, 2007.

[32] R. Jamiruddin, A. O. Sari, J. Shabbier, and A. Tarique, “Rgb-depth slam review,” 2018. review paper.

[33] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[34] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pp. 38–47, July 2007.

[35] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., 1998.

[36] B. Randall Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning.," *J. ACM*, vol. 40, pp. 1048–1066, 11 1993.

[37] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, 11 2016.

[38] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug 1996.

[39] N. Zhou, 2014.

[40] H. B. CURRY, "The method of steepest descent for non-linear minimization problems," *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261, 1944.

[41] R. Philippsen, "A light formulation of the e interpolated path replanner," tech. rep., ETH Zurich, 2006.

[42] A. Haseltalab and R. R. Negenborn, "Model predictive maneuvering control and energy management for all-electric autonomous ships," *Applied Energy*, vol. 251, p. 113308, 2019.

[43] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, November 2011.