Per Gunnar Berg Torvund

Nonlinear Autonomous Docking and Path-Following Control Systems for the Otter USV

June 2020

NTNU
Norwegian University of
Science and Technology

Nonlinear Autonomous Docking and Path-Following Control Systems for the Otter USV

Per Gunnar Berg Torvund

Cybernetics and Robotics

Submission date:	June 2020
Supervisor:	Thor I. Fossen
Co-supervisor:	Johann A. Dirdal

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Nonlinear Autonomous Docking and Path-Following Control Systems for the Otter USV

Per Gunnar Berg Torvund

Master of Science in Cybernetics and Robotics
Submission date: June 2020

Supervisor:	Thor I. Fossen,	Department of Engineering Cybernetics
Co-supervisor:	Johann A. Dirdal,	Department of Engineering Cybernetics

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem description

The main purpose of the project is to develop nonlinear control algorithms for docking and path following for small USVs. This project is going to investigate these topics for a small research vessel with an underactuated actuator configuration. The algorithms should be simulated on a model of the Otter USV.

The following topics and challenges should be considered in more detail:

1. Literature study on methods for docking, path following, obstacle avoidance and how to simulate these topics in a realistic manner. Appropriate research questions and requirement specifications should be formulated in order to solve the problem. Differences and shortcomings between simulation studies and real experiments should be identified and explained.
2. Develop a simulator in Matlab for testing of different control algorithms. The simulator should include the USV dynamics, actuator dynamics, sensory systems with realistic measurement noise and stochastic time-varying ocean currents.
3. Develop control algorithms for autonomous docking (including control allocation), LOS path following and path re-planning. The path following should utilize a predefined path that is recalculated based on proximity to obstacles. Derive and include stability properties if appropriate.
4. Develop algorithms/methods for generating realistic sensor noise. This includes stochastic time-varying ocean currents and sensor noise.
5. Conduct a large Monte-Carlo simulation study where the methods and algorithms are evaluated in detail. The Monte-Carlo simulations should be based on varying ocean current and obstacle locations. The results should be discussed and related to a real-life scenario.
6. Conclude the findings in a report.

Abstract

This thesis aimed to use nonlinear control algorithms for autonomous docking, path following and obstacle avoidance for Maritime Robotics smallest Unmanned Surface Vessel (USV), the Otter USV. The field of autonomous docking is currently a popular topic where multiple solutions have been presented for fully actuated vessels. However, the Otter USV is only controlled by two fixed rear thrusters, making the vessel underactuated. The fact that the thrusters are fixed means that they are unable to directly affect the sideways motion of the vessel, which complicates the control problem. In order to address this problem, a simulator was developed in Matlab where the nonlinear dynamics of the Otter USV were based on data from previous experiments. The simulator was developed in order to test various control schemes on the Otter USV.

In this thesis, a PID controller and two higher order Sliding Mode Controllers (SMCs) were implemented as course controllers and a PI controller as a surge controller. The SMCs that were implemented was a PID-SMC and a Super-Twisting Controller (STC). These controllers were chosen since they are robust to parameter uncertainties. The PID controller was implemented to compare the SMCs with a "Simple" controller. These controllers were used together with a Line of Sight (LOS) guidance law and a obstacle avoidance algorithm in order to calculate the desired course for path following while avoiding obstacles. The vessel entered a docking phase when it was within a given distance from the dock. This phase consisted of reducing the surge speed from 1 to 0.2 m/s.

In order to compare the controllers to each other, a Monte-Carlo simulation of 10000 iterations was conducted. The current velocity, crab angle and obstacles was stochastically generated for each iteration. The results also consists of two constructed cases: one with weak ocean currents and one with strong ocean currents. These cases were included in order to show how the USV handled both "mild" and "extreme" conditions. The Monte-Carlo simulation showed that the STC tracked the course and had the lowest cross-track error out of all controllers, while the PID controller had the worst performance. The total median course and cross-track error for the PID-SMC was 114% and 66% larger than for the STC, respectively. Furthermore, the energy consumption of the STC was slightly lower than the other controllers. However, since there was very little difference in energy consumption between the controllers, this was seen as trivial.

It's suggested that the surge controller can be improved by using more a advance controller such as the STC. For even more realistic simulations, it's recommended that other external disturbances such as wind and waves are included. Lastly, it's recommended to test the controllers by implementing them on the actual Otter USV.

Sammendrag

Denne oppgaven gikk ut på å bruke ulineære kontrollalgoritmer for autonom dokking, banefølging og hinderunngåelse for Maritime Robotics sin minste overflatefarkost, "oteren". Autonom dokking er for tiden et hett tema der flere løsninger har blitt presentert for fullaktuerte skip. "oteren" blir bare kontrollert av to faste propeller/thrustere plassert akter, noe som gjør fartøyet underaktuert. Dette gjør kontrollproblemet mer utfordrende siden fartøyet ikke kan direkte beveges sidelengs (ved bruk av thrusterene). For å adressere dette ble det utviklet en simulator i Matlab der forskjellige kontrollordninger kunne testes der den ulineære dynamikken var modellert basert på data funnet i tidligere eksperimenter.

Oppgaven fokuserte på å implementere to høyere ordens Sliding Mode kontrollere (SMC) som kurskontrollere og å bruke en PI-kontroller som en fartskontroller. SMC-ene som ble implementert var en PID-SMC og en Super-Twisting kontroller (STC). Disse kontrollerne ble valgt ettersom de er robuste for parameterusikkerheter. I tillegg til disse ble en enkel PID-kurskontroller implementert og sammenlignet med SMC-ene. Disse kontrollerne ble brukt sammen med en Line of Sight (LOS) veiledningslov og en algoritme for hinderunngåelse til å beregne ønsket kurs for banefølging uten å kollidere med hinder. Fartøyet gikk inn i en dokkingsfase da det var innenfor en gitt avstand fra kaien der hastigheten til fartøyed ble redusert fra 1 m/s til 0.2 m/s.

For å sammenligne kontrollene med hverandre ble det utført en Monte-Carlo-simulering med 10000 iterasjoner. Strømhastigheten, strømvinkelen og hindringene ble generert stokastisk for hver iterasjon. Resultatet inneholder også to konstruerte tilfeller: en med svake havstrømmer og en med sterke havstrømmer. Disse tilfellene ble inkludert for å vise hvordan USVen håndterte både "milde" og "ekstreme" forhold. Monte-Carlo-simuleringen viste at STCen hadde lavest kurs- og kryssporingsfeil, og at PID-kontrolleren hadde dårligst ytelse. Den totale medianen av kryssporings- og kursfeilen for PID-SMCen var henholdsvis 114% og 66% større enn for STCen. Videre var energiforbruket til STCen litt lavere enn de andre kontrollerne, men siden det var veldig lite forskjell i energiforbrukmellom kontrollerne ble dette sett på som trivielt.

Det foreslås at hastighetskontrolleren kan forbedres ved å bruke en mer avansert kontroller som for eksempel STC. For enda mer realistiske simuleringer, anbefales det at andre ytre forstyrrelser som vind og bølger inkluderes. Til slutt anbefales det å teste kontrollerne på den faktiske "oteren".

Preface

This thesis is carried out at the Department of Engineering Cybernetics, at NTNU in Trondheim the spring of 2020. It is submitted as a requirement for the master's thesis TTK4900. Parts of chapters 1 to 4 are based on specialization project TTK4551, which was submitted the fall of 2019. Parts of these chapters were developed in joint work with Henrik B. Strand.

I would like to thank my supervisor, Thor Inge Fossen and my co-supervisor, Johann Alexander Dirdal for guidance and feedback. I would also like to thank my partner and family for their continuous support.

In the winter of 2020, my father, Kjell Tore, sadly passed away from cancer. As a supportive father and electrical engineer, he was always genuinely interested in my studies. I would like to dedicate this thesis to him as he inspired me to choose this field of study.

Per Gunnar Berg Torvund
Trondheim, June 2020

Contents

List of Figures

List of Tables

Acronyms

CF Center of force. 17

CG Center of gravity. 14, 15

CO Center of origin. 15, 17

DOF Degree of freedom. 13, 14, 16

FSMC First-order Sliding Mode Controller. 27, 31–33

PID-SMC Sliding Mode Controller based on a PID controller. 31, 42, 50, 51, 54, 55, 58, 60, 63

SMC Sliding Mode Controller. 6, 9–11, 27, 28, 31, 33

STC Super-Twisting Controller. 10, 11, 33, 34, 43, 50, 51, 54, 55, 57, 58, 60, 63

USV Unmanned Surface Vehicle. 1–5, 8–10, 22, 23, 25, 37, 39, 44, 45, 47, 63

Symbols

V_c	Current velocity	[m/s]
β_c	Crab angle of current	[rad]
β	Crab angle of craft	[rad]
χ	Course	[rad]
ω_b	Bandwidth frequency	[rad/s]
ω_n	Natural frequency	[rad/s]
ϕ	Roll angle	[rad]
ψ	Yaw angle	[rad]
τ	Control force	[N]
θ	Pitch angle	[rad]
e	Cross-track error	[m]
m	Mass of Otter	[kg]
n_i	Propeller shaft speed (input)	[rad/s]
p	Roll velocity	[rad/s]
q	Pitch velocity	[rad/s]
r	Yaw velocity	[rad/s]
s	Sliding surface	[-]
u	Surge velocity	[m/s]
v	Sway velocity	[m/s]
w	Heave velocity	[m/s]
x	Position in x direction	[m]
y	Position in y direction	[m]
z	Position in z direction	[m]

Chapter 1

Introduction

1.1 Motivation

Unmanned Surface Vehicles (USV) are vehicles that operates on the surface of the water without a crew. These vehicles have the advantages of being fast, small, inexpensive and have the ability of autonomous navigation. These advantages have made USVs popular for several applications such as ocean surveillance, search, rescue and military operations [1, 2]. The performance of the system is, however, affected by external disturbances and uncertainties in system dynamics. Because of this, it's important that the implemented controllers are robust to the aforementioned factors. This is the main motivation for using nonlinear controllers in this thesis, as these controllers do not remove system dynamics and many of them are robust to parametric uncertainties.

When developing control systems for the Otter USV, one interesting problem emerges. This USV is designed to only operate in the horizontal plane, which gives it three generalized coordinates: surge, sway and heading. By design, the Otter USV is only powered by two fixed thrusters in the aft, giving it only two control inputs. This results in the USV being *underactuated*, meaning that the surge and course controllers are coupled by an actuator model.

One challenging part of the thesis is the implementation of a docking maneuver for the USV. There has not been done a great deal of research on autonomous docking of USVs when compared to path following [3]. Moreover, the research mostly concerns fully actuated USVs, unlike the Otter USV.

1.2 The Otter USV

Maritime robotics is a company founded in 2005 that focuses on delivering vehicles, tools and systems that operates unmanned both in the air and on the surface. One of their products is the Otter USV, which is the smallest USV that Maritime Robotics produces. It can be used for several applications including seabed mapping and monitoring of sheltered waters [4]. It consists of a frame mounted on 2 pontoons, with the control box, batteries and sensors mounted on top of the frame (see Figure 1.1). It has a fixed electrical motor (thruster) integrated to each of the pontoons, meaning that difference in thrust between the two motors are necessary in order to turn the vessel. The length of the Otter USV is 2 meters and its width is 1.08 meters [4].

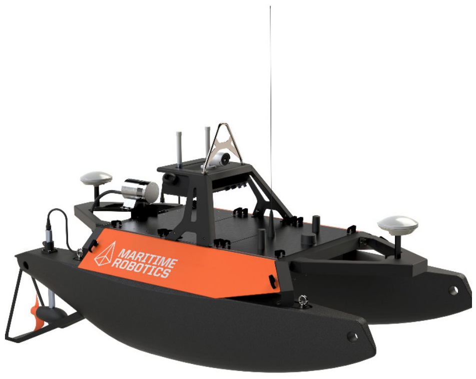


Figure 1.1: Image of the Otter USV, illustration from Geo-matching [5]

1.3 System overview

The control system of the Otter USV is divided into two parts: the path-following control system and the docking control system, as illustrated in Figure 1.2. The transition from path following to docking happens when the USV is within range of the on-board camera system. The input to the path following controller is given by the on-board Real Time Kinematic Global Navigation Satellite System (RTK GNSS), which outputs the position and course of the vessel. The RTK GNSS is part of a high performance inertial sensor called *Ellipse2-D* which is able to deliver centimeter accuracy in position with a 200 Hz output rate [6]. The path following controller uses a waypoint generator as input reference. These waypoints are predefined by the user based on the location of the dock, obstacles in the path and wanted behavior of the USV towards the dock.

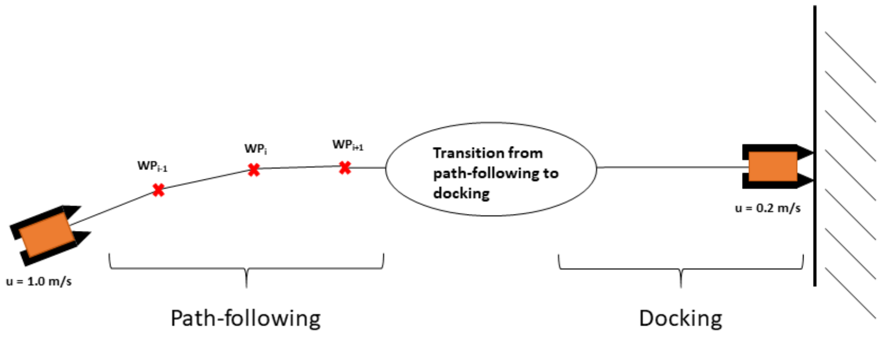


Figure 1.2: The Otter USV will use the path following algorithm until it's within range of the camera system.

When the docking algorithm is activated, the USV uses the stereo-camera vision on board the vessel. This control system calculates the position of the USV based on the position of multiple markers placed on the dock. Each marker contains information about its own position which the camera system extracts through a convolutional neural network. In addition, the camera module calculates the relative distance between the markers and the vessel. This relative position to the dock is used to decide the desired surge velocity. Figure 1.3 shows the system overview of the two control systems.

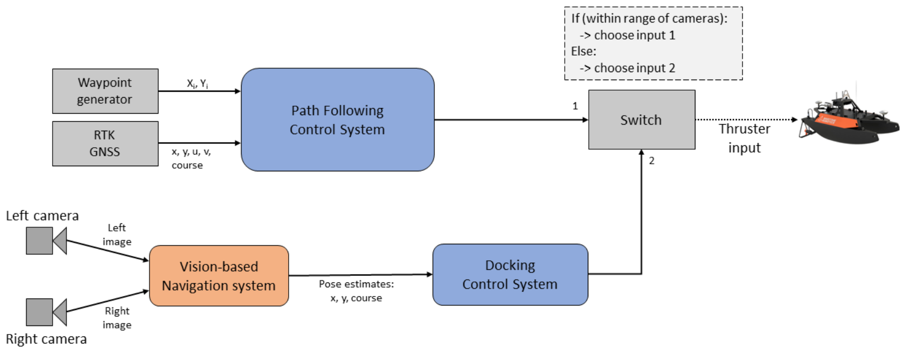


Figure 1.3: System diagram showing the two different control systems on board of the Otter USV. The path-following control system uses GNSS to determine the position of the USV, while the docking algorithm utilizes the on-board cameras mounted in the front.

1.4 Research questions

The following research questions will be answered in this thesis

Q1 What is the lowest speed at which the USV still is controllable with the presence of slow stochastic time-varying currents?

Q2 Is it possible for an underactuated USV to follow a path and dock with the presence of slow stochastic time-varying currents?

Q3 Does a controller that is robust to parameter uncertainty exist for an underactuated USV?

Q4 Is it possible for an underactuated USV to avoid obstacles in real-time while following a path?

1.5 Objectives

The objectives of this thesis are summarized as follows

- Description of the Otter USV mathematical model
- Implementation of surge and nonlinear course controllers
- Implementation of a path following guidance law
- Implementation of real-time obstacle avoidance algorithm
- Monte-Carlo simulation study for testing of different course controllers with slow stochastic time-varying ocean currents and obstacle locations
- Comparison of the performances for the different course controllers
- Suggestions for future work based on the results and research

1.6 Assumptions

The following assumptions was made in the thesis

- Surge speed $u_{\mathrm{d}} \in [0.2, 1]$ m/s, with desired cruising speed of 1 m/s while following a path
- Current speed $\mathrm{V_c} \in [0, 0.5]$ m/s

- Crab angle $\beta_c \in [-180°, 180°]$
- V_c and β_c vary slowly and can be considered constant
- Obstacle radius $R_o \in [2, 3]$ m
- Distance from obstacles are found using radar technology
- Obstacles are stationary
- Max 1 obstacle between two waypoints
- Same sampling rate for all sensors
- Course is measured
- No wind or waves present
- Measurement noise is filtered by low-pass filter or Kalman filter (i.e. not raw noise, but filtered)

1.7 Requirement specifications

The following requirement specifications was set for the system

- Controller bandwidth lower than system natural frequency $\omega_b < \omega_n$
- Median cross-track error $e < 0.5$ m
- Median course error $\tilde{\chi} < 6.2°$
- Convergence of states in the presence of ocean currents with amplitude 0.5 m/s
- The USV must always be outside the obstacle radius
- The energy consumption must not vary with more than 1.5% between the controllers

1.8 Contributions

The following constitutes the main contributions of the thesis

- Robust nonlinear course controllers designed for the Otter USV for path following and docking with a PI controller for surge

- Performance analysis of different course controllers, including comparisons between SMCs and a PID course controller
- Path re-planning algorithm for avoiding generated obstacles of varying location and size
- Simulator in Matlab for testing the control algorithms, including the USV dynamics, sensory systems with realistic measurement noise and ocean currents

1.9 Outline

The report is organized as follows

Chapter 1 Introduction including system overview, research questions, objectives, assumptions, requirement specifications and contributions

Chapter 2 Brief introduction to autonomy, different nonlinear controllers, path following, docking and obstacle avoidance with references to related work

Chapter 3 Mathematical model for the vessel, theory for LOS guidance laws, obstacle avoidance and the controllers used in the project

Chapter 4 Description of how the simulator was implemented in Matlab. This includes the measurement noise, external disturbance, controllers and discretization

Chapter 5 Presentation and discussion of the results from the simulations for two constructed cases and a large Monte-Carlo simulation of 10000 iterations

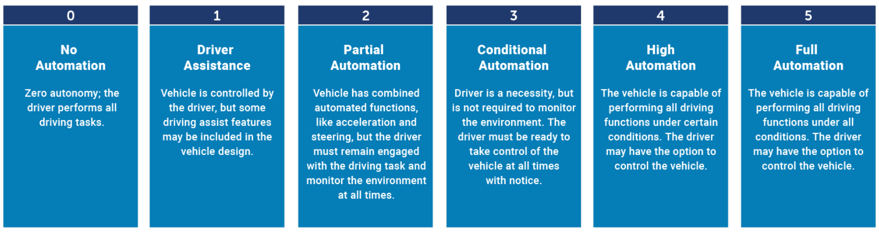
Chapter 6 Summarizing the findings in the report with suggestions for future work

Chapter 2

Literature study

2.1 Autonomous Systems

The Society of Automotive Engineers (SAE) have defined 6 levels of driving automation for on-road motor vehicles, ranging from no driving automation to full driving automation [7]. Even though these levels refer to on-road vehicles, it can also be used for water- and aircrafts given the general description of the different levels. Figure 2.1 gives a brief description to each of the levels as described by the SAE [7].



0	1	2	3	4	5
No Automation	**Driver Assistance**	**Partial Automation**	**Conditional Automation**	**High Automation**	**Full Automation**
Zero autonomy; the driver performs all driving tasks.	Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle design.	Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.	Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.	The vehicle is capable of performing all driving functions under certain conditions. The driver may have the option to control the vehicle.	The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

Figure 2.1: Description of all 6 levels of autonomy, obtained from Automotive Electronics [8]

Figure 2.1 states that autonomy level 2 means that the vehicle has combined automated function such as acceleration and steering. Given that the focus of the thesis is to design controllers for the same functions as mentioned in the figure, it can be stated that the automation referred to in this thesis is autonomy of level 2.

2.2 Autonomous applications for USVs

This thesis will focus on two main autonomous applications, namely: autonomous docking and path following. It should be noted that there exist several other autonomous applications such as situational awareness and risk assessment, though these will not be considered in the report.

Autonomous docking

The field of autonomous docking has been presented with various solutions for USVs under different conditions. An article by Breivik and Loberg presents a solution for docking a small USV with a larger mother ship traversing the sea [9]. The USV utilized constant-bearing guidance to track a virtual-target point on the mother ship. When the USV has matched both position and velocity of the mother ship, the docking is achieved by aligning the USV with the desired docking point. The work by Woo and Kim resulted in a solution for docking underactuated USVs by using a vector-field guidance method to avoid dangerous areas around the desired docking position [10]. A solution using numerical optimal control was presented by Martinsen, Lekkas, and Gros, for fully actuated marine vessels [3].

Path following

In order to safely guide a marine vessel along a desired path, a stable controller is needed to minimize the distance between the path and the vessel. Fossen, Breivik, and Skjetne presents a nonlinear guidance system for an underactuated marine vessel, where the cross-track error is minimized using a backstepping control law for surge and yaw [11]. The same problem of minimizing the cross-track error was solved by using a Lyaponov-based control law where the vessel was set to track a virtual-target vessel defined by a Serret-Frenet frame which moves along the path [12].

2.3 Obstacle avoidance

For vehicles to navigate unknown environments in real-time, it's crucial that the vehicle is able to avoid obstacles located along the path. There have been presented several methods for avoiding obstacles for marine vehicles, such as

- Potential fields [13]
- Dynamic window [14]
- A* path re-planning [15, 16]
- Set-based guidance [17]

The potential fields method have been shown to consist of several drawbacks and limitations, such as oscillating behaviour [18]. The dynamic window approach assumes that there is only forward velocities. When USVs move through water they tend to glide on the surface, which contradicts the assumption of only forward velocity. Furthermore, the dynamic window approach can be computationally expensive [17]. The paragraphs below will give a brief introduction of the A* path re-planning and set-based guidance methods.

A* path re-planning

The A* algorithm, published in 1968 by Hart, Nilsson, and Raphael [19], is one of the most effective direct search method for static networks shortest path problem [20]. It operates similar to the Dijkstra algorithm, published in 1959 [21], with the main difference being that the A* algorithm guides its search towards the most promising path by using heuristic cost estimations [22]. The algorithm will ensure that the obstacles are avoided by defining a high cost-value for the obstacle locations, as it will choose the path of lowest cost. A* has been shown to be a viable option for path re-planning for USVs as long as the obstacles are static [15, 16].

Set-based guidance

Recent results in set-based guidance theory has lead to a switching guidance system for underactuated USVs [17]. This method ensures path following and guarantees collision avoidance for both static and moving obstacles by switching between two different modes: path following and collision avoidance [23]. The goal for the collision avoidance mode is to track a given safe radius around the obstacle centre. Collisions will never occur as long as this radius is maintained [17]. Furhtermore, the set-based guidance presented by Moe and Pettersen has been shown to assure collision avoidance while abiding by the International Regulations for Preventing Collisions at Sea (COLREGs) [17].

2.4 Nonlinear controllers

Most real processes are inherently nonlinear in nature. This is due to the fact that the relationships in physics are nonlinear [24]. Nonlinear control methods are more complex than the control methods used in linear models, but they include more of the system dynamics than the linear control methods [25]. A common way to solve nonlinear problems is by linearizing it such that linear control methods can be used, simplifying the problem. However, the problem with linearization is that the model properties are “destroyed” and that the design process can be more complicated with a limited physical insight [26]. As research question Q2 states, this thesis will focus on implementing a controller that is robust to uncertainties, for which the following control methods are relevant.

- Sliding Mode Control (SMC)

- Super-Twisting Mode Control (STC)
- Model Predictive Control (MPC)

The paragraphs below will give a brief introduction of the different methods.

Sliding Mode Control

The first English publications of Sliding Mode Control (SMC) was in 1977 by Vadim Utkin [27]. SMC is considered to be one of the most promising robust control techniques [28]. SMCs have been used on multiple vehicles such as USVs, Autonomous Underwater Vehicles (AUV) and Remotely Operated Vehicles (ROV) [29, 30, 31, 32, 33, 34]. The concept of SMC is to define asymptotically stable sliding surfaces which all system trajectories converge to in finite time and then slide along until reaching the origin [27, 33]. One important assumption for SMC is that the uncertainties are bounded and that these bounds are known [29]. The main disadvantage of SMC is the chattering effect (see Section 3.7.1).

Super-Twisting Control

One of the problems that causes the chattering effect is that the First order SMC only ensures that the sliding variable (s) tends to zero. Several second order SMCs have been designed in order to ensure that $\dot{s}$ also tends to zero such that chattering is reduced [35]. Most second order SMCs are dependent on measurements of $\dot{s}$ or its sign, whereas the Super-Twisting controller (STC) can be implemented as long as the control appears in the first derivative of the sliding variable (relative degree of 1) [36]. One of the first publications about STC was by Arie Levant in 1993 [37], it has since been considered as one of the most popular second order SMCs [35].

Model Predictive Control

The first published formulation of Model Predictive Control (MPC) was in 1963 by Propoi [38]. A MPC functions in the following way: It starts by finding the current control action by solving a finite horizon open-loop optimal control problem, using the current and initial state of the plant [39]. This results in an optimal control sequence that is predicted to drive the output to the reference. The first control in this sequence is applied to the plant, and then the cycle repeats for each sampling instant [40]. MPC are widely used in industrial plants [41], but has also been used for path following as heading controllers [42, 43].

2.5 Summary

The performance of MPCs depend on the quality of the system model, they also usually have a high computational cost [44]. For these reasons, MPC was not considered as a viable option in the thesis. The thesis will therefore focus on designing different SMCs (including the STC) as course controllers, given that it's a robust and effective control approach for underactuated nonlinear systems [45]. The different SMCs will then be compared to each other and a standard PID controller.

The set-based guidance method has been shown to be a good choice when it comes to collision avoidance for static and moving obstacles [23]. This can make the method unnecessary complicated since this thesis only consider static obstacles. However, the switching mechanism can be used together with the A* path re-planning method, which is less computationally expensive. The obstacle avoidance method in this thesis will therefore be a hybrid of both methods.

Chapter 3

Theory

3.1 Kinematics of the Otter USV

In order to describe the position and orientation of a marine craft moving freely in 3 dimensions it's necessary to use 6 degrees of freedom (DOFs), 3 translational and 3 rotational components [26]. The 3 translational components are **surge**, **sway** and **heave**, while the 3 rotational components are **roll**, **pitch** and **yaw**. See Figure 3.1 for a visual representation of the 6 DOFs of the Otter USV.

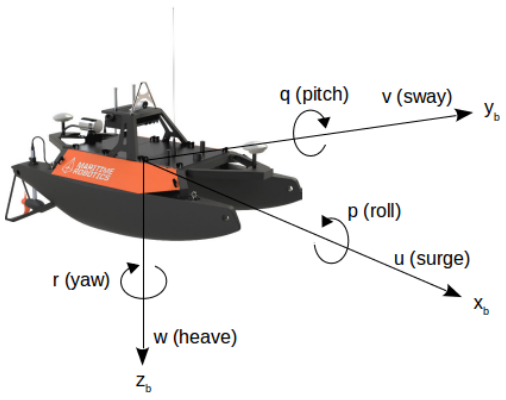


Figure 3.1: The 6 degrees of freedom for the Otter USV

The notation in Figure 3.1 is adopted from the Society of Naval Architects and Marine Engineers (SNAME) [46]. Table 3.1 gives a description for each of the components.

DOF		Forces and moments	Linear and angular velocities	Positions and Euler angles
1	motions in the x-direction (surge)	X	u	x
2	motions in the y-direction (sway)	Y	v	y
3	motions in the z-direction (heave)	Z	w	z
4	rotation about the x-axis (roll)	K	p	ϕ
5	rotation about the y-axis (pitch)	M	q	θ
6	rotation about the z-axis (yaw)	N	r	ψ

Table 3.1: Notation from SNAME [46]

3.2 The Otter USV model

The Otter USV model used in this thesis had the following representation [26, p. 13]:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) + \mathbf{g}_0 = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{wind}} + \boldsymbol{\tau}_{\text{wave}} \tag{3.1}$$

with $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ defined as

$$\boldsymbol{\nu} = [u, v, w, p, q, r]^\top \tag{3.2a}$$
$$\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^\top \tag{3.2b}$$
$$\tag{3.2c}$$

where $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ are generalized velocities and positions used to describe motions in 6 DOF. $\boldsymbol{\tau}$ are the generalized forces acting on the craft. In this model $\mathbf{M}$, $\mathbf{C}(\boldsymbol{\nu})$ and $\mathbf{D}(\boldsymbol{\nu})$ denotes the inertia, Coriolis and damping matrices respectively, $\mathbf{g}(\boldsymbol{\eta})$ is the generalized gravitational and buoyancy force-matrix and $\mathbf{g}_0$ consists of static restoring forces and moments due to ballast systems and water tanks [26, p. 13]. A table containing the value of the physical parameters of the Otter USV can be found in Appendix A.

3.2.1 Inertia matrices

In order to find $\mathbf{M}$ and $\mathbf{C}(\boldsymbol{\nu})$ the rigid-body inertia matrix $\mathbf{M}_{\text{RB}}$ and the rigid-body Coriolis and centripetal forces-matrix $\mathbf{C}_{\text{RB}}(\boldsymbol{\nu})$ in CG are calculated [26, p. 49]:

$$\mathbf{M}_{\text{RB}}^{\text{CG}} = \begin{bmatrix} (\text{m} + \text{m}_\text{p})\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_g \end{bmatrix} \tag{3.3a}$$
$$\mathbf{C}_{\text{RB}}^{\text{CG}} = \begin{bmatrix} (\text{m} + \text{m}_\text{p})\mathbf{S}(\omega_{\text{b/n}}^{\text{b}}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -\mathbf{S}(\mathbf{I}_\text{g}\omega_{\text{b/n}}^{\text{b}}) \end{bmatrix} \tag{3.3b}$$
$$\tag{3.3c}$$

where

$$\boldsymbol{\omega}_{\text{b/n}}^{\text{b}} = [p, q, r]^\top \tag{3.4}$$

and $\mathrm{m_p}$ is the payload mass for the Otter USV, $\mathbf{S}(\mathrm{x})$ is the skew-symmetric matrix of x and $\mathbf{I}_\mathrm{g}$ is the inertia matrix. $\mathbf{I}_\mathrm{g}$ was defined as

$$\mathbf{I}_\mathrm{g} := \begin{bmatrix} \mathrm{I}_x & -\mathrm{I}_{xy} & -\mathrm{I}_{xz} \\ -\mathrm{I}_{yx} & \mathrm{I}_y & -\mathrm{I}_{yz} \\ -\mathrm{I}_{yz} & -\mathrm{I}_{zy} & \mathrm{I}_z \end{bmatrix} = \mathrm{m} \begin{bmatrix} \mathrm{R}_{44}^2 & 0 & 0 \\ 0 & \mathrm{R}_{55}^2 & 0 \\ 0 & 0 & \mathrm{R}_{66}^2 \end{bmatrix} \tag{3.5}$$

where R_{44}, R_{55} and R_{66} are the radii of gyration. $\mathbf{M}_\mathrm{RB}^\mathrm{CG}$ and $\mathbf{C}_\mathrm{RB}^\mathrm{CG}$ was transformed from CG to CO as follows

$$\mathbf{M}_\mathrm{RB}^\mathrm{CO} = \mathbf{H}^\top(\mathbf{r}_\mathrm{g}^\mathrm{b})\mathbf{M}_\mathrm{RB}^\mathrm{CG}\mathbf{H}(\mathbf{r}_\mathrm{g}^\mathrm{b}) \tag{3.6a}$$

$$\mathbf{C}_\mathrm{RB}^\mathrm{CO}(\boldsymbol{\nu}) = \mathbf{H}^\top(\mathbf{r}_\mathrm{g}^\mathrm{b})\mathbf{C}_\mathrm{RB}^\mathrm{CG}\mathbf{H}(\mathbf{r}_\mathrm{g}^\mathrm{b}) \tag{3.6b}$$

where $\mathbf{H}(\mathbf{r}_\mathrm{g}^\mathrm{b})$ is the transformation matrix defined as

$$\mathbf{H}(\mathbf{r}_\mathrm{g}^\mathrm{b}) := \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{S}^\top(\mathbf{r}_\mathrm{g}^\mathrm{b}) \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix}, \quad \mathbf{H}^\top(\mathbf{r}_\mathrm{g}^\mathrm{b}) = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{S}(\mathbf{r}_\mathrm{g}^\mathrm{b}) & \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.7}$$

A marine vessel has to take the resistance of the fluid into account when finding the $\mathbf{M}$ and $\mathbf{C}(\boldsymbol{\nu})$ matrices. This is done by including hydrodynamic added mass $\mathbf{M}_A$ and $\mathbf{C}_A(\boldsymbol{\nu})$. These matrices were found using the following equations [26, p. 118-121]

$$\mathbf{M}_\mathrm{A} = - \begin{bmatrix} \mathrm{X}_{\dot u} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathrm{Y}_{\dot v} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{Z}_{\dot w} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{K}_{\dot p} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathrm{M}_{\dot q} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathrm{N}_{\dot r} \end{bmatrix} \tag{3.8}$$

$$\mathbf{C}_\mathrm{A}(\boldsymbol{\nu}) = - \begin{bmatrix} 0 & 0 & 0 & 0 & -\mathrm{Z}_{\dot w}w & \mathrm{Y}_{\dot v}v \\ 0 & 0 & 0 & \mathrm{Z}_{\dot w}w & 0 & -\mathrm{X}_{\dot u}u \\ 0 & 0 & 0 & -\mathrm{Y}_{\dot v}v & \mathrm{X}_{\dot u}u & 0 \\ 0 & -\mathrm{Z}_{\dot w}w & \mathrm{Y}_{\dot v}v & 0 & -\mathrm{N}_{\dot r}r & \mathrm{M}_{\dot q}q \\ \mathrm{Z}_{\dot w}w & 0 & -\mathrm{X}_{\dot u}u & \mathrm{N}_{\dot r}r & 0 & -\mathrm{K}_{\dot p}p \\ -\mathrm{Y}_{\dot v}v & \mathrm{X}_{\dot u}u & 0 & -\mathrm{M}_{\dot q}q & \mathrm{K}_{\dot p}p & 0 \end{bmatrix} \tag{3.9}$$

Where the following assumptions were made

$$\begin{aligned}
\mathrm{X}_{\dot{u}} &= -0.1 \cdot \mathrm{m} && (3.10a)\\
\mathrm{Y}_{\dot{v}} &= -1.5 \cdot \mathrm{m} && (3.10b)\\
\mathrm{Z}_{\dot{w}} &= -1.0 \cdot \mathrm{m} && (3.10c)\\
\mathrm{K}_{\dot{p}} &= -0.2 \cdot \mathrm{R}_{44} && (3.10d)\\
\mathrm{M}_{\dot{q}} &= -0.8 \cdot \mathrm{R}_{55} && (3.10e)\\
\mathrm{N}_{\dot{r}} &= -1.7 \cdot \mathrm{R}_{66} && (3.10f)\\
& && (3.10g)
\end{aligned}$$

$\mathbf{M}$ and $\mathbf{C}(\boldsymbol{\nu})$ are then found by summing the rigid-body and added mass matrices

$$\begin{aligned}
\mathbf{M} &= \mathbf{M}_{\mathrm{RB}}^{\mathrm{CO}} + \mathbf{M}_{\mathrm{A}} && (3.11a)\\
\mathbf{C}(\boldsymbol{\nu}) &= \mathbf{C}_{\mathrm{RB}}^{\mathrm{CO}}(\boldsymbol{\nu}) + \mathbf{C}_{\mathrm{A}}(\boldsymbol{\nu}) && (3.11b)
\end{aligned}$$

3.2.2 Restoring forces

Since the Otter USV is modeled in 6 DOF, the motions in heave, roll and pitch can't be represented by a zero-frequency model. The natural frequencies in these second-order mass-damper-spring systems are dominating and needs to be modeled by the following equations

$$\begin{aligned}
\omega_{\mathrm{heave}} &= \sqrt{\frac{\mathrm{G}_{33}}{\mathrm{M}_{33}}} && (3.12a)\\
\omega_{\mathrm{roll}} &= \sqrt{\frac{\mathrm{G}_{44}}{\mathrm{M}_{44}}} && (3.12b)\\
\omega_{\mathrm{pitch}} &= \sqrt{\frac{\mathrm{G}_{55}}{\mathrm{M}_{55}}} && (3.12c)
\end{aligned}$$

with G_{33}, G_{44} and G_{55} calculated as follows

$$\begin{aligned}
\mathrm{G}_{33} &= 2\rho \mathrm{g} \mathrm{A}_{w,\mathrm{pont}} && (3.13a)\\
\mathrm{G}_{44} &= \rho \mathrm{g} \nabla \overline{\mathrm{GM}}_{\mathrm{T}} && (3.13b)\\
\mathrm{G}_{55} &= \rho \mathrm{g} \nabla \overline{\mathrm{GM}}_{\mathrm{L}} && (3.13c)
\end{aligned}$$

where $\overline{\mathrm{GM}}_{\mathrm{T}}$ and $\overline{\mathrm{GM}}_{\mathrm{L}}$ are the traverse and longitudinal metacentric height, and ∇ and $\mathrm{A}_{w,\mathrm{pont}}$ are given by

$$\nabla = \frac{\mathrm{m} + \mathrm{m_p}}{\rho} \tag{3.14a}$$

$$\mathrm{A}_{w,\mathrm{pont}} = \mathrm{C}_{w,\mathrm{pont}} \cdot \mathrm{L} \cdot \mathrm{B}_{\mathrm{pont}} \tag{3.14b}$$

This can then be used to find the restoring matrix $\mathbf{G}^{\mathrm{CF}}$ in Center of Force (CF) [26, p. 181]

$$\mathbf{G}^{\mathrm{CF}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{G}_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{G}_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathrm{G}_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.15}$$

Which has to be transformed to CO by using the transformation matrix $\mathbf{H}(\mathbf{r}_{\mathrm{f}}^{\mathrm{b}})$

$$\mathbf{G} = \mathbf{H}^{\top}(\mathbf{r}_{\mathrm{f}}^{\mathrm{b}})\mathbf{G}^{\mathrm{CF}}\mathbf{H}(\mathbf{r}_{\mathrm{f}}^{\mathrm{b}}) \tag{3.16}$$

where $\mathbf{r}_{\mathrm{f}}^{\mathrm{b}} = \begin{bmatrix} -0.2 & 0 & 0 \end{bmatrix}$ is the distance from CF to CO. This can then be used to find $\mathbf{g}(\boldsymbol{\eta})$ in (3.1).

$$\mathbf{g}(\boldsymbol{\eta}) \approx \mathbf{G}\boldsymbol{\eta} \tag{3.17}$$

Lastly, the forces and moments $\mathbf{g}_0$ due to the ballast tanks is given by the following equation [26, p. 75]

$$\mathbf{g}_0 = \begin{bmatrix} 0 \\ 0 \\ -\mathrm{Z}_{\mathrm{ballast}} \\ -\mathrm{K}_{\mathrm{ballast}} \\ -\mathrm{M}_{\mathrm{ballast}} \\ 0 \end{bmatrix} \tag{3.18}$$

where $\mathrm{Z}_{\mathrm{ballast}}$, $\mathrm{K}_{\mathrm{ballast}}$ and $\mathrm{M}_{\mathrm{ballast}}$ are the moments in heave, roll and pitch due to ballast. The value of these were found by manual pre-trimming as shown in the following equation [26, p. 76]

$$\mathbf{G}\boldsymbol{\eta} + \mathbf{g}_0 = 0 \tag{3.19}$$

3.2.3 Damping forces

The linear viscous damping matrix $\mathbf{D}(\boldsymbol{\nu})$ is given by

$$\mathbf{D}(\boldsymbol{\nu}) = -\begin{bmatrix} \mathrm{X}_u & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathrm{Y}_v & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{Z}_w & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{K}_p & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathrm{M}_q & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathrm{N}_r \end{bmatrix} \tag{3.20}$$

where the linear damping terms on the diagonal of the damping matrix $\mathbf{D}(\boldsymbol{\nu})$ are described by the following equations [26, p. 125]

$$-\mathrm{X}_u = B_{11v} = \frac{\mathrm{M}_{11}}{\mathrm{T}_{\text{surge}}} \tag{3.21a}$$
$$-\mathrm{Y}_v = B_{22v} = 0 \tag{3.21b}$$
$$-\mathrm{Z}_w = B_{33v} = 2\zeta_{\text{heave}}\ \omega_{\text{heave}} M_{33} \tag{3.21c}$$
$$-\mathrm{K}_p = B_{44v} = 2\zeta_{\text{roll}}\ \omega_{\text{roll}} M_{44} \tag{3.21d}$$
$$-\mathrm{M}_q = B_{55v} = 2\zeta_{\text{pitch}}\ \omega_{\text{pitch}} M_{55} \tag{3.21e}$$
$$-\mathrm{N}_r = B_{66v} = \frac{\mathrm{M}_{66}}{\mathrm{T}_{\text{yaw}}} \tag{3.21f}$$

3.2.4 Cross-flow drag for sway and yaw

The nonlinear damping force in sway and the yaw moment are found by [26]

$$\mathrm{Y} = -\frac{1}{2}\rho \int_{-\frac{\mathrm{L}}{2}}^{\frac{\mathrm{L}}{2}} T(x) C_d^{2D}(x) |v_r + xr| (v_r + xr) dx \tag{3.22a}$$
$$\mathrm{N} = -\frac{1}{2}\rho \int_{-\frac{\mathrm{L}}{2}}^{\frac{\mathrm{L}}{2}} T(x) C_d^{2D}(x) x |v_r + xr| (v_r + xr) dx \tag{3.22b}$$

where $v_r = v - v_c$ is the relative sway velocity and $C_d^{2D}(x)$ is calculated using the Hoerner function from the Matlab MSS toolbox [47].

3.2.5 Control allocation

As stated under assumptions in Section 1.6, wind and waves were neglected, meaning $\boldsymbol{\tau}_{\text{wind}} = \boldsymbol{\tau}_{\text{wave}} = 0$. The control forces were calculated using the following equation [26, p. 413]

$$\boldsymbol{\tau} = \mathbf{TKu} \tag{3.23}$$

where $\mathbf{T}$ is the actuator configuration matrix, $\mathbf{K}$ is a diagonal matrix of thrust coefficients and $\mathbf{u}$ is the control variable given by

$$\mathbf{u} = \begin{bmatrix} \mathrm{u}_1 \\ \mathrm{u}_1 \end{bmatrix} = \begin{bmatrix} \mathrm{n}_1|\mathrm{n}_1| \\ \mathrm{n}_2|\mathrm{n}_2| \end{bmatrix}^\top \tag{3.24}$$

where $\mathrm{n_i}$ is the propeller revolutions per minute (rpm). Since the thrusters only act on the surge and heading of the vessel, $\boldsymbol{\tau}$ can be described as

$$\boldsymbol{\tau} = [\tau_1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; \tau_6]^\top \tag{3.25}$$

where τ_1 and τ_6 are the control inputs for surge and yaw respectively. Furthermore, the thrust coefficients are equal for both of the thrusters, only depending on positive or negative rotation of the propellers. Using this in (3.23) gives

$$\begin{bmatrix} \tau_1 \\ \tau_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -\mathrm{l}_1 & -\mathrm{l}_2 \end{bmatrix} \begin{bmatrix} \mathrm{k}_1 & 0 \\ 0 & \mathrm{k}_2 \end{bmatrix} \begin{bmatrix} \mathrm{u}_1 \\ \mathrm{u}_2 \end{bmatrix} \tag{3.26}$$

where

$$\mathrm{l}_1 = -\mathrm{l}_2 = -\mathrm{Y}_{\text{pont}} \tag{3.27a}$$

$$\mathrm{k_i} = \begin{cases} \mathrm{k}_{\text{pos}} & \text{if } \mathrm{n_i} > 0 \\ \mathrm{k}_{\text{neg}} & \text{otherwise} \end{cases} \tag{3.27b}$$

Solving (3.26) for $\mathbf{u}$ yields the following

$$\begin{bmatrix} \mathrm{u}_1 \\ \mathrm{u}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{k}_1 & 0 \\ 0 & \mathrm{k}_2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ -\mathrm{l}_1 & -\mathrm{l}_2 \end{bmatrix}^{-1} \begin{bmatrix} \tau_1 \\ \tau_6 \end{bmatrix} \tag{3.28}$$

which can be rewritten using (3.24) such that the controller input for both of the controllers can be modeled as

$$\begin{bmatrix} \mathrm{n}_1 \\ \mathrm{n}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{sign}(\mathrm{u}_1)\sqrt{|\mathrm{u}_1|} \\ \mathrm{sign}(\mathrm{u}_2)\sqrt{|\mathrm{u}_2|} \end{bmatrix} \tag{3.29}$$

where n is bounded as follows

$$\mathrm{n}_{\mathrm{max}} = \sqrt{\frac{0.5 \cdot 24.4 \cdot \mathrm{g}}{\mathrm{k}_{\mathrm{pos}}}} \tag{3.30a}$$

$$\mathrm{n}_{\mathrm{min}} = \sqrt{\frac{0.5 \cdot 13.6 \cdot \mathrm{g}}{\mathrm{k}_{\mathrm{neg}}}} \tag{3.30b}$$

3.3 Path following

A Line of Sight (LOS) guidance law is commonly used for following paths [17]. By using a list of waypoints, the desired path can be generated as straight lines between waypoints $\mathbf{p}_{\mathrm{k}}^{\mathrm{n}} = [x_{\mathrm{k}}, y_{\mathrm{k}}]$ and $\mathbf{p}_{\mathrm{k+1}}^{\mathrm{n}} = [x_{\mathrm{k+1}}, y_{\mathrm{k+1}}]$, respectively. The desired course χ_{LOS} can then be calculated using the vessels path-tangential angle α_{k}, cross-track error $e(\mathrm{t})$ and lookahead distance Δ [26]:

$$\chi_{\mathrm{LOS}} = \alpha_{\mathrm{k}} + \arctan\left(\frac{-e(\mathrm{t})}{\Delta}\right), \qquad \Delta > 0 \tag{3.31a}$$

$$\alpha_{\mathrm{k}} = \mathrm{atan2}(y_{\mathrm{k+1}} - y_{\mathrm{k}}, x_{\mathrm{k+1}} - x_{\mathrm{k}}) \tag{3.31b}$$

$$e(\mathrm{t}) = -[x(\mathrm{t}) - x_{\mathrm{k}}]\sin(\alpha_{\mathrm{k}}) + [y(\mathrm{t}) - y_{\mathrm{k}}]\cos(\alpha_{\mathrm{k}}) \tag{3.31c}$$

where $\mathrm{atan2(y, x)}$ is the 4-quadrant inverse tangent confining the result to $(-\pi, \pi]$ [26]. The lookahead distance Δ is a design parameter, often chosen to be two times the length of the vessel. The waypoint $\mathbf{p}_{\mathrm{k}}^{\mathrm{n}}$ is updated when the vessel is in within a given radius of the desired waypoint. An illustration of the LOS guidance law is shown in Figure 3.2.

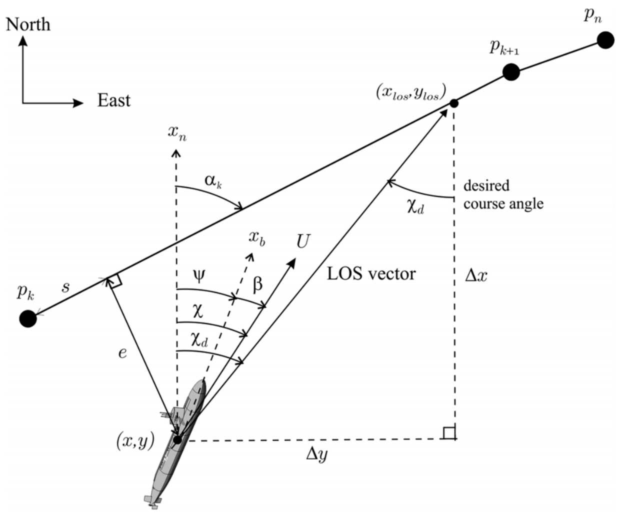


Figure 3.2: LOS guidance law with lookahead-based steering. Illustration byFossen [26]

3.4 Set-Based Guidance

First part of designing the guidance law for obstacle avoidance is to define a safe radius $\mathrm{R_o}$ around the obstacle center $\mathbf{p}_\mathrm{o}$ given by

$$\mathbf{p}_\mathrm{o}(\mathrm{t}) = [x_\mathrm{c}(\mathrm{t}) \quad y_\mathrm{c}(\mathrm{t})]^\top \tag{3.32}$$

As long as $\mathrm{R_o}$ is tracked, no collisions will ever occur. Given that the obstacles are static, the obstacle velocity can be neglected. Furthermore, the following is denoted [17]

$$\phi = \arctan\left(\frac{y - y_\mathrm{c}}{x - x_\mathrm{c}}\right) \tag{3.33}$$

The cross-track error of the circular path is given by

$$\mathrm{e} = \mathrm{R_o} - \rho = \mathrm{R_o} - \sqrt{(x - x_\mathrm{c})^2 + (y - y_\mathrm{c})^2} \tag{3.34}$$

This is used to calculate the desired heading for obstacle avoidance by using the following guidance law [17]

$$\psi_{\mathrm{oa}} = \phi + \lambda \left(\frac{\pi}{2} - \arctan\left(\frac{\mathrm{e} + \mathrm{k}}{\Delta} \right) \right) - \arctan\left(\frac{v}{u_{\mathrm{oa}}} \right) \tag{3.35}$$

where $\lambda = -1$ corresponds to counter-clockwise motion and $\lambda = 1$ to clock-wise motion, and u_{oa} is the desired surge velocity for obstacle avoidance. Furthermore, the parameter k due to the obstacles being static. The parameters used in the obstacle avoidance guidance law are illustrated in Figure 3.3.

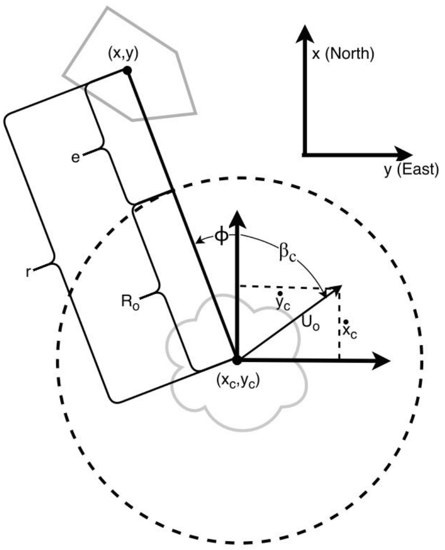


Figure 3.3: Figure illustrating parameters used for obstacle avoidance [17]

When the USV switches from path following to obstacle avoidance, it should choose λ based on the current heading in order to avoid sharp turns. This can be done by choosing λ as follows [17]

$$\lambda = \begin{cases} -1 & \text{if } |\psi - \psi_{\mathrm{oa,cc}}| \leq |\psi - \psi_{\mathrm{oa,c}}| \\ 1 & \text{if } |\psi - \psi_{\mathrm{oa,cc}}| > |\psi - \psi_{\mathrm{oa,c}}| \end{cases} \tag{3.36}$$

where $\psi_{\mathrm{oa,c}}$ and $\psi_{\mathrm{oa,cc}}$ denotes ψ_{oa} from (3.35) calculated in clock-wise and counter-clockwise motion respectively.

3.4.1 Switching mechanism

In order to switch between obstacle avoidance and path following, a set-based control method was used [17]. The distance between an obstacle center and the

USV is given by ρ in (3.34):

$$\sigma = \rho = \sqrt{(x - x_c)^2 + (y - y_c)^2} \tag{3.37}$$

Furthermore, its derivative is given by

$$\dot{\sigma} = \frac{2(x - x_c)(\dot{x} - \dot{x}_c) + 2(y - y_c)(\dot{y} - \dot{y}_c)}{2\sqrt{(x - x_c)^2 + (y - y_c)^2}} = \frac{(x - x_c)\dot{x} + (y - y_c)\dot{y}}{\rho} \tag{3.38}$$

Next, a mode change radius around the obstacle is introduced as $R_m > R_o$. As long as the USV is outside this radius, it will actively follow the path. Path following will also be active if the USV is inside R_m, as long as this either increases or maintains the distance from the obstacle, i.e. if $\dot{\sigma} \geq 0$. The mode change radius should be chosen large enough such that the USV can converge to R_o when switching to obstacle avoidance without overshooting. This can be done by using a tangent cone T_D to the set $D = [\sigma_{min}, \sigma_{max}]$ at the point $\sigma \in D$ [17]:

$$T_D(\sigma) = \begin{cases} [0, \infty) & \text{if } \sigma = \sigma_{min} \\ \mathbb{R} & \text{if } \sigma \in (\sigma_{min}, \sigma_{max}) \\ (-\infty, 0] & \text{if } \sigma = \sigma_{max} \end{cases} \tag{3.39}$$

From this, it follows that $\dot{\sigma}(t) \in T_D(\sigma(t))$, which implies that $\sigma(t) \in D$ for $t \geq t_0$. This means that a valid set D can be defined such that the USV will keep following the path as long as σ and $\dot{\sigma}$ is in the tangent cone of D. Moe and Pettersen suggests defining D as [17]

$$D = [\min(R_m, \max(\sigma, R_o)), \infty) \tag{3.40}$$

The distance between an obstacle and the USV will always be greater than or equal to R_o as long as $\sigma \in D$.

3.5 A* path re-planning

Since the obstacles are assumed to be static, it's possible to use a shortest path algorithm such as A* in order to avoid them. The first part of the A* algorithm is to define the obstacles, start pixel and end pixel. The algorithm operates using two lists; an *open list* and an *closed list*. All of the reachable pixels are inserted to the *open list*, excluding obstacle pixels and pixels that has been "visited". The path that results in the lowest cost is then added to the *closed list.* The cost for each step is found by calculating F in the following equation [48]

$$F = G + H \tag{3.41}$$

where G is the cost to move from the start pixel to a given pixel and H is the estimated movement cost from the given pixel to the end pixel [15]. The pixel from the *open list* that leads to the lowest F cost is moved to the *closed list.* The path is generated by continuously moving towards the goal pixel and selecting the pixel with the lowest F cost.

3.5.1 Locating waypoints

Using A* will result in a path consisting of each pixel of the optimal movement, an example of this can be seen in Figure 3.4 where the open list is illustrated in blue, the open list in green, the final path in red and the obstacle in white.

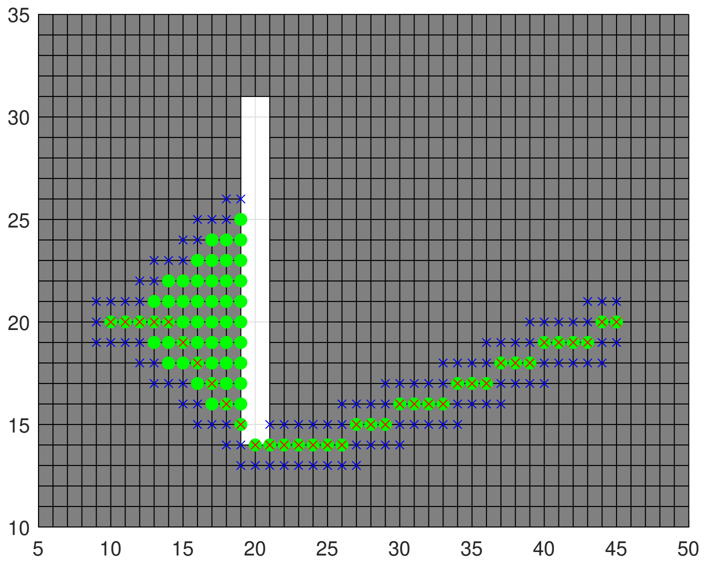


Figure 3.4: Illustration of the A* algorithm with the open list, closed list and final path illustrated, algorithm from [49]

However, for path following, only the final path is of interest. A simplified plot with the open and closed list excluded is shown in Figure 3.5.

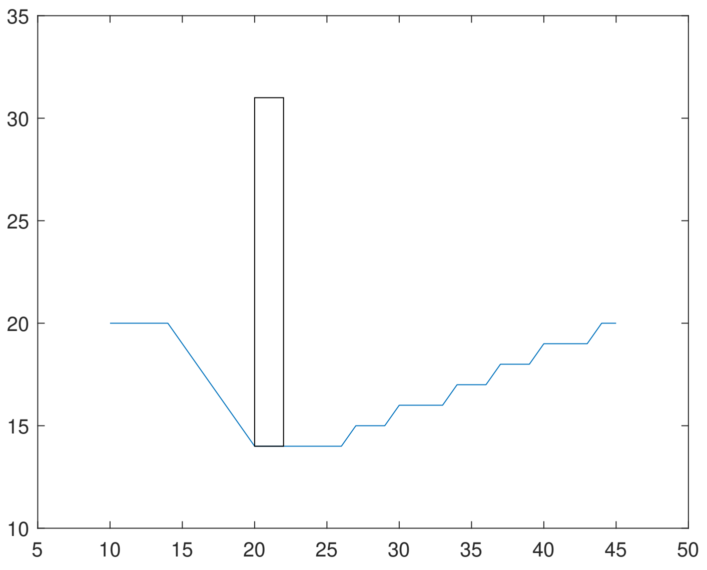


Figure 3.5: Simplified illustration of the A* algorithm showing the obstacle and the final path

Waypoints can then be generated by locating the extrema/turning points of the path as illustrated in Figure 3.6. These waypoints can be used in a guidance law such that the USV is able to avoid obstacles.

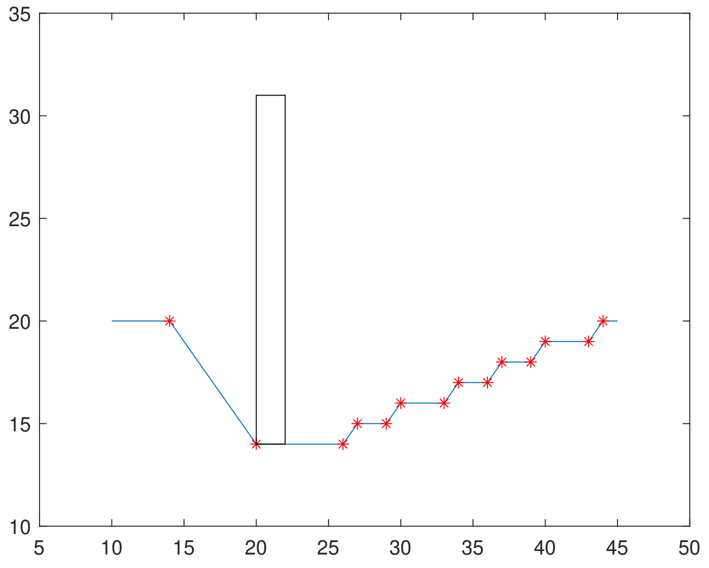


Figure 3.6: Figure showing the path from A* with the waypoints illustrated in red

3.6 PI surge and PID course controllers

When designing the PI surge and PID course controllers for the Otter USV, the motions in surge, sway and yaw were considered decoupled. The forward speed is considered slowly varying and the sideways motion v is assumed small such that $\mathrm{U} = \sqrt{u^2 + v^2} \approx u$. The two motions can therefore be linearly modeled as

$$(\mathrm{m} + \mathrm{m_p} - \mathrm{X}_{\dot{u}})\dot{u} + \mathrm{X}_u u = \tau_1 \tag{3.42a}$$

$$(\mathrm{I}_z - \mathrm{N}_{\dot{r}})\ddot{\psi} + \mathrm{N}_r \dot{\psi} = \tau_6 \tag{3.42b}$$

where (3.42b) describes the motion as a function of the heading, this equation can be rewritten to describe the motion as a function of the course under the assumption that β_c is constant since the vessel is mostly travelling in straight lines:

$$(\mathrm{I}_z - \mathrm{N}_{\dot{r}})\ddot{\chi} + \mathrm{N}_r \dot{\chi} = \tau_6 \tag{3.43}$$

From where the following surge and course controllers can be designed as PI and PID controllers, respectively, both with reference feedforward [26, p. 372]:

$$\tau_1 = (\mathrm{m} + \mathrm{m_p} - \mathrm{X}_{\dot{u}})\dot{u}_\mathrm{d} + \mathrm{X}_u u_\mathrm{d} - \mathrm{K}_{\mathrm{p},u}\tilde{u} - \mathrm{K}_{\mathrm{i},u}\int_0^\mathrm{t} \tilde{u}(\tau)\mathrm{d}\tau \tag{3.44a}$$

$$\tau_6 = (\mathrm{I}_z - \mathrm{N}_{\dot{r}})\ddot{\chi}_\mathrm{d} + \mathrm{N}_r\dot{\chi}_\mathrm{d} - \mathrm{K}_{\mathrm{p},\chi}\tilde{\chi} - \mathrm{K}_{\mathrm{d},\chi}\dot{\tilde{\chi}} - \mathrm{K}_{\mathrm{i},\chi}\int_0^\mathrm{t} \tilde{\chi}(\tau)\mathrm{d}\tau \tag{3.44b}$$

where $\tilde{u} = u - u_\mathrm{d}$ and $\tilde{\chi} = \chi - \chi_\mathrm{d}$ are the error in surge and course respectively. This gives the following when inserted in (3.42)

$$(\mathrm{m} + \mathrm{mp} - \mathrm{X}_{\dot{u}})\dot{\tilde{u}} + (\mathrm{X}_u + \mathrm{K}_{\mathrm{p},u})\tilde{u} + \mathrm{K}_{\mathrm{i},u}\int_0^\mathrm{t} \tilde{u}(\tau)\mathrm{d}\tau = 0 \tag{3.45a}$$

$$(\mathrm{I}_z - \mathrm{N}_{\dot{r}})\ddot{\tilde{\chi}} + (\mathrm{N}_r + \mathrm{K}_{\mathrm{d},\chi})\dot{\tilde{\chi}} + \mathrm{K}_{\mathrm{p},\chi}\tilde{\chi} + \mathrm{K}_{\mathrm{i},\chi}\int_0^\mathrm{t} \tilde{\chi}(\tau)\mathrm{d}\tau = 0 \tag{3.45b}$$

3.7 First-order Sliding Mode Control

The first part of designing a first-order SMC (FSMC) is to define the sliding surface variable s, which is dependent on the system error dynamics:

$$\dot{\tilde{\chi}} = \tilde{\mathrm{r}} \tag{3.46a}$$

$$(\mathrm{I}_z - \mathrm{N}_{\dot{r}})\dot{\tilde{r}} + \mathrm{N}_r\tilde{r} = \tau_6 \tag{3.46b}$$

from which s can be chosen as [25]

$$\mathrm{s} = \tilde{r} + \lambda\tilde{\chi} = 0, \qquad \lambda > 0 \tag{3.47}$$

In order to achieve convergence of the state variables $\tilde{\chi}$ and $\tilde{\mathrm{r}}$ towards zero, it's necessary to drive s to zero, within finite time, by using the control τ_6 [35]. To find values for τ_6 at which this is achieved, it's necessary to analyze the stability of the s-dynamics. Consider the following candidate Lyapunov function

$$\mathrm{V(s)} = \frac{1}{2}\mathrm{s}^2 \tag{3.48}$$

where the derivative of s is given by

$$\dot{\mathrm{s}} = \lambda\dot{\tilde{\chi}} + \dot{\tilde{r}} = \lambda\tilde{r} - (\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}\mathrm{N}_r\tilde{r} + (\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}\tau_6 \tag{3.49}$$

where the following holds true

$$\left|\frac{\lambda\tilde{r} - (\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}\mathrm{N}_r\tilde{r}}{(\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}}\right| = |(\lambda(\mathrm{I}_z - \mathrm{N}_{\dot{r}}) - \mathrm{N}_r)\tilde{r}| \leq \varrho(\chi), \qquad \forall\, \chi \in \mathrm{R}^2 \tag{3.50}$$

with $\varrho(\chi)$ as a known function [25]. This yields the following when differentiating V along the trajectories of s

$$\dot{\mathrm{V}}(\mathrm{s}) = \mathrm{s}\dot{\mathrm{s}} = \mathrm{s}(\lambda\tilde{r} - (\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}\mathrm{N}_r\tilde{r}) + (\mathrm{I}_z - \mathrm{N}_{\dot{r}})^{-1}\mathrm{s}\tau_6 \leq |\mathrm{s}|\varrho(\chi) + \mathrm{s}\tau_6 \tag{3.51}$$

From which the control law can be chosen as

$$\tau_6 = -\beta(\chi)\mathrm{sign}(\mathrm{s}) \tag{3.52}$$

with $\beta(\chi) \geq \varrho(\chi) + \beta_0$, $\beta_0 > 0$, and sign(s) given by

$$\mathrm{sign}(\mathrm{s}) = \begin{cases} 1 & \text{if s} > 0 \\ 0 & \text{if s} = 0 \\ -1 & \text{if s} < 0 \end{cases} \tag{3.53}$$

Inserting this control law for τ_6 in (3.51) yields the following

$$\dot{\mathrm{V}}(\mathrm{s}) \leq |\mathrm{s}|\varrho(\chi) - \mathrm{s}(\varrho(\chi) + \beta_0)\mathrm{sign}(\mathrm{s}) = -\beta_0|\mathrm{s}| \leq 0 \tag{3.54}$$

From which it can be concluded that the trajectory reaches s=0 within finite time, and that it does not leave once reached [25]. From (3.54) it can be concluded that the origin is asymptotically stable, which in turn means that the origin of the error dynamics of the course is asymptotically stable. Since this control-law is designed as a course controller, it's not possible to guarantee global results. This is due to the fact that errors in the roll, pitch and yaw are in SO(3), which means that the angles are defined from 0 to 2π and not the whole space in $\mathbb{R}$ [50]. This holds true for all course controllers designed in this thesis.

3.7.1 Chattering

The main problem of the SMC control-law is that it leads to a effect known as chattering, which is shown in Figure 3.7.

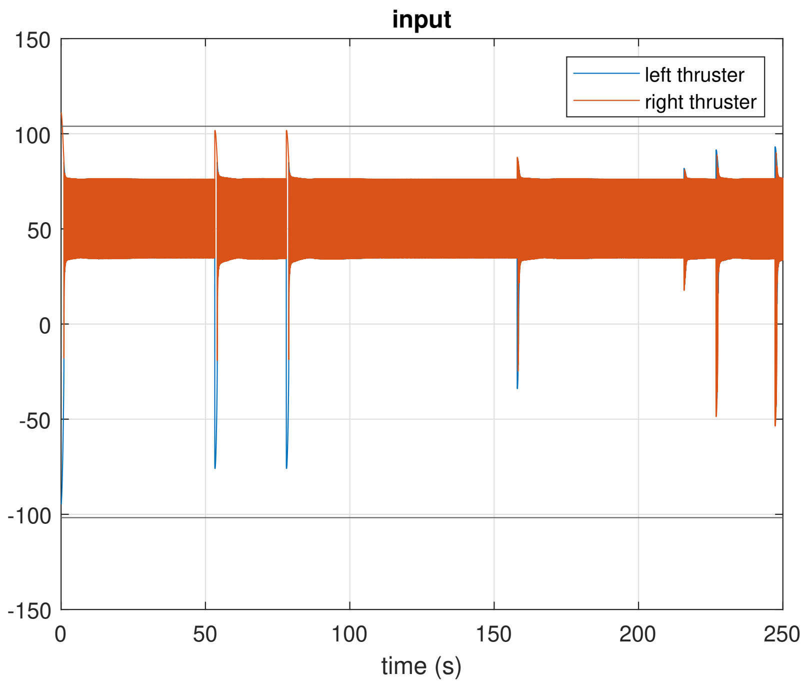


Figure 3.7: An example showing how the controller input suffers from chattering

Chattering comes from the fact that there is a delay between when the sign of s changes and when the control switches. The trajectory can then cross the sliding surface during this delay, such that it is in the region $s < 0$ or $s > 0$ and not $s = 0$. This causes a "zig-zag" motion for the trajectory where the trajectory keeps crossing the sliding surface when the control switches [25]. This effect is illustrated in Figure 3.8 (sliding manifold=sliding surface).

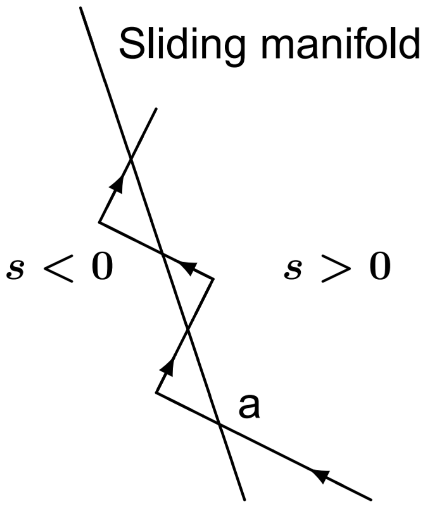


Figure 3.8: Chattering due to delay in control switching, illustration from Khalil [25]

One way to reduce chattering is by replacing sign(s) with a continuous function. One such function the a *sigmoid function* [35]

$$\text{sign(s)} \approx \frac{\text{s}}{|\text{s}| + \varepsilon} \tag{3.55}$$

where ε is a positive constant. Another way to reduce chattering is by replacing the signum function with a high-slope saturation function given by

$$\text{sat}\left(\frac{\text{s}}{\varepsilon}\right) = \begin{cases} \frac{\text{s}}{\varepsilon} & \text{if } \left|\frac{\text{s}}{\varepsilon}\right| \leq 1 \\ \text{sign(s)} & \text{if } \left|\frac{\text{s}}{\varepsilon}\right| > 1 \end{cases} \tag{3.56}$$

where $\varepsilon > 0$ can be described as the boundary layer thickness [26]. Figure 3.9 shows how the chattering from Figure 3.7 was reduced using these methods.

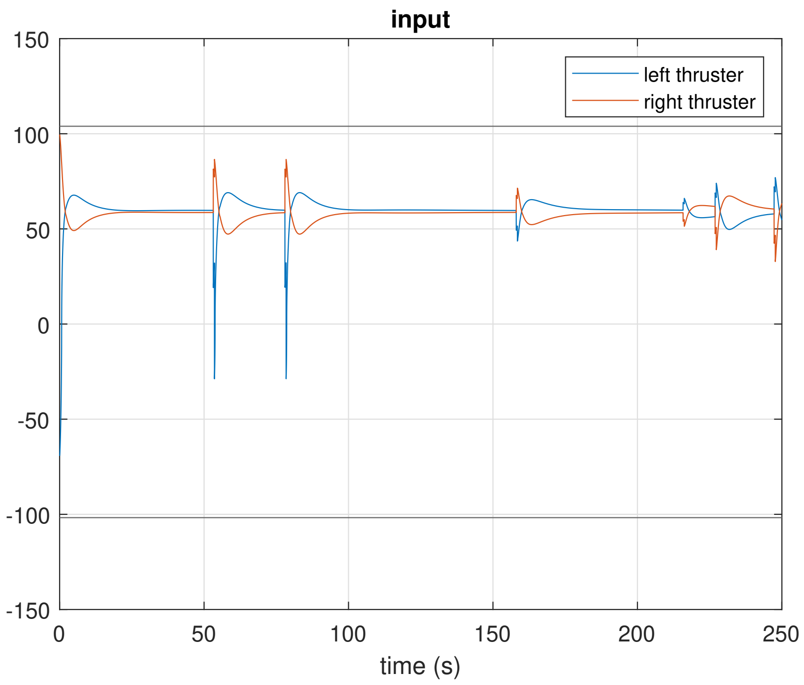


Figure 3.9: Same example as in Figure 3.7 with chattering reduced

3.8 PID-Sliding Mode Controller

The FSMC in Section 3.7 only consists of a single gain as a control law, which limits the performance of the controller. The aforementioned controller can be improved by including more of the system dynamics in addition to derivative and integral parts to the control-law, which results in a higher order SMC. This controller will be referred to as a PID-SMC, since it's a SMC based on a PID controller. The following sliding surface will be considered for the PID-SMC [26]

$$\mathrm{s} = \tilde{r} + 2\lambda\tilde{\chi} + \lambda^2 \int_0^{\mathrm{t}} \tilde{\chi}(\tau)\mathrm{d}\tau \tag{3.57}$$

where λ is a design parameter that reflects the controller bandwidth. Next, the state space equation for course can be written as

$$\dot{\chi} = r \tag{3.58a}$$

$$\mathrm{T}\dot{r} + r = \mathrm{K}\tau_6 \tag{3.58b}$$

where K and T are given by

$$\mathrm{K} = \frac{1}{\mathrm{N}_r}, \qquad \mathrm{T} = \frac{\mathrm{I}_z - \mathrm{N}_{\dot{r}}}{\mathrm{N}_r} \tag{3.59}$$

The next step is then to define a virtual reference variable v and to calculate its derivative [26]:

$$\mathrm{v} := r - \mathrm{s} = r - \tilde{r} - 2\lambda\tilde{\chi} - \lambda^2 \int_0^{\mathrm{t}} \tilde{\chi}(\tau)\mathrm{d}\tau = r_{\mathrm{d}} - 2\lambda\tilde{\chi} - \lambda^2 \int_0^{\mathrm{t}} \tilde{\chi}(\tau)\mathrm{d}\tau \tag{3.60a}$$

$$\dot{\mathrm{v}} = \dot{r}_{\mathrm{d}} - 2\lambda\tilde{r} - \lambda^2\tilde{\chi} \tag{3.60b}$$

By using (3.60), the following equation for $\mathrm{T}\dot{\mathrm{s}}$ can be found

$$\mathrm{T}\dot{\mathrm{s}} = \mathrm{T}\dot{r} - \mathrm{T}\dot{\mathrm{v}} = \mathrm{K}\tau_6 - r - \mathrm{T}\dot{\mathrm{v}} \tag{3.61a}$$

$$= \mathrm{K}\tau_6 - (\mathrm{v} + \mathrm{s}) - \mathrm{T}\dot{\mathrm{v}} \tag{3.61b}$$

Similar to the FSMC, the control τ_6 has to be found such that the state variables converges towards zero. Considering the following candidate Lyapunov function [26]

$$\mathrm{V}(\mathrm{s}) = \frac{1}{2}\mathrm{T}\mathrm{s}^2 \tag{3.62}$$

which yields the following when differentiating along the trajectories of s

$$\dot{\mathrm{V}}(\mathrm{s}) = \mathrm{s}\mathrm{T}\dot{\mathrm{s}} = \mathrm{s}(\mathrm{K}\tau_6 - (\mathrm{v} + \mathrm{s}) - \mathrm{T}\dot{\mathrm{v}}) \tag{3.63}$$

From which the following control law can be chosen

$$\tau_6 = \frac{\mathrm{T}}{\mathrm{K}}\dot{\mathrm{v}} + \frac{1}{\mathrm{K}}\mathrm{v} - \mathrm{K}_{\mathrm{d}}\mathrm{s} - \eta\,\mathrm{sat}(\mathrm{s}) \tag{3.64}$$

where $\mathrm{K}_{\mathrm{d}} > 0$ and η are design parameters with η bounded by [26]

$$\eta \geq \mathrm{r}_1\frac{\mathrm{T}}{\mathrm{K}}|\dot{\mathrm{v}}| + \mathrm{r}_2\frac{1}{\mathrm{K}}|\mathrm{v}| \tag{3.65}$$

where r_{i} represents the percentage of uncertainty for each element (i.e. 1.1 represents 10% uncertainty). Inserting this control-law for τ_6 in (3.63) yields

$$\dot{\mathrm{V}}(\mathrm{s}) = \mathrm{s}(-\mathrm{K_d s} - \eta\,\mathrm{sat(s)}) = -\mathrm{K_d s^2} - \eta|\mathrm{s}| < 0, \qquad \forall\,\mathrm{s} \neq 0 \tag{3.66}$$

which is locally exponentially stable in s=0 due to the non-positive term $-\mathrm{K_d s^2}$. Furthermore, since $\eta > 0$, it can be concluded that the trajectory converges to the sliding surface $\mathrm{s} \to 0$ within finite time [26].

3.9 Super-Twisting Controller

In order to get a SMC that reduces the chattering effect without using sat(s), one can use a Super-Twisting Controller (STC). The STC is a second order SMC which ensures that both s and $\dot{s}$ tends to zero within finite time, resulting in a reduced chattering effect. The STC does this by generating a continuous control function. Though it does not eliminate the chattering effect completely since it contains an integral part ($\mathrm{v_{stc}}$) with a discontinuous function (sign(s)). Similarly to the FSMC, the first part of designing a STC is to define the sliding surface, but it is important that the sliding surface has a relative degree of 1. In order to find the relative degree of the sliding surface it has to be differentiated until the controller appears in the equation:

$$\mathrm{s} = \tilde{r} + \lambda\tilde{\chi} \tag{3.67a}$$

$$\dot{\mathrm{s}} = \dot{\tilde{r}} + \lambda\tilde{r} \tag{3.67b}$$

It can be seen that the controller appears in the equation for $\dot{\tilde{\mathrm{r}}}$ after one differentiation, meaning that the sliding surface has a relative degree of 1 and can therefore be used. The STC control-law is given by [35]

$$\tau_6 = -\mathrm{k_1}|\mathrm{s}|^{\frac{1}{2}}\mathrm{sign(s)} + \mathrm{v_{stc}} \tag{3.68}$$

with

$$\dot{\mathrm{v}}_{\mathrm{stc}} = -\mathrm{k_2 sign(s)} \tag{3.69}$$

where $\mathrm{k_1} > 0$ and $\mathrm{k_2} > 0$ are design parameters. One way to design $\mathrm{k_1}$ and $\mathrm{k_2}$ in order to achieve convergence of $\mathrm{s} = \dot{\mathrm{s}} = 0$ is given by Levant [37]

$$\mathrm{k_2} > \mathrm{C}, \qquad \mathrm{k_1^2} \geq 4\mathrm{C}\frac{\mathrm{k_2} + \mathrm{C}}{\mathrm{k_2} - \mathrm{C}} \tag{3.70}$$

with $\mathrm{C} > 0$.

3.9.1 Adaptive Gain STC

Although choosing gains as shown in the previous section can yield acceptable results, the controller can be improved by using adaptive gains for the STC. For the sliding surface

$$\mathrm{s} = \tilde{r} + \lambda_1 \tilde{\chi} \tag{3.71}$$

with $\lambda_1 > 0$, the following adaptive gain STC control-law have been proposed [51]

$$\begin{aligned} \tau_6 &= -\alpha |\mathrm{s}|^{1/2} \mathrm{sign}(\mathrm{s}) + \mathrm{v}_{\mathrm{stc}} && (3.72a) \\ \dot{\mathrm{v}}_{\mathrm{stc}} &= -\beta \mathrm{sign}(\mathrm{s}) && (3.72b) \\ & && (3.72c) \end{aligned}$$

where α and β are the adaptive gains chosen by the following equations

$$\begin{aligned} \dot{\alpha} &= \begin{cases} \omega \sqrt{\frac{\gamma}{2}}, & \text{if } \mathrm{s} \neq 0 \\ 0, & \text{if } \mathrm{s} = 0 \end{cases} && (3.73a) \\ \beta &= 2\varepsilon\alpha + \lambda_2 + 4\varepsilon^2 && (3.73b) \end{aligned}$$

where ε, λ_2, γ and ω are positive constants to be tuned. In order to reduce chattering, a boundary layer can be applied to the sliding surface [52]

$$\begin{aligned} \dot{\alpha} &= \begin{cases} \omega \sqrt{\frac{\gamma}{2}}, & \text{if } |\mathrm{s}| > \alpha_{\mathrm{m}} \\ 0, & \text{if } |\mathrm{s}| \leq \alpha_{\mathrm{m}} \end{cases} && (3.74a) \\ \beta &= 2\varepsilon\alpha + \lambda_2 + 4\varepsilon^2 && (3.74b) \end{aligned}$$

where α_{m} is a design parameter, chosen as a small positive constant. It has been shown that this control-law makes the sliding surface and its derivative converge to zero asymptotically [53]. As mentioned in Section 3.7, since the control-law is designed for the course, it's not possible to guarantee global results.

3.10 Reference Models

In order to guarantee a smooth reference signal to the surge and course controllers, a reference model was implemented. The reference models were of third order and consisted of a low-pass filter and mass-damper-spring system in a cascade. These cascades can be represented as transfer functions on the following forms:

$$\frac{\mathrm{u_d}}{\mathrm{u_r}}(\mathrm{s}) = \frac{\omega_\mathrm{n}^3}{(\mathrm{s}+\omega_\mathrm{n})(\mathrm{s}^2+2\zeta\omega_\mathrm{n}\mathrm{s}+\omega_\mathrm{n}^2)} \tag{3.75a}$$

$$\frac{\chi_\mathrm{d}}{\chi_\mathrm{r}}(\mathrm{s}) = \frac{\omega_\mathrm{n}^3}{(\mathrm{s}+\omega_\mathrm{n})(\mathrm{s}^2+2\zeta\omega_\mathrm{n}\mathrm{s}+\omega_\mathrm{n}^2)} \tag{3.75b}$$

where $\mathrm{u_d}$ and χ_d are the desired surge and course passed on to the controller, $\mathrm{u_r}$ and χ_r are the reference signals from the guidance system, ζ is the relative damping ratio and ω_n is the natural frequency. In order to limit the surge rate $\mathrm{r_d}$ and acceleration $\mathrm{a_d}$ of the desired surge signal, saturation was added to the model. The transfer function in (3.75a) can then be represented on the following state-space form [26, p. 378]

$$\dot{\mathrm{u}}_\mathrm{d} = \mathrm{sat}(\mathrm{r_d}) \tag{3.76a}$$

$$\ddot{\mathrm{u}}_\mathrm{d} = \mathrm{sat}(\mathrm{a_d}) \tag{3.76b}$$

$$\dot{\mathrm{a}}_\mathrm{d} = -(2\zeta+1)\omega_\mathrm{n}\mathrm{sat}(\mathrm{a_d}) - (2\zeta+1)\omega_\mathrm{n}^2\mathrm{sat}(\mathrm{r_d}) + \omega_\mathrm{n}^3(\mathrm{u_r}-\mathrm{u_d}) \tag{3.76c}$$

Similarly, for the transfer function in (3.75b):

$$\dot{\chi}_\mathrm{d} = \mathrm{sat}(\mathrm{r_d}) \tag{3.77a}$$

$$\dot{\mathrm{r}}_\mathrm{d} = \mathrm{sat}(\mathrm{a_d}) \tag{3.77b}$$

$$\dot{\mathrm{a}}_\mathrm{d} = -(2\zeta+1)\omega_\mathrm{n}\mathrm{sat}(\mathrm{a_d}) - (2\zeta+1)\omega_\mathrm{n}^2\mathrm{sat}(\mathrm{r_d}) + \omega_\mathrm{n}^3(\chi_\mathrm{r}-\chi_\mathrm{d}) \tag{3.77c}$$

with the saturation function

$$\mathrm{sat}(\mathrm{x}) = \begin{cases} \mathrm{sign}(\mathrm{x})\mathrm{x_{max}} & \text{if } |\mathrm{x}| \geq \mathrm{x_{max}} \\ \mathrm{x} & \text{otherwise} \end{cases} \tag{3.78}$$

Chapter 4

Implementation

4.1 Overview

A block diagram of the system can be seen in Figure 4.1. The Otter block is the Otter USV model described in Section 3.2, which outputs its states to the LOS guidance law (Section 3.3), and the surge and course controllers (Section 3.6). The output of the LOS block is fed to two reference models (Section 3.10) which outputs the desired value and its derivatives. The controller output is fed to a control-allocation block which translates the input to thruster inputs for the USV, as described in Section 3.2.5.

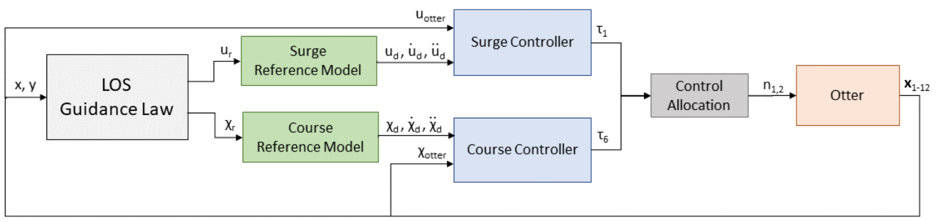


Figure 4.1: Illustrated block diagram of the Matlab simulator

4.2 Measurement noise

In order to make the simulation as realistic as possible, white Gaussian noise was added to the measurements using the Matlab function wgn(m,n,power). This function generates a m $\times$ n matrix of white Gaussian noise samples, where power is associated to the amplitude of the noise [54]. The power-value was tuned for each variable until the amplitude of the generated noises corresponded to the values in Table 4.1.

Measurement	Accuracy
Heading	$< 0.2°$
Roll/Pitch	$0.1°$
Velocity	0.03 m/s
Position	2 cm

Table 4.1: Table of measurement accuracy for the Ellipse RTK GNSS [6]

After the noise was calculated, it was added to the corresponding states. Figure 4.2 shows the simulated noise that affected the surge velocity.

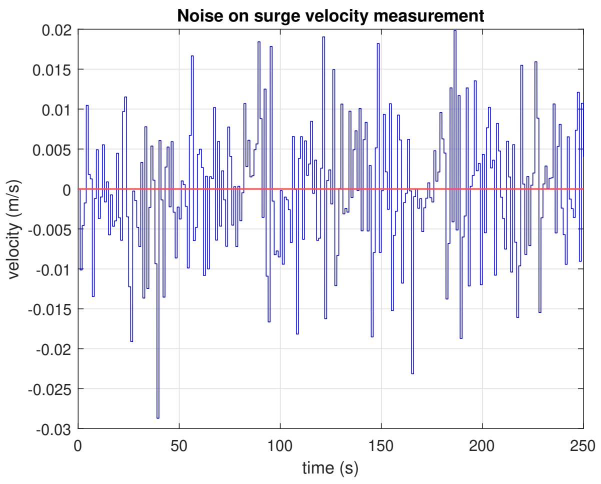


Figure 4.2: Noise affecting the surge velocity, max amplitude at around ±0.03 m/s

The noise was filtered in the simulations as it's assumed that it will be filtered on the real process using a Kalman or low-pass filter. For simplicity, a low-pass filter of the following form was implemented [26, p. 547]

$$\mathrm{y_f} = \frac{\mathrm{h}}{\mathrm{T}} \cdot (\mathrm{y} - \mathrm{y_f}) + \mathrm{y_f} \tag{4.1}$$

where $\mathrm{y_f}$ is the filtered signal, y is the measured signal, h is the sampling time and T is the time constant. T was chosen to be as quick as the noise frequency $(\mathrm{T} = \frac{1}{\text{Noise frequency}} = \frac{1}{1\ \mathrm{Hz}} = 1\ \mathrm{s})$. The unfiltered and the filtered signal of the surge velocity can be seen in Figure 4.3.

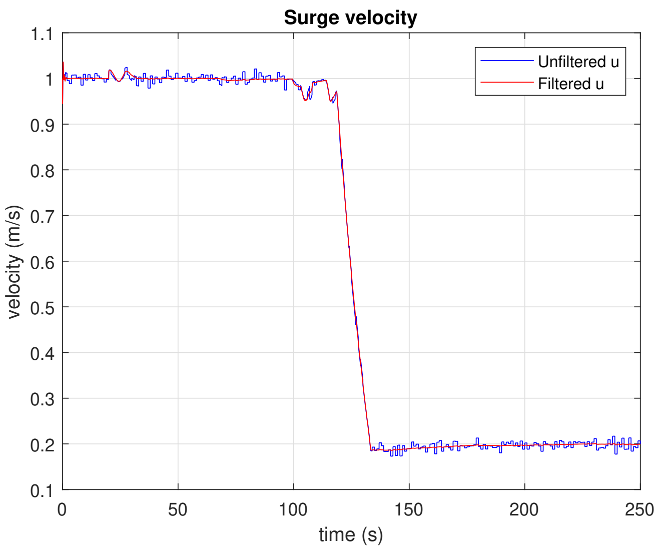


Figure 4.3: Filtered and unfiltered surge velocity signal

4.3 External disturbances

Ocean currents were implemented as external disturbances in the simulator, meaning that the velocity vector in (3.1) had to be replaced by the relative velocities:

$$\boldsymbol{\nu}_{\mathrm{r}} = \boldsymbol{\nu} - \boldsymbol{\nu}_{\mathrm{c}} \tag{4.2}$$

where $\boldsymbol{\nu}_{\mathrm{c}} \in \mathbb{R}^6$ is the velocity of the ocean currents expressed in body frame of the USV. Using this, the equations of motion can be rewritten as

$$\mathbf{M}\dot{\boldsymbol{\nu}}_{\mathrm{r}} + \mathbf{C}(\boldsymbol{\nu}_{\mathrm{r}})\boldsymbol{\nu}_{\mathrm{r}} + \mathbf{D}(\boldsymbol{\nu}_{\mathrm{r}})\boldsymbol{\nu}_{\mathrm{r}} + \mathbf{G}\boldsymbol{\eta} + \mathbf{g}_0 = \boldsymbol{\tau} \tag{4.3}$$

Realistic time varying ocean currents was generated using a first-order Gauss-Markov process described by

$$\dot{\mathrm{V}}_{\mathrm{c}} + \mu \mathrm{V}_{\mathrm{c}} = w \tag{4.4}$$

where w is Gaussian white noise and μ is a non-negative constant [26, p. 281]. The value of the corresponding V_{c} was saturated in the integration process, limiting it to

$$\mathrm{V_{c,min}} \leq \mathrm{V_c(t)} \leq \mathrm{V_{c,max}} \tag{4.5}$$

Furthermore, the current velocity was modeled such that it decreases closer to the shore. The crab angle of the current (illustrated in Figure 4.4) was implemented using the same method, replacing $\mathrm{V_c}$ with β_c.

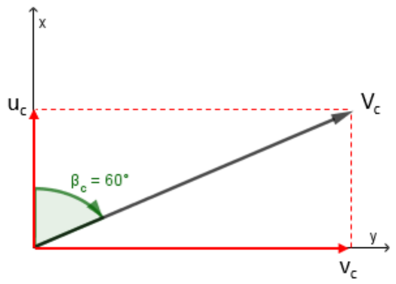


Figure 4.4: Figure showing 60° current crab angle

Both $\mathrm{V_c}$ and β_c was then low-pass filtered, using the filter shown in (4.1), in order to make it slow time varying. Figure 4.5 shows an example of the resulting ocean current velocity and crab angle. From this figure it can also be seen that $\mathrm{V_c}$ starts decreasing as the vessel approaches the shore at around t = 150 s.

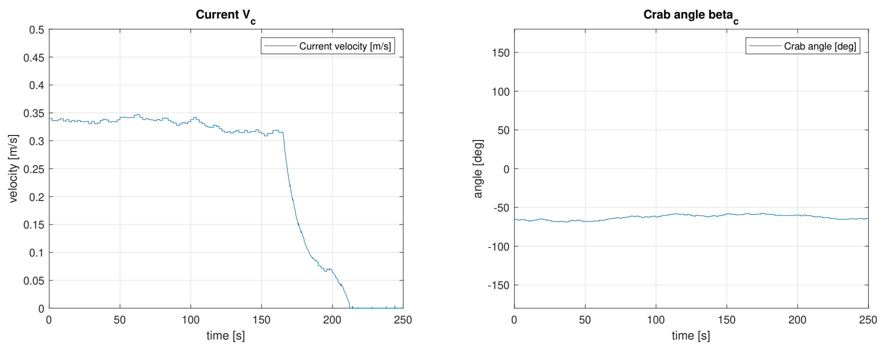


Figure 4.5: Figure showing the generated ocean current velocity and crab angle

4.4 Discretization

As discrete systems are not able to directly calculate derivatives and integrals in continuous time, a different approach is necessary. Derivatives was simply found by using the reference model presented in Section 3.10. In order to integrate the state derivatives in discrete time, the forward Euler's method was used:

$$x_{\mathrm{n+1}} = x_{\mathrm{n}} + \mathrm{h} \cdot \dot{x}_{\mathrm{n}} \tag{4.6}$$

where h is the sampling time, x_{n} is the current value in discrete time and $\dot{x}_{\mathrm{n}}$ is the derivative of x_{n} in continuous time. In the Matlab implementation, all of the Euler integrals were done at the end of the main-loop for each time step to ensure correct state values. The complete code used in the project can be found in Appendix C.

4.5 Controller implementation

The controllers were tuned such that they fulfilled the requirements in Section 1.7. The energy consumption of the controllers was found by calculating the following equation

$$\text{Energy consumption} = \int_0^{\mathrm{t_{end}}} (\tau_1^2 + \tau_6^2)\ \mathrm{dt} \tag{4.7}$$

which gives a rough indicator of the energy consumption, but it should be noted that this value is dimensionless.

4.5.1 PI and PID controllers

The gains for the controllers presented in (3.44) was tuned with the pole-placement algorithm [26, p. 374]. This ensured that the poles were placed in the left-hand plane, making the system locally asymptotically stable. First, the gain K and time constant T for the two linear models (3.42a) and (3.43) were found based on the mass and damping terms. The control bandwidth ω_b can then be determined by the time constant T. The terms for the surge controller was found as follows

$$\mathrm{m}_u = \frac{\mathrm{T}_u}{\mathrm{K}_u} = (\mathrm{m} + \mathrm{m_p} - \mathrm{X}_{\dot{u}}) = 60.5 \tag{4.8a}$$

$$\mathrm{d}_u = \frac{1}{\mathrm{K}_u} = \mathrm{X}_u = 77.55 \tag{4.8b}$$

$$\omega_{\mathrm{b},u} = \frac{1}{\mathrm{T}_u} = 1.28 \tag{4.8c}$$

Equally, the terms for the course controller was found by

$$\mathrm{m}_\chi = \frac{\mathrm{T}_\chi}{\mathrm{K}_\chi} = (\mathrm{I}_z - \mathrm{N}_{\dot{r}}) = 72.24 \tag{4.9a}$$

$$\mathrm{d}_\chi = \frac{1}{\mathrm{K}_\chi} = -\mathrm{N}_r = 90.53 \tag{4.9b}$$

$$\omega_{\mathrm{b},\chi} = \frac{1}{\mathrm{T}_\chi} = 1.25 \tag{4.9c}$$

The P, I and D gains can then be found based on the pole-placement algorithm [26, p. 374]:

$$\omega_\mathrm{n} = \frac{1}{\sqrt{1 - 2\zeta^2 + \sqrt{4\zeta^4 - 4\zeta^2 + 2}}}\omega_\mathrm{b} \tag{4.10a}$$

$$\mathrm{K}_\mathrm{p} = \mathrm{m}\omega_\mathrm{n}^2 \tag{4.10b}$$

$$\mathrm{K}_\mathrm{d} = 2\zeta\omega_\mathrm{n}\mathrm{m} - \mathrm{d} \tag{4.10c}$$

$$\mathrm{K}_\mathrm{i} = \frac{\omega_\mathrm{n}}{10}\mathrm{K}_\mathrm{p} \tag{4.10d}$$

The pole-placement algorithm guarantees that the controller bandwidth is lower than the natural frequency of the system. Critical damping is wanted for both controllers, making the relative damping ratio $\zeta = 1$. The resulting controller gains are summarized in Table 4.2.

Controller	**Parameter**	**Value**
Surge	$\mathrm{K}_{\mathrm{p},u}$	239.30
	$\mathrm{K}_{\mathrm{i},u}$	47.59
Course	$\mathrm{K}_{\mathrm{p},\chi}$	272.5
	$\mathrm{K}_{\mathrm{i},\chi}$	52.93
	$\mathrm{K}_{\mathrm{d},\chi}$	190.08

Table 4.2: The parameters chosen for PI and PID controllers.

4.5.2 PID-SMC

The PID-SMC was tuned by choosing λ to reflect the bandwidth of the controller, ω_b. K_d was chosen such that

$$\frac{\mathrm{K}_\mathrm{d}}{\mathrm{I}_z - \mathrm{N}_{\dot{r}}} \tag{4.11}$$

was close to 1. The parameter ε was chosen as a low value such that there was no chattering present in the controller input. Lastly, the uncertainty was set to be 10% for all of the variables. The resulting parameter values are shown in Table 4.3.

Parameter	Value
λ	1.25
ε	0.1
r_1	1.1
r_2	1.1
K_d	72

Table 4.3: Parameter values for the PID-SMC

4.5.3 STC

The tuning of the STC was done mostly by testing the performance of different parameter values. As a baseline for the tuning, the value of λ_1 was set to be close to the bandwidth of the controller. α_m was then chosen as a small value in order to reduce chattering. To further reduce chattering, the signum function was replaced with the *sigmoid function* shown in (3.55), using the variable ε_sat. γ was chosen as 1 for the entire tuning. ω, λ, ε and ε_sat was then tuned in order to reduce chattering and to minimize the course error. The resulting parameter values are shown in Table 4.4.

Parameter	Value
ω	17
γ	1
ε	0.08
α_m	0.005
λ_1	1.6
λ_2	8
ε_sat	0.005

Table 4.4: Parameter values for the STC

4.6 Reference models

The surge and course reference models, described in Section 3.10, was tuned in Matlab. The relative damping ration ζ_n and natural frequency ω_n, found in (4.10a), was set to match the natural frequency of the systems. The following values was then chosen

Ref. model	Parameter	Value
Course	ζ_{n}	0.9
	ω_{n}	1.94
	$\mathrm{r_{d,max}}$	1 [deg/s]
	$\mathrm{a_{d,max}}$	0.9 [deg/s^2]
Surge	ζ_{n}	1.0
	ω_{n}	1.8
	$\mathrm{r_{d,max}}$	1 [m/s]
	$\mathrm{a_{d,max}}$	0.5 [m/s^2]

Table 4.5: The parameters chosen for course and surge reference models

This resulted in a reference model for course where the USV used 5 seconds to change the desired course from 0 to 90°. The surge reference model also used 5 seconds to change the surge speed from 1 to 2 m/s, this is illustrated in Figure 4.6.

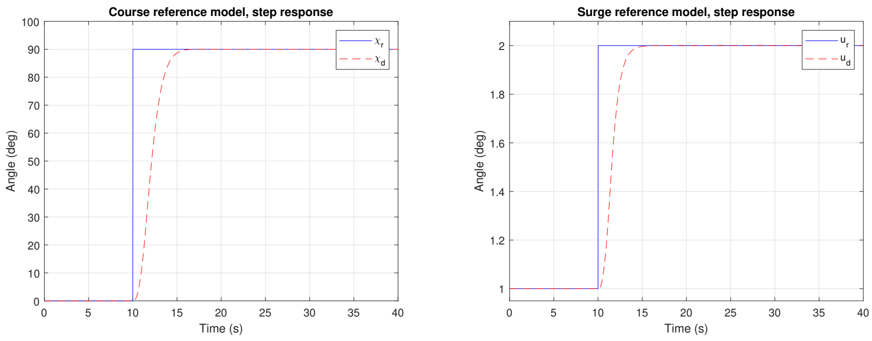


Figure 4.6: Tuning of surge and course reference models

4.7 Obstacle implementation

This section will explain in detail how the obstacle generation and avoidance algorithms were implemented. The generation and avoidance of the obstacles was done such that the requirements in Section 1.7 were fulfilled.

4.7.1 Obstacle avoidance

The obstacle avoidance algorithm used in the project consisted of a hybrid of the guidance law described in Section 3.4 and the A* algorithm described in Section 3.5. The first part of the algorithm consisted of calculating if σ and $\dot{\sigma}$ is outside the tangent cone D defined in (3.40). If this holds true, the USV will enter obstacle avoidance mode. In this mode, a new path will be calculated from the current position of the USV using the A* algorithm. The results from the A* algorithm depends on the value of the heuristic weight. Larger values for the heuristic weight will make the algorithm greedier and more likely to take a longer path, but with less computations [49]. The weight was chosen as

$$\text{HeuristicWeight} = 2 \tag{4.12}$$

The safety distance for the obstacle avoidance was calculated to be 4 times greater than the safe radius R_o.

4.7.2 Obstacle generation

The obstacles were generated in the original path of the USV in order to ensure that it would enter obstacle avoidance mode. A random number of obstacles were generated based on a given maximum number of obstacles (n_{max}) and a given minimum and maximum safe radius R_o. The parameters used in the obstacle generation are shown in Table 4.6.

Parameter	**Value**
n_{max}	3
$R_{o,min}$	2
$R_{o,max}$	3

Table 4.6: Parameter values for the obstacle generation

Chapter 5

Results and discussion

This chapter will present and discuss the results from the simulations of the different controllers. The simulations consisted of a path following phase where u_d was set to 1 m/s and a docking phase where u_d was reduced to 0.2 m/s. The docking phase started when the vessel was less than 25 meters away from the docking area.

The results will first consider two constructed cases: one with weak ocean currents and one with strong ocean currents (relative to the requirement specifications in Section 1.7). Both cases had the same number of obstacles of the same sizes. Furthermore, all of the iterations had the same seed for the random number generator (rng) so that the foundation was equal for all course controllers. These two cases were chosen in order to show how the USV handled both “mild” and “extreme” conditions. The “extreme” condition was chosen as the highest current velocity presented in the requirement specifications.

In addition to these two cases, a Monte-Carlo simulation was conducted in order to test the robustness of the controllers. The simulation study consisted of 10000 iterations where the current velocity, crab angle and obstacles were generated as presented in Section 4.3 and 4.7 respectively. The initial current velocity and crab angle were chosen as a random value within the assumptions stated in 1.6. Similarly to the two cases described above, the seed for the rng was equal for all controllers.

5.1 Case 1: Weak currents

Figure 5.1 shows the current velocity and crab angle generated in the case of weak currents. The initial current velocity was set to be around 0.1 m/s while the initial crab angle was chosen as a random integer between $-180°$ and $180°$. The current velocity drop presented in Section 4.3 can be seen to occur at t $\approx$ 170 s, as the vessel approached the shore.

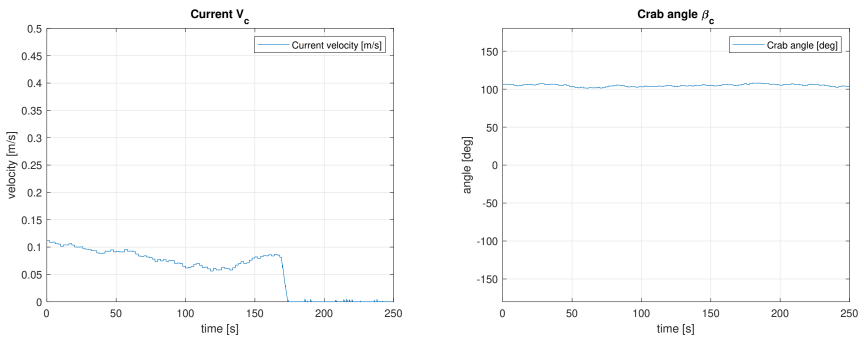


Figure 5.1: The generated current velocity and crab angle for Case 1

Figure 5.2 shows how the PI surge controller tracked the desired surge with weak currents. From this figure it can be seen that there were minuscule tracking errors. This is not unexpected, given that the external disturbances affecting the surge velocity of the vessel were small.

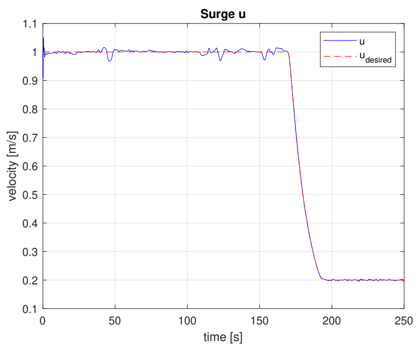


Figure 5.2: Tracking of desired surge with weak currents

5.1.1 PID course controller

As seen in Figure 5.3, the vessel tracked the path well while avoiding the obstacles. It can, however, be seen that the controller had some problems tracking the desired course. The controller input is smooth, and the controller handled the drop in surge (at t $\approx$ 170 s) without any complications.

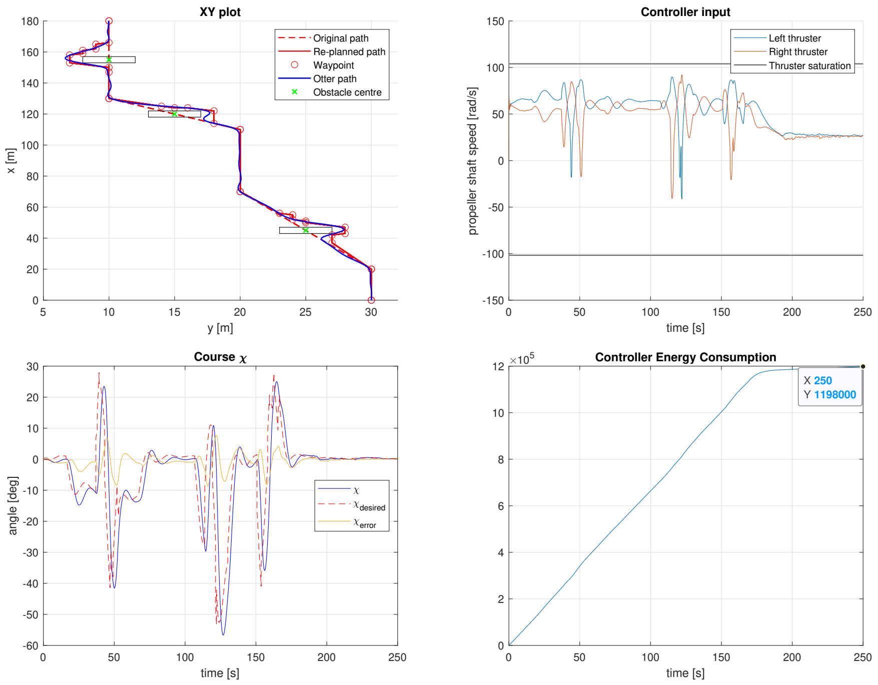


Figure 5.3: Tracking of desired course for the PID controller with corresponding thruster input and energy consumption

5.1.2 PID-SMC

The results in Figure 5.4 show that the vessel tracked the path well, with very small errors in the course. The controller input is smooth up until the end of the sequence, where some very small oscillations can be seen. Furthermore, the results show that the controller had no problems handling the docking phase.

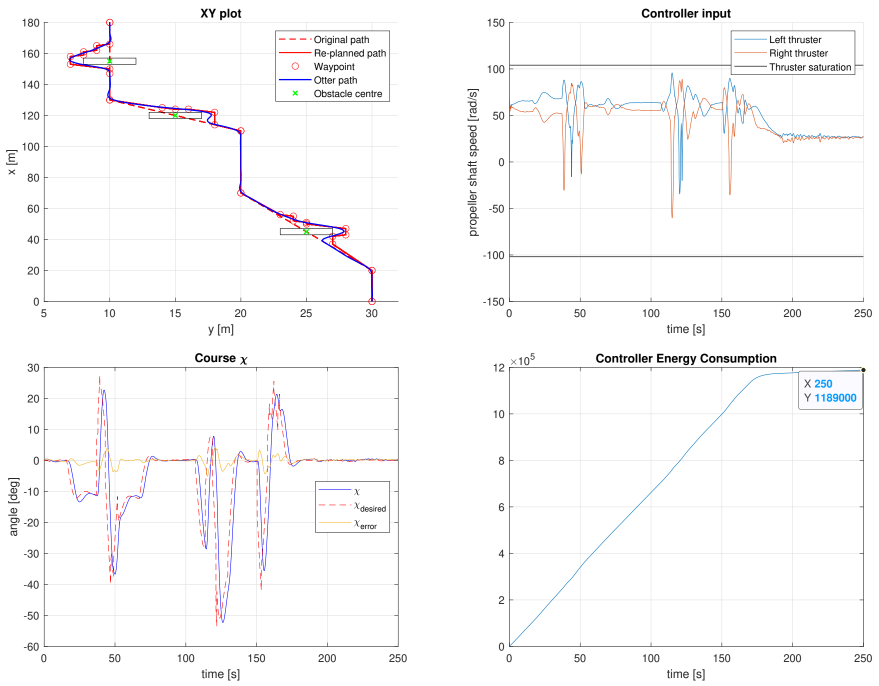


Figure 5.4: Tracking of desired course for the PID-SMC with corresponding thruster input and energy consumption

5.1.3 STC

The results in Figure 5.5 show that the STC performed just as good as the PID-SMC. The STC can also be seen to have smooth controller input up until the end of the sequence, where some small oscillations are visible. These oscillations, while larger than for the PID-SMC, are still quite small.

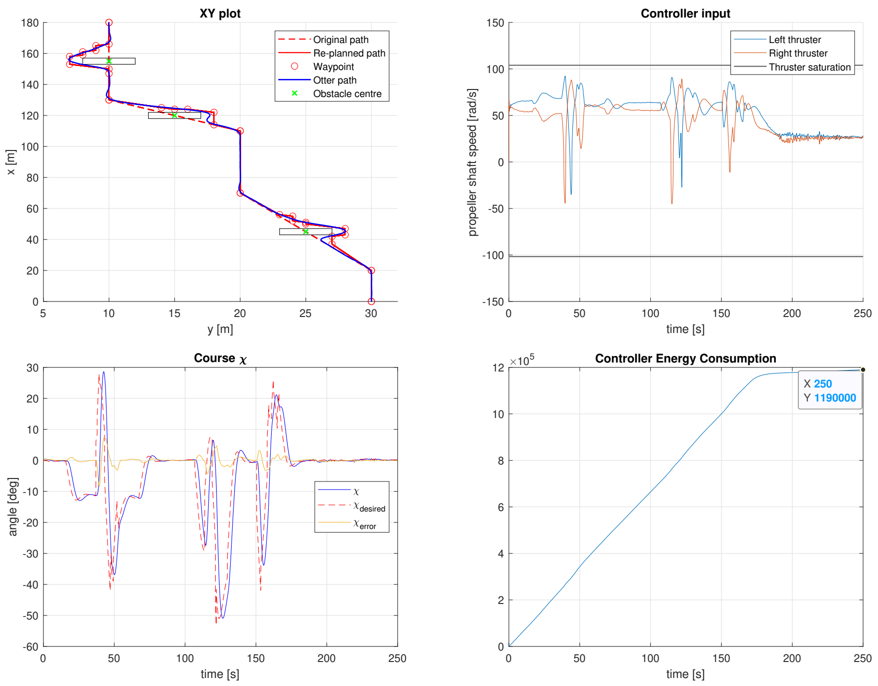


Figure 5.5: Tracking of desired course for the STC with corresponding thruster input and energy consumption

5.1.4 Summary of the results

The resulting median cross-track error (e), course error ($\tilde{\chi}$) and energy consumption for the three controllers can be seen in Table 5.1. The median was used due to the mean cross-track error not being symmetrical (see Section 5.4.1). Table 5.1 shows that the STC had the lowest median course error, but that the PID-SMC had the lowest median cross-track error. Furthermore, it can be seen that the PID-SMC consumed around the same amount of energy as the STC. The PID controller had the worst performance, in addition to consuming the most energy.

Controller	Median e	Median $\tilde{\chi}$	Energy consumption
PID	0.067 m	0.91°	1 198 466
PID-SMC	0.024 m	0.28°	1 188 604
STC	0.025 m	0.23°	1 189 946

Table 5.1: Results from the simulation with weak currents

5.2 Case 2: Strong currents

Figure 5.6 shows the current velocity and crab angle generated in the case of strong currents. The initial current velocity was set to be 0.5 m/s while the initial crab angle was equal to Case 1.

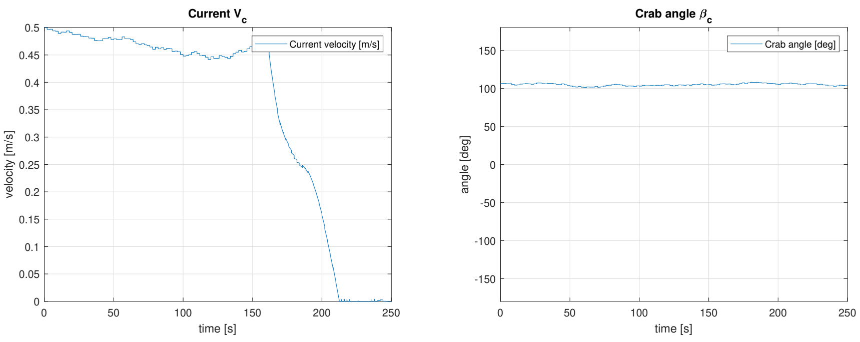


Figure 5.6: The generated current velocity and crab angle for Case 2

When simulating with 0.5 m/s initial current velocity, the surge controller experienced drops and overshoots in surge while the vessel was turning. However, the controller was able to recover within 8 seconds. The controller had no problems tracking the desired surge at the speed drop (at t $\approx$ 160 s). This is due to the fact that the current velocity also starts decreasing at this point, as seen in Figure 5.6. A overshoot of $\tilde{u} \approx 0.1$ m/s can be seen right before the speed drop. This is because of an obstacle located between the last two waypoints.

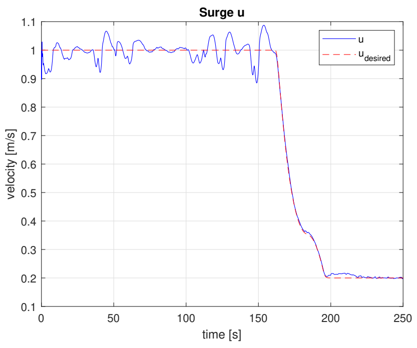


Figure 5.7: Tracking of desired surge with strong currents

5.2.1 PID course controller

The path following seen in the XY-plot in Figure 5.8 shows that the vessel had problems tracking the path, but that it was still able to avoid the obstacles. The controller input can be seen to suffer from oscillations during the whole sequence, except from the docking phase. Furthermore, it can be seen that the controller had problems tracking the desired course. The nominal course error can be seen to be smaller when the vessel entered the docking phase, this is due to the fact that the current velocity decreased at this point, as seen in Figure 5.6.

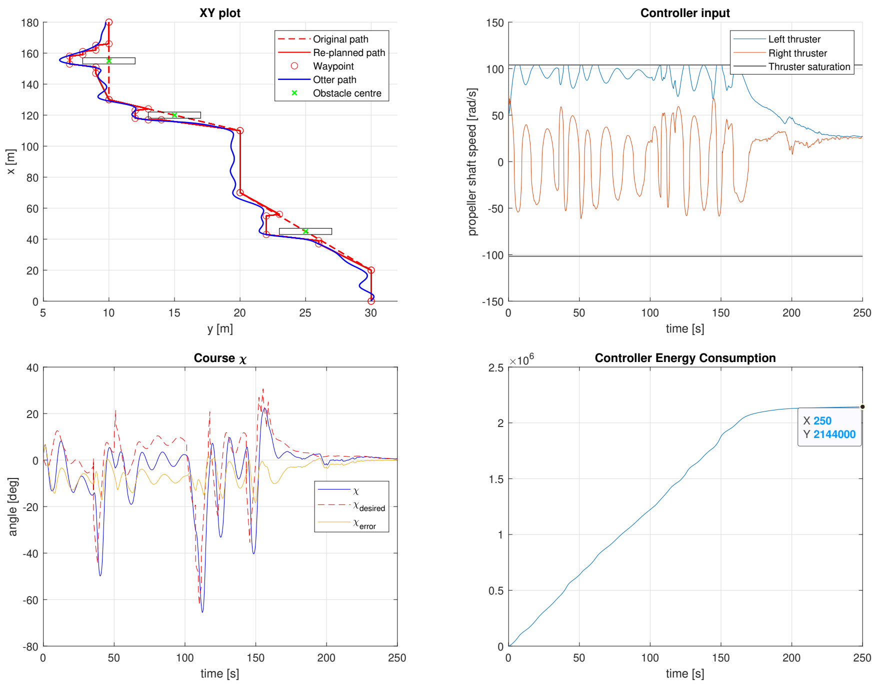


Figure 5.8: Tracking of desired course for the PID controller with corresponding thruster input and energy consumption

5.2.2 PID-SMC

The results in Figure 5.9 show that the vessel had problems tracking the path. The XY-plot shows that the vessel had a small, but noticeable, cross-track error. The controller input can be seen to slowly oscillate during the path following phase. From the figure, it can be seen that the vessel only achieved nominal course error of 0 when entering the docking phase.

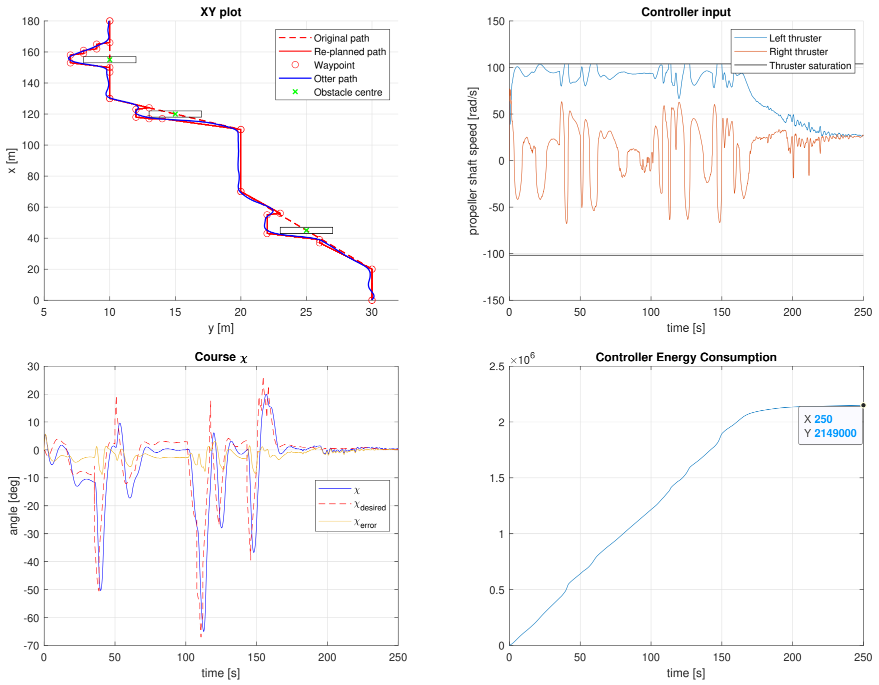


Figure 5.9: Tracking of desired course for the PID-SMC with corresponding thruster input and energy consumption

5.2.3 STC

From Figure 5.10 it can be seen that the vessel tracked the path almost as well as it did in Case 1, with the exception of the beginning of the maneuver and the docking phase. The STC was able to track the desired course with small errors during the whole maneuver. The controller input oscillated less than the two other controllers during the path following phase, but can be seen to experience a small amount of chattering at the docking phase.

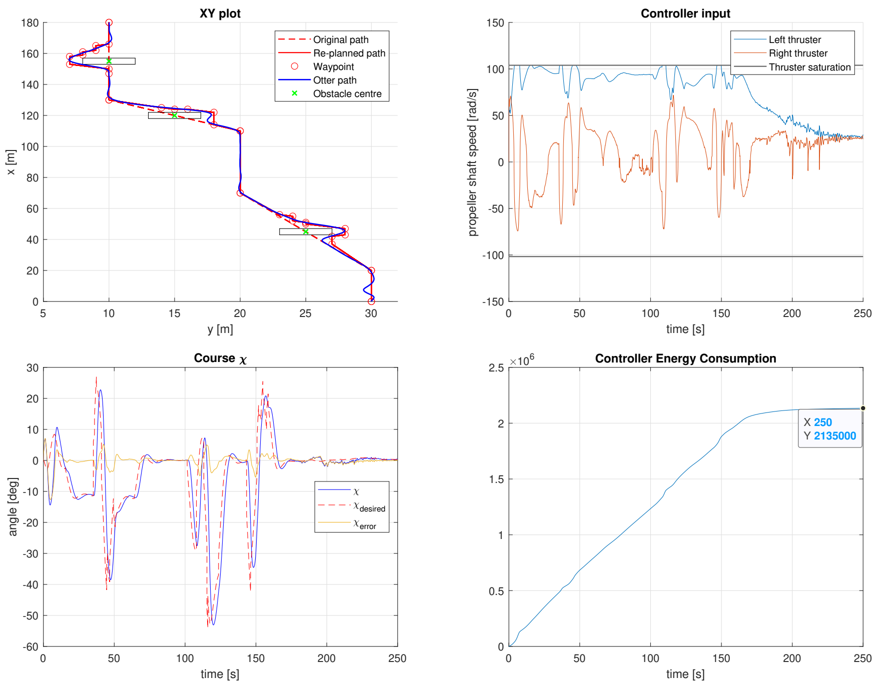


Figure 5.10: Tracking of desired course for the STC with corresponding thruster input and energy consumption

5.2.4 Summary of the results

The results from Case 2 can be seen in Table 5.2. From this table it can be seen that the PID controller still had the worst performance, with a median course error almost 10 times larger than for the STC. Furthermore, it can be seen that the STC had the lowest median course and cross-track error, in addition to having the lowest energy consumption.

Controller	Median e	Median $\tilde{\chi}$	Energy consumption
PID	0.461 m	5.44°	2 144 215
PID-SMC	0.164 m	1.78°	2 149 471
STC	0.047 m	0.49°	2 134 574

Table 5.2: Results from the simulation with strong currents

Both the PID-SMC and STC seemingly suffered from small amounts of chattering at the docking phase. However, from Figure 5.11 it can be seen that the chattering effect is non-existent when zooming in on the controller input at the docking phase. Small adjustments in the controller input looks like chattering in this phase due to the scaling of the figures.

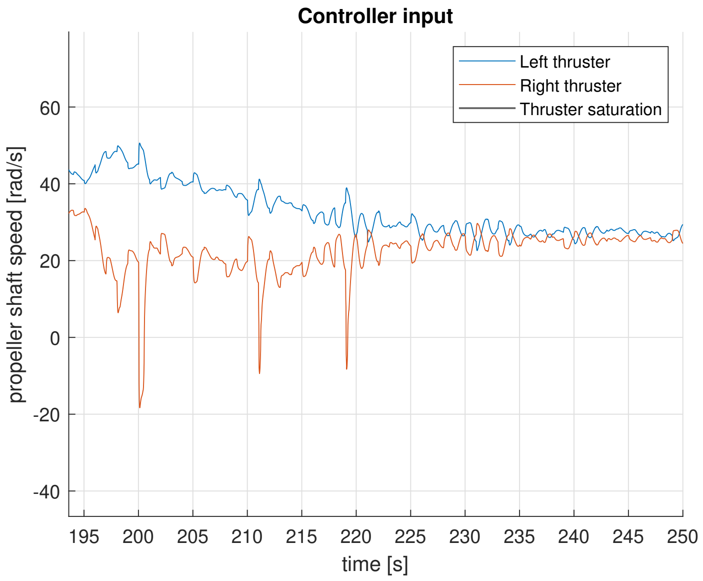


Figure 5.11: Figure showing the controller input for the STC in Case 2 zoomed in

5.3 Monte-Carlo simulation

Median cross-track error, median course error, energy consumption and time used from start to end was tracked for each iteration. The resulting data from the Monte-Carlo simulation is shown in Figure 5.12.

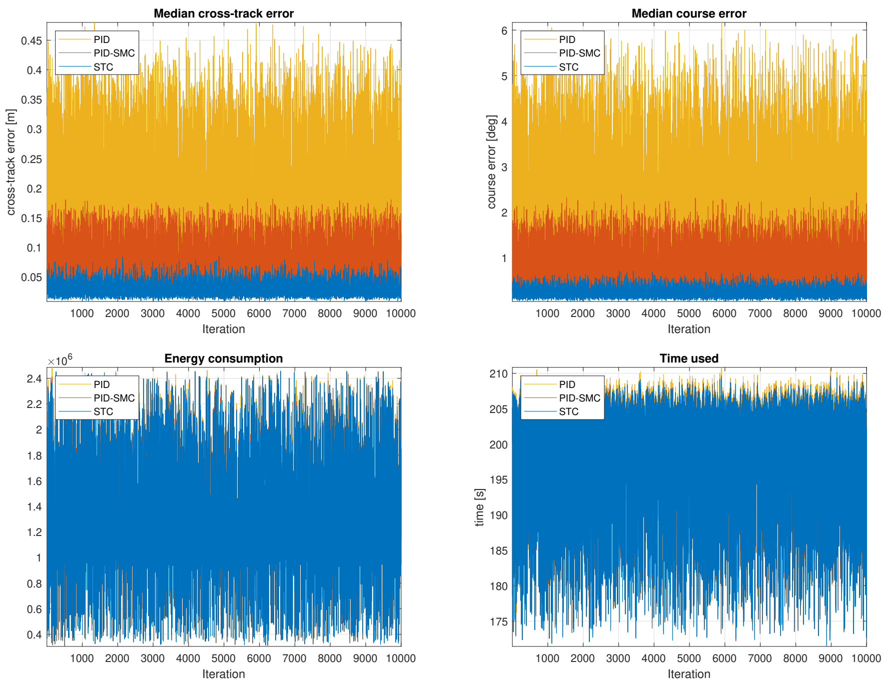


Figure 5.12: Results from Monte-Carlo simulation for all controllers at given iterations

Due to the chaotic nature of the results in Figure 5.12, it can be challenging to visually compare the controllers. The results were therefore interpreted using bar-chart representation as shown in Figure 5.13. From the figures it can be see that all the controllers have peaks at low median cross-track and course errors, but that the STC has the largest peaks. Furthermore, the STC has its peaks at lower values than the other controllers. The PID controller can be seen to have the smallest peaks and the largest errors for both median cross-track and course errors. Lastly, it can be seen that the time and energy consumption is similar for all of the controllers.

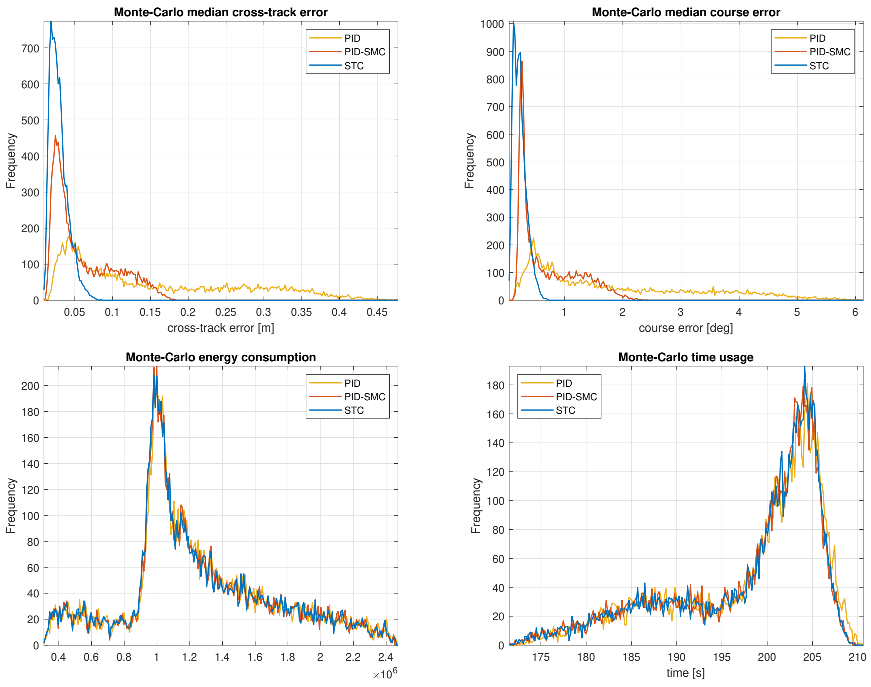
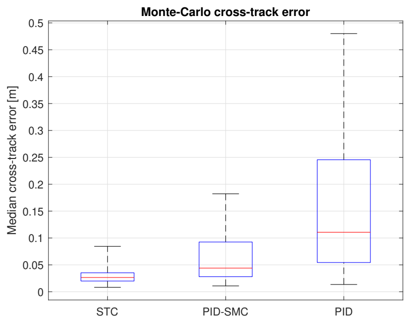


Figure 5.13: Results from Monte-Carlo simulation with bar-chart representation

5.3.1 Summary of the results

The resulting median errors from the Monte-Carlo simulation is represented as box plots in Figure 5.14. Time usage and energy consumption were not taken into account since differences between the controllers were considered trivial. From the figures, it can be seen that the STC had the smallest box size, median and max value for both cross-track and course errors. Furthermore, the median of the PID controller was higher than the max value of the STC for both errors. The total median course and cross-track error for the PID-SMC was 114% and 66% larger than for the STC, respectively. Exact values for the total median cross-track error, total median course error and median energy consumption can be found in Table B.1 in Appendix B.



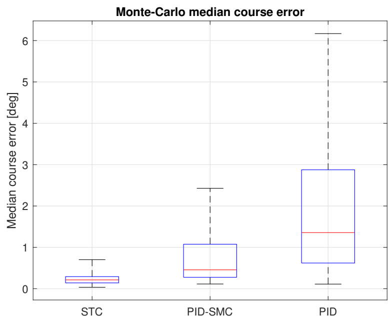


Figure 5.14: Box plots of the median cross-track and course errors from the Monte-Carlo simulation

5.4 Discussion

This section will address choices made throughout the project, possible system weaknesses and sources of error, as this may affect the results.

5.4.1 Median instead of Mean values

When the mean error values were calculated, it was found that the cross-track error did not have a symmetrical, but a multimodal distribution as seen in Figure 5.15. The median does not depend on all of the values in the dataset, unlike the mean. Consequently making the median value less affected by “extreme” values [55]. For this reason, it was chosen to use median instead of mean values in the results.

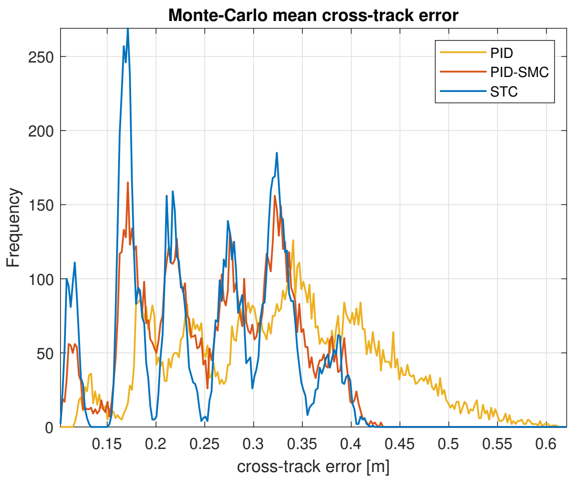


Figure 5.15: Resulting mean cross-track error from the Monte-Carlo simulation

5.4.2 Cross-track error Case 1

The STC had the lowest median course error for all of the cases, but the median cross-track error was 4.2% lower for the PID-SMC in Case 1. The fact that the STC had a lower course error, but larger cross-track error means that the inaccuracy most likely lied in the output from the guidance law for this specific case. It should, however, be noted that the difference of 4.2% in case 1 corresponds to 0.001 meters. Furthermore, the total median cross-track error for the STC was 63% lower than the PID-SMC, which is 15 times larger than the difference observed in Case 1.

5.4.3 Waypoint generation

The vessel was able to avoid obstacles with the A* algorithm generating new waypoints. However, one weakness with the A* algorithm is that it generated many waypoints with a short distance from each other. This can cause problems due to the fact that the vessel iterates to the next waypoint when it's within 4 meters of the current waypoint, causing the vessel to cut corners. This was somewhat avoided by using the mode change radius $\mathrm{R_m}$, such that the path was generated well in advance of the obstacle.

Another weakness of the A* algorithm is that it usually generated paths that consisted of sharp turns. This affected the tracking of desired surge velocity due to the sway that occurs while turning. One way of smoothing the sharp turns could be to use Dubins path when generating the desired path. This is a method that represents the desired path by using circular arcs to connect the waypoints

[56].

5.4.4 Real-world application

One main difference between this simulation study and real life testing is that neither waves nor wind was included as external disturbances. These forces would cause the results from the controllers to deviate from wanted behaviour, and most likely the vessel to drift. Furthermore, if the vessel were to be tested at current velocities above 0.5 m/s (directly towards the Otter USV) would likely lead to the controllers not operating properly. This is, however, mostly due to the fact that the thruster input caps at around that point.

Chapter 6

Conclusions and future work

6.1 Conclusion

Given that the STC tracked the course and cross-track error best out of all of the controllers, it can be concluded that the STC is the best choice out of the three course controllers. While the PID-SMC performed significantly better than the PID controller, it's clear from the box plots in Figure 5.14 that the errors for the STC was even lower. Furthermore, it can be seen from Case 2 that the STC tracks the path considerably better than the other controllers with the presence of strong currents.

Lastly, it can be concluded that all of the three research questions were answered by the project

Q1: The lowest speed at which the USV was still controllable with the presence of slow time-varying currents was found to be 0.2 m/s. The problem with choosing u_d lower than this is the fact that the course and surge controllers are coupled. When the surge controller reduces the surge speed, the course controller loses the ability to quickly recover from tracking errors.

Q2: As shown in the results, all of the three controllers were capable of following the path and to perform the docking procedure with the presence of slow time-varying currents.

Q3: Both of the nonlinear controllers shown in the project have been proven to be robust to parameter uncertainties, which is shown in the theory by stability analysis. Furthermore, the PID-SMC has the possibility to adjust the parameter uncertainty with r_1 and r_2.

Q4: By using a hybrid of Set-Based Guidance and the A* algorithm, the USV was able to avoid all obstacles in real-time while following the path.

6.2 Future work

This part will describe possible future work that could improve the results shown in this report. Different factors that could affect a future implementation will also be mentioned.

6.2.1 Include wind and waves

This project did not include waves or wind as external disturbances, which means that the results are not as realistic as they could be. This could be added to the simulation, but was excluded due to time limitations.

6.2.2 Improve tuning

As is the case for most implementations that includes controllers, the results could be improved with further tuning. The controllers were tuned to have around the same amount of energy consumption, other tuning methods could be used. While testing, it was found that the PID controller could be tuned to give better results, but it was chosen to use the method shown in the report as this is a defined method of tuning a PID controller.

6.2.3 Collision avoidance

As the main focus on the thesis was to test the robustness of the controllers, it was considered sufficient only including static obstacles. However, Set-Based Guidance can be used to avoid moving obstacles in real time while abiding the COLREGS [17].

6.2.4 Include thruster dynamics

The model used in this thesis did not include a delay between controller input and thruster output. This means that as soon as the controller gave input to the thruster, it could instantly reach the given revolution. During the thesis, a first order low-pass filter was tested as a method for including thruster dynamics. However, this lead to highly unstable behaviour where the controller input suffered from large oscillations and chattering. Another method of including thruster dynamics could be to include a saturation of the rate of change for the controller input. This was not tested both due to thruster data not being available for the Otter USV and time constraints.

6.2.5 Real-world application

As this thesis only focused on a model of the Otter USV, it could be beneficial to actually test the controller on it. One of the assumptions made in the thesis was that all of the sensors had the same sampling rate, which is most likely not the case for real-world applications. Another assumption in this thesis was that the course of the Otter USV was measured. If this is not the case for a real-world application, the course controller would make the vessel drift due to the ocean currents. This could somewhat be fixed by implementing an integrated LOS as the guidance law [57].

References

[1] F. Fahimi. “Sliding-Mode Formation Control for Underactuated Surface Vessels.” In: *IEEE Transactions on Robotics* 23.3 (June 2007), pp. 617–622.

[2] Morten Breivik. “Topics in Guided Motion Control of Marine Vehicles.” PhD thesis. June 2010.

[3] Andreas B Martinsen, Anastasios M Lekkas, and Sebastien Gros. “Autonomous docking using direct optimal control.” In: *arXiv preprint arXiv: 1910.11625* (October 2019).

[4] Maritime Robotics. *THE OTTER*. URL: `https://www.maritimerobotics.com/otter` (visited on 08/29/2019).

[5] Geo-matching. *Maritime Robotics Otter USV*. URL: `https://geo-matching.com/usvs-unmanned-surface-vehicles/otter-usv` (visited on 08/27/2019).

[6] SBG Systems. *RMS Inertial Sensors*. 2018. URL: `http://www.cnsens.com/PDF/Ellipse.pdf` (visited on 11/19/2019).

[7] SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. June 2018. URL: `https://doi.org/10.4271/J3016_201806`.

[8] Automotive Electronics. *Society of Automotive Engineers (SAE) Automation Levels for cars*. July 2018. URL: `https://www.automotivelectronics.com/sae-levels-cars/` (visited on 10/17/2019).

[9] Morten Breivik and Jon-Erik Loberg. “A virtual target-based underway docking procedure for unmanned surface vehicles.” In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 13630–13635.

[10] Joohyun Woo, Nakwan Kim, et al. “Vector Field based Guidance Method for Docking of an Unmanned Surface Vehicle.” In: *The Twelfth ISOPE Pacific/Asia Offshore Mechanics Symposium*. International Society of Offshore and Polar Engineers. 2016.

[11] Thor Fossen, Morten Breivik, and Roger Skjetne. “Line-of-Sight Path Following of Underactuated Marine Craft.” In: (September 2003).

[12] Marco Bibuli et al. “Path-Following Algorithms and Experiments for an Unmanned Surface Vehicle.” In: *J. Field Robotics* 26 (August 2009), pp. 669–688.

[13] O. Khatib. “Real-time obstacle avoidance for manipulators and mobile robots.” In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation.* Vol. 2. March 1985, pp. 500–505.

[14] D. Fox, W. Burgard, and S. Thrun. “The dynamic window approach to collision avoidance.” In: *IEEE Robotics Automation Magazine* 4.1 (March 1997), pp. 23–33.

[15] Thanapong Phanthong et al. “Application of A* algorithm for real-time path re-planning of an unmanned surface vehicle avoiding underwater obstacles.” In: *Journal of Marine Science and Application* 13 (February 2014).

[16] Jacoby Larson, Michael Bruch, and John Ebken. “Autonomous navigation and obstacle avoidance for unmanned surface vehicles.” In: *Unmanned Systems Technology VIII.* Ed. by Grant R. Gerhart, Charles M. Shoemaker, and Douglas W. Gage. Vol. 6230. International Society for Optics and Photonics. SPIE, 2006, pp. 53–64. URL: `https://doi.org/10.1117/12.663798`.

[17] S. Moe and K. Y. Pettersen. “Set-based Line-of-Sight (LOS) path following with collision avoidance for underactuated unmanned surface vessel.” In: *2016 24th Mediterranean Conference on Control and Automation (MED).* June 2016, pp. 402–409.

[18] Y. Koren and J. Borenstein. “Potential field methods and their inherent limitations for mobile robot navigation.” In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation.* April 1991, 1398–1404 vol.2.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[20] Y. Li et al. “IBAS: Index Based A-Star.” In: *IEEE Access* 6 (2018), pp. 11707–11715.

[21] Edsger W Dijkstra et al. “A note on two problems in connexion with graphs.” In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[22] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. “A guide to heuristic-based path planning.” In: *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS).* 2005, pp. 9–18.

[23] Y. Wang et al. “Real-time Obstacle Avoidance of Hovercraft Based on Follow the Gap with Dynamic Window Approach.” In: *OCEANS 2018 MTS/IEEE Charleston.* 2018, pp. 1–8.

[24] Javier Fernandez De Canete, Cipriano Galindo, and Inmaculada Garcia-Moral. *System Engineering and Automation: An Interactive Educational Approach.* Springer Science & Business Media, 2011.

[25] Hassan K Khalil. *Nonlinear systems.* eng. Third Edition. Harlow, England: Pearson, 2015.

[26] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control.* eng. Chichester, UK: John Wiley & Sons, Ltd, 2011.

[27] V. Utkin. "Variable structure systems with sliding modes." In: *IEEE Transactions on Automatic Control* 22.2 (April 1977).

[28] Asif Chalanga et al. "Implementation of Super-Twisting Control: Super-Twisting and Higher Order Sliding-Mode Observer-Based Approaches." eng. In: *IEEE Transactions on Industrial Electronics* 63.6 (2016), pp. 3677–3685.

[29] Nguyen Quang Hoang and E Kreuzer. "A robust adaptive sliding mode controller for remotely operated vehicles." In: *Technische Mechanik* 28.3 (2008), pp. 185–193.

[30] Roberto Cristi, Fotis A Papoulias, and Anthony J Healey. "Adaptive sliding mode control of autonomous underwater vehicles in the dive plane." In: *IEEE journal of Oceanic Engineering* 15.3 (1990), pp. 152–160.

[31] D. Yoerger and J. Slotine. "Robust trajectory control of underwater vehicles." In: *IEEE Journal of Oceanic Engineering* 10.4 (October 1985), pp. 462–470.

[32] R. Yu et al. "Sliding mode tracking control of an underactuated surface vessel." In: *IET Control Theory and Applications* 6.3 (2012), pp. 461–466.

[33] H Ashrafiuon et al. "Sliding-Mode Tracking Control of Surface Vessels." eng. In: *IEEE Transactions on Industrial Electronics* 55.11 (2008), pp. 4004–4012.

[34] F. Fahimi. "Sliding-Mode Formation Control for Underactuated Surface Vessels." In: *IEEE Transactions on Robotics* 23.3 (June 2007), pp. 617–622.

[35] Yuri Shtessel et al. *Sliding Mode Control and Observation.* eng. Control Engineering. New York, NY: Springer New York, 2014.

[36] A. Chalanga et al. "Implementation of Super-Twisting Control: Super-Twisting and Higher Order Sliding-Mode Observer-Based Approaches." In: *IEEE Transactions on Industrial Electronics* 63.6 (June 2016), pp. 3677–3685.

[37] Arie Levant. "Sliding order and sliding accuracy in sliding mode control." In: *International Journal of Control* 58.6 (1993), pp. 1247–1263. eprint: `https://doi.org/10.1080/00207179308923053`. URL: `https://doi.org/10.1080/0020717930892305`.

[38] A.I. Propoi. "Application of linear programming methods for the synthesis of automatic sampled-data systems." In: *Avtomat. i Telemekh.* 24.7 (1963), pp. 912–920.

[39] Bjarne Foss and Tor Aksel N Heirung. "Merging optimization and control." In: *Lecture Notes* (2013).

[40] Huarong Zheng, Rudy R Negenborn, and Gabriel Lodewijks. "Trajectory tracking of autonomous vessels using model predictive control." In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 8812–8818.

[41] Eduardo Fernandez-Camacho and Carlos Bordons-Alba. "Introduction to Model Based Predictive Control." In: *Model Predictive Control in the Process Industry*. Springer, 1995, pp. 1–8.

[42] Zhen Li and Jing Sun. "Disturbance compensating model predictive control with application to ship heading control." In: *IEEE transactions on control systems technology* 20.1 (2011), pp. 257–265.

[43] Zhen Li, Jing Sun, and Soryeok Oh. "Path following for marine surface vessels with rudder and roll constraints: An MPC approach." In: *2009 American Control Conference*. IEEE. 2009, pp. 3611–3616.

[44] Carlos Bordons Alba. *Model Predictive Control*. eng. London, 1999.

[45] L.C Mcninch, H Ashrafiuon, and K.R Muske. "Optimal specification of sliding mode control parameters for unmanned surface vessel systems." eng. In: *2009 American Control Conference*. IEEE, 2009, pp. 2350–2355.

[46] SNAME. *Nomenclature for treating the motion of a submerged body through a fluid*. Vol. 1-5. Technical and research bulletin. 1950.

[47] *MSS Toolbox*. URL: `https://github.com/cybergalactic/MSS` (visited on 12/10/2019).

[48] Patrick Lester. *A* Pathfinding for Beginners*. July 18, 2005. URL: `http://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf`.

[49] Anthony Chrabieh. *A star search algorithm*. November 6, 2017. URL: `https://www.mathworks.com/matlabcentral/fileexchange/64978-a-star-search-algorithm`.

[50] S. P. Bhat and D. S. Bernstein. “A topological obstruction to global asymptotic stabilization of rotational motion and the unwinding phenomenon.” In: *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207).* Vol. 5. June 1998, 2785–2789 vol.5.

[51] Yuri B Shtessel et al. “Super-twisting adaptive sliding mode control: A Lyapunov design.” In: *49th IEEE conference on decision and control (CDC).* IEEE. 2010, pp. 5109–5113.

[52] Ida Louise G Borlaug et al. “Trajectory tracking for underwater swimming manipulators using a super twisting algorithm.” In: *Asian Journal of Control* 21.1 (2019), pp. 208–223.

[53] Zhilin Feng and Juntao Fei. “Design and analysis of adaptive Super-Twisting sliding mode control for a microgyroscope.” In: *PLOS ONE* 13.1 (January 2018), pp. 1–18. URL: `https://doi.org/10.1371/journal.pone.0189457`.

[54] MathWorks. *wgn.* URL: `https://se.mathworks.com/help/comm/ref/wgn.html` (visited on 11/22/2019).

[55] Jim Frost. *Measures of Central Tendency: Mean, Median, and Mode.* October 2019. URL: `https://statisticsbyjim.com/basics/measures-central-tendency-mean-median-mode/` (visited on 05/18/2020).

[56] L. E. Dubins. “On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents.” In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516. URL: `http://www.jstor.org/stable/2372560`.

[57] E Borhaug, A Pavlov, and K.Y Pettersen. “Integral LOS control for path following of underactuated marine surface vessels in the presence of constant ocean currents.” eng. In: *2008 47th IEEE Conference on Decision and Control.* IEEE, 2008, pp. 4984–4991.

Appendices

A Physical parameters

Parameter	Description	Value	Unit
m	Mass	55.0	[kg]
m_p	Payload mass	0	[kg]
L	Length	2.0	[m]
B	Beam	1.08	[m]
r_g	Center of gravity for the hull	$[0.2 \ 0 \ -0.2]^\top$	[m]
R_{44}	Radii of gyration	$0.4 \cdot B$	[m]
R_{55}	Radii of gyration	$0.25 \cdot L$	[m]
R_{66}	Radii of gyration	$0.25 \cdot L$	[m]
B_{pont}	Beam of one pontoon	0.25	[m]
Y_{pont}	Distance from centerline to waterline area center	0.395	[m]
$\overline{GM}_T$	Traverse metacentric height	1.9967	[m]
$\overline{GM}_L$	Longitudinal metacentric height	4.7295	[m]
ω_{heave}	Natural frequency in heave	8.28	[rad/s]
ω_{roll}	Natural frequency in roll	7.9241	[rad/s]
ω_{pitch}	Natural frequency in pitch	8.2968	[rad/s]
$C_{w,pont}$	Waterline area coefficient	0.75	[–]
$C_{b,pont}$	Block coefficient	0.4	[–]
k_{pos}	Positive bollard constant	0.0111	[–]
k_{neg}	Negative bollard constant	0.0064	[–]
X_u	Linear damping term	-77.5544	[–]
Y_v	Linear damping term	0	[–]
Z_w	Linear damping term	-546.4805	[–]
K_p	Linear damping term	-54.3823	[–]
M_q	Linear damping term	-246.0496	[–]
N_r	Linear damping term	-90.53	[–]
$Z_{ballast}$	Linear damping due to ballast	0	[–]
$K_{ballast}$	Linear damping due to ballast	-320	[–]
$M_{ballast}$	Linear damping due to ballast	0	[–]

Table A.1: Physical parameters of the Otter

B Monte-Carlo results

	Total median e	Total median $\tilde{\chi}$	Median energy consumption
PID	0.111 m	1.36°	1 270 548
PID-SMC	0.044 m	0.46°	1 260 208
STC	0.027 m	0.21°	1 256 567

Table B.1: Results from the Monte-Carlo simulation

C Matlab Code

Monte-Carlo simulation

```
clearvars;
clear global variables;
clf

N  = 12500;

num_sim = 10000;
num_obstacles=3;

%Mean error values
track_e = zeros(3,num_sim);
course_e = zeros(3,num_sim);
energy = zeros(3,num_sim);
finish_time = zeros(3,num_sim);

%Median error values
median_track_e = zeros(3,num_sim);
median_course_e = zeros(3,num_sim);

%Grand mean error values
m_track_e = zeros(3,1);
m_course_e = zeros(3,1);
m_energy = zeros(3,1);

%% Main loop
for i=1:3
Controller_choise=i-1;

Current_iteration=i

[track_e(i,:),m_track_e(i),course_e(i,:),m_course_e(i),...
    energy(i,:),m_energy(i),finish_time(i,:),...
    median_track_e(i,:), median_course_e(i,:)] =
        RunSimulations(N,num_sim,Controller_choise,
        num_obstacles);

end

%% Plots
t = 1: num_sim;
controller=["STC","PID-SMC","PID"];

figure(1)
plot(t,track_e)
title("Cross track error")
xlabel("Iteration")
```

```
ylabel("Distance [m]")
legend(controller,'Location','northwest')
axis tight
grid

figure(2)
plot(t,(180/pi)*course_e)
title("Course error")
xlabel("Iteration")
ylabel("angle [deg]")
legend(controller,'Location','northwest')
axis tight
grid

figure(3)
plot(t,energy)
title("Energy consumption")
xlabel("Iteration")
legend(controller,'Location','northwest')
axis tight
grid

figure(4)
plot(t,finish_time)
title("Time used")
xlabel("Iteration")
ylabel("time [s]")
legend(controller,'Location','northwest')
axis tight
grid


figure(5)
plot(t,sort(track_e,2))
title("Cross track error sorted")
xlabel("Iteration")
ylabel("Distance [m]")
legend(controller,'Location','northwest')
axis tight
grid

figure(6)
plot(t,sort((180/pi)*course_e,2))
title("Course error sorted")
xlabel("Iteration")
ylabel("angle [deg]")
legend(controller,'Location','northwest')
axis tight
grid

figure(7)
plot(t,sort(energy,2))
```

```
title("Energy consumption sorted")
xlabel("Iteration")
legend(controller,'Location','northwest')
axis tight
grid

figure(8)
plot(t,sort(finish_time,2))
title("Time used sorted")
xlabel("Iteration")
ylabel("time [s]")
legend(controller,'Location','northwest')
axis tight
grid
```

Controller simulations

```
function [mean_e,cross_track,mean_chi_e,course_error,
    energy_consumption, mean_energy,finish_time, median_e,
    median_chi_e] = RunSimulations(N,num_sim,Controller_choise
    ,num_obstacles)

%Select seed for RNG
rng('default')
% rng(3)

mean_e = zeros(num_sim,1);
mean_chi_e = zeros(num_sim,1);

energy_consumption = zeros(num_sim,1);
finish_time = zeros(num_sim,1);

median_e = zeros(num_sim,1);
median_chi_e = zeros(num_sim,1);

for i=1:num_sim
    [buffer,finish_time(i)]= MonteCarlo(N,num_obstacles,
        Controller_choise);
    buffer=buffer';

    mean_e(i) = mean(abs(buffer(21,:)));
    mean_chi_e(i) = mean(abs(buffer(20,:)));
    energy_consumption(i) = buffer(22,end)+buffer(23,end);

    median_e(i) = median(abs(buffer(21,:)));
    median_chi_e(i)=median(abs(buffer(20,:)));
end

cross_track=mean(mean_e);
```

```
course_error=mean(mean_chi_e);
mean_energy=mean(energy_consumption);

end
```

Iteration simulations

```
function [simdata,finish_time ] = MonteCarlo(N,num_obstacles,
    Controller_choise)
%% Init
h   = 0.02;          % sampling time [s]
global Xudot Nrdot Iz Xu m mp alpha v_stw ...
    tau_1_int tau_6_int wpt obstacle_list ;

%  x = [ u v w p q r x y z phi theta psi u_int psi_int]'
x = zeros(14,1);
x_dot=zeros(14,1);

% Surge speed
u_d=1;
u_r=1;

%docking speed
u_r_min=0.2;

%Course
chi_d = 0;
chi_d_ddot  = h*chi_d;

chi_r=0;

%Course of otter
chi_otter=0;
chi_e = chi_d-x(12);

x(1)=u_d; %Set initial velocity to the desired velocity

cross_track_error = 0;
n=[0 0];

n_out=[0 0];
Thrust_calculated = n_out;

% Boat mass
m = 55;

% Load condition
mp = 0;                    % payload mass (kg), max value 45 kg
rp = [0 0 -0.35]';          % location of payload (m)
```

```
rg = [0.2 0 -0.2]';      % CG for hull only (m)

T_yaw = 0.5;                              % time constant in
    yaw (s)
k_pos = 0.02216/2;                        % Positive Bollard,
    one propeller
k_neg = 0.01289/2;                        % Negative Bollard,
    one propeller

% Init (finding Xudot, Nrdot)
init(rg,rp,mp,T_yaw,k_pos,k_neg)

%Gain for PID controller
K = T_yaw/Iz;

% Current
V_c_min=0.;
V_c_max = 0.5;
V_c = randi(100*(V_c_max))/100;

beta_c_min = -180 * pi/180;   % current direction (rad)
beta_c_max = 180 * pi/180;   % current direction (rad)
beta_c = randi(round(2*beta_c_max))+beta_c_min;

%Time constants for time varying currents
T_v=1;
T_beta=1;

ZOH_current=100;

%% WAYPOINTS
wpt.pos.y=[30,30,20,20, 10, 10];
wpt.pos.x=[0,20,70,110, 130, 180];

obstacle_list=[0;0];

wp_LOSindex=1;

R_min=2;
R_max=3;

num_obstacles = randi(num_obstacles);

[RO, obstacle] = obstacle_generator(R_min,R_max,num_obstacles
    );

%Set position for the obstacle center
y_obstacle= obstacle(1,:);
x_obstacle=obstacle(2,:);

Rm=RO*4;
```

```
weight=2;

%Set initial position at the first waypoint
x(7)=wpt.pos.x(wp_LOSindex);
x(8)=wpt.pos.y(wp_LOSindex);

%% LOS
R=4;                                           %Radius for path
   -finding
Delta=4;                                       %Delta for
   lookahead path-finding

add_noise=1;         %0=no noise, 1=add noise
%% Reference model parameters

%Reference model course
r_d=0;
r_dmax=1;
a_d=0;
a_dmax= 0.5;
omega_n=1.3;
damp=0.9;

%Reference model surge
u_r_d=0;
u_r_dmax=1;
u_a_d=0;
u_a_dmax= 0.5;
u_omega_n=1.8;
u_damp=1;

%% Time varying current
alpha_c1=0.005;
alpha_c2=0.01;

%% Course STC constants
omega=17;
gamma=1;
epsilon=0.08;
alpha_m=0.005;

lambda_stc_1= 1.6;
lambda_stc_2= 8;
e_chi_stc=0.005;

%% Course PID SMC contstants
lambda_smc=1.2493;
e_smc=0.1;
k_s=72.241;
```

```matlab
%% Surge PI constants
Kp_u = 239.3;
Ki_u =47.594;

%% Course PID constants
Kp_chi = 272.5044;
Kd_chi = 190.0829;
Ki_chi = 52.9263;

%% MAIN LOOP
simdata = zeros(N,31);                    % table for
    simulation data

for i=1:N+1
    t = (i-1) * h;                              % time (s)

    %Store simulation data in a table
    simdata(i,:) = [t x' n u_d chi_r chi_e cross_track_error
        tau_1_int tau_6_int...
         chi_otter chi_d n_out Thrust_calculated V_c beta_c];

    %% LOS Guidance law

    %Calculation of desired course using LOS

    [~,obstacle_index] = min((sqrt(((x_obstacle-x(7)).^2 + (
        y_obstacle-x(8)).^2))));

    sigma = sqrt((x(8) - y_obstacle(obstacle_index))^2+(x(7)
        - x_obstacle(obstacle_index))^2);
    sigma_dot = ((2*(x(7)-x_obstacle(obstacle_index))*x_dot
        (7)) +(2*(x(8)-y_obstacle(obstacle_index))*x_dot(8)))
        /(2*sigma);

    T_c = in_T_C(sigma,sigma_dot,min(Rm(obstacle_index),max(
        sigma,RO(obstacle_index))),inf);

    %Obstacle avoidance
    if T_c~=true
        if ( ~any(all(obstacle_list ==[x_obstacle(
            obstacle_index);y_obstacle(obstacle_index)])))
            path_replanning(RO(obstacle_index),x_obstacle(
                obstacle_index), y_obstacle(obstacle_index),
                weight,wp_LOSindex,x(7:8));
        end
    end

    [chi_r,cross_track_error,wp_LOSindex]= lookahead_LOS(
        wp_LOSindex,x(7:8),wpt.pos.x,wpt.pos.y,R,Delta);

    %% Reference models
    [u_d_dot, u_r_d_dot, u_a_d_dot] = referencemodel(u_r,
        u_r_d,u_r_dmax,u_a_d,u_a_dmax,u_omega_n,u_damp,u_d);

    [chi_d_dot, r_d_dot, a_d_dot] = referencemodel(chi_r,r_d,
        r_dmax,a_d,a_dmax,omega_n,damp,chi_d);

    %% Calculate error
    chi_e=chi_otter - chi_d;
    chi_e_dot= x(6) - chi_d_dot;

    %% Control-laws
    %PI surge controller
    tau_1 = (m+mp - Xudot)*u_d_dot + Xu*u_d - Kp_u*(x(1)-u_d)
         - Ki_u*x(13);

    %Super-Twist controller
    if Controller_choise==0
        [tau_6,v_stw_dot,alpha_dot]=STC(lambda_stc_2,alpha_m,
            omega,gamma,epsilon,lambda_stc_1,chi_e,chi_e_dot,
            e_chi_stc);
    end

    %PID Sliding Mode Controller
    if Controller_choise==1
        tau_6 =PID_SMC(k_s,lambda_smc,e_smc,chi_e,chi_e_dot,
            chi_d_dot,chi_d_ddot,x(14));
    end

    %PID course controller
    if Controller_choise == 2
        tau_6 = (Iz - Nrdot)*r_d_dot + 1/K*chi_d_dot - Kp_chi
            *(chi_e)  - Kd_chi*(chi_e_dot) - Ki_chi * x(14);
    end

    %% Control allocation
    [n, Thrust_calculated]=calc_thrust(tau_1,tau_6,k_pos,
        k_neg);

    %% Calculate states
    [x_dot(1:12),n_out]= otter(x(1:12),n,mp, rp, V_c,beta_c);

    %Time varying current
    if mod(i,ZOH_current)==0
        V_c_dot= -alpha_c1 * V_c + (rand(1)-0.5)/2;
        beta_c_dot= -alpha_c2 * beta_c + (rand(1)-0.5)/0.5;
    else
        V_c_dot=0;
        beta_c_dot=0;
    end

    %% Euler integration (k+1)
    x(1:12)=x(1:12)+h*x_dot(1:12);

    % Calculate u_e_int x(13)
    x(13) = x(13)+ h*(x(1)-u_d);

    % Calculate saturated chi_e_int x(14)
    x(14) = sat(x(14)+ h*(x(12)-chi_d),0.01);

    % Calculate integral from STC
    if Controller_choise==0
        v_stw=v_stw + h* v_stw_dot;
        alpha = alpha + h*alpha_dot;
    end

    %Calculate energy consumption
    tau_1_int=tau_1_int + h*(tau_1^2);
    tau_6_int=tau_6_int + h*(tau_6^2);

    % Calculate integral from reference models
    chi_d = chi_d + h* chi_d_dot;
    r_d = r_d + h*r_d_dot;
    a_d = a_d + h* a_d_dot;

    u_d=u_d + h* u_d_dot;
    u_r_d=u_r_d + h * u_r_d_dot;
    u_a_d=u_a_d + h * u_a_d_dot;

    %Calculate stochastic time varying current values
    V_c = sat2(V_c + (h/T_v) * V_c_dot,V_c_min,V_c_max);

    beta_c =sat2(beta_c + (h/T_beta)*beta_c_dot,beta_c_min,
        beta_c_max);

    %% Calculate course and add noise
    chi_otter=atan2(x_dot(8),x_dot(7));

    if add_noise==1
        [x(7:8),x(1),x(10:11),chi_otter]= noiseGenerator(i,x
            (7:8),x(1),x(10:11),chi_otter);
    end

    %% Docking phase

    total_distance = sqrt((x(7)-wpt.pos.x(end))^2 + (x(8)-wpt
        .pos.y(end))^2);

    %Reduction of desired surge speed and current
    if total_distance <= 15
        xd=0.0001;
        u_r=sat2(u_r - xd*total_distance,u_r_min,inf);
        V_c = sat2(V_c - (xd*0.3)*total_distance,V_c_min,
            V_c_max);
        alpha_c1=0.2;
    end

end

eta  = simdata(:,8:13);
[~,time_index] = min(sqrt((wpt.pos.y(end)-eta(:,2)).^2 + (wpt
    .pos.x(end)-eta(:,1)).^2));
finish_time = (time_index-1)*h;

end
```

Init

```
function  init(rg,rp,mp,T_yaw,k_pos,k_neg)

global Xudot Nrdot Iz Xu Nr n_max n_min v_stw alpha ...
    tau_1_int tau_6_int pos_noise vel_noise pitchroll_noise
        heading_noise;


m=55;
g = 9.81;

L = 2.0;               % length (m)
B = 1.08;              % beam (m)
rg = (m*rg + mp*rp)/(m+mp);

R44 = 0.4 * B;         % radii of gyration (m)
R55 = 0.25 * L;
R66 = 0.25 * L;

Ig_CG = m * diag([R44^2, R55^2, R66^2]);     % only hull in CG
Ig = Ig_CG - m * Smtrx(rg)^2 - mp * Smtrx(rp)^2;  % hull +
    payload in CO

Iz = 45.126;
Xu = 24.4*(9.81/(6*0.5144));

Nr = Iz/T_yaw;
Xudot = -0.1 * m;
Nrdot = -1.7 * Ig(3,3);

n_max =  sqrt((0.5*24.4 * g)/k_pos);     % maximum propeller
    rev. (rad/s)
n_min = -sqrt((0.5*13.6 * g)/k_neg);     % minimum propeller
    rev. (rad/s)
```

```
v_stw=0;
alpha=0;

tau_1_int=0;
tau_6_int=0;

pos_noise=zeros(2,1);
vel_noise=0;
pitchroll_noise=zeros(2,1);
heading_noise=0;
end
```

Reference model

```
function [chi_d_dot, r_d_dot, a_d_dot] = referencemodel(chi_r
    ,r_d,r_dmax,a_d,a_dmax,omega_n,damp,chi_d)

chi_d_dot = sat(r_d,r_dmax);
r_d_dot   = sat(a_d,a_dmax);
a_d_dot   = -(2*damp + 1)*omega_n * sat(a_d,a_dmax)-(2*damp
    +1)*omega_n^2 * sat(r_d,r_dmax) + omega_n^3 * (chi_r-chi_d
    );

end

function [y] = sat(x,xmax)

if abs(x)>= xmax
  y = sign(x)*xmax;
else
  y = x;
end

end
```

Noise generation

```
function [pos,vel,pitchroll,heading]= noiseGenerator(i,pos,
    vel,pitchroll,heading)

global pos_noise vel_noise pitchroll_noise heading_noise;

h=0.02;
T=1;

%Sample frequency of 1Hz
if mod(i,50)==0
```

```
    %Position
    pos_noise   = wgn(1,2,0.00003,'linear')';

    %Velocity
    vel_noise     = wgn(1,1,0.00006,'linear')';

    %Angle
    pitchroll_noise = deg2rad(wgn(1,2,0.0007,'linear'))' ;
    heading_noise    = deg2rad(wgn(1,1,0.001,'linear')) ;
end

%Position with raw noise
pos_in   = pos + pos_noise;

%Velocity with raw noise
vel_in     = vel + vel_noise;

%Angle with raw noise
pitchroll_in = pitchroll + pitchroll_noise;
heading_in     = heading   + heading_noise;

% Low—pass filtering
pos = (h/T).*(pos_in-pos)+pos;
vel = (h/T).*(vel_in-vel)+vel;
pitchroll = (h/T).*(pitchroll_in-pitchroll)+pitchroll;
heading = (h/T).*(heading_in-heading)+heading;

end
```

Obstacle generation

```
function[RO ,obstacle] = obstacle_generator(R_min,R_max,
    num_obstacles)
global wpt

num_wpt= length(wpt.pos.x);

obstacle=zeros(2,num_obstacles);
RO = zeros(1,num_obstacles);
index= sort(randperm(num_wpt-2,num_obstacles)+1);

for i=1:num_obstacles
    space_y=(wpt.pos.y(index(i)+1)-wpt.pos.y(index(i)))/2;
    space_x=(wpt.pos.x(index(i)+1)-wpt.pos.x(index(i)))/2;

    obstacle_y=randi(sort([wpt.pos.y(index(i))+round(space_y
        /1) wpt.pos.y(index(i)+1)-space_y]));
    obstacle_x=randi(sort([wpt.pos.x(index(i))+round(space_x
        /1) wpt.pos.x(index(i)+1)-space_x]));

    R0(i)=randi([R_min,R_max]);
    obstacle(:,i)= [obstacle_y; obstacle_x];
end

end
```

Control allocation

```
function [n,Thrust] = calc_thrust(tau_1,tau_6,k_pos,k_neg)
global n_min n_max
l1 = -0.395;
l2 = 0.395;

Thrust(2)=(tau_6 + l1*tau_1)/(l1 - l2);
Thrust(1)=tau_1 - Thrust(2);

%Preallocating array for k
k= [0 0];

%Multiplying with correct coefficient depending on rotational
     direction of thrusters
for j=1:2
    if Thrust(j) >0
        k(j)=k_pos;
    else
        k(j)=k_neg;
    end
end

B=[1 1; -l1 -l2]*[k(1) 0; 0 k(2)];
input = B\ [tau_1; tau_6];

n(1)=sat2(sign(input(1))*sqrt(abs(input(1))),n_min,n_max);
n(2)=sat2(sign(input(2))*sqrt(abs(input(2))),n_min,n_max);

end

function [y] = sat2(x,xmin,xmax)

if x< xmin
    y = xmin;
elseif x> xmax
    y= xmax;
else
    y = x;
end
```

```
end
```

STC

```
function [u_st,v_stw_dot, alpha_dot]=STC(lambda2,alpha_m,
    omega,gamma, epsilon,lambda,chi_e,chi_e_dot,e_stc)

global alpha v_stw;

%Sliding surface
s =  lambda*chi_e + chi_e_dot;

%Boundary layer
if abs(s) > alpha_m
    alpha_dot = omega * sqrt (gamma /2);
else
    alpha_dot=0;
end

beta = (2 * epsilon*alpha) + lambda2 + (2*(epsilon^2));

%Saturate s
sign_s=s/(abs(s) + e_stc);

v_stw_dot= -beta * sign_s;

%Control-law
u_st= - alpha * sqrt(abs(s)) * sign_s + v_stw;

end
```

PID-SMC

```
function [u_smc,s,s_dot]=PID_SMC(k_s,lambda,e_smc,chi_e,
    chi_e_dot,chi_d_dot,chi_d_ddot,chi_e_int)

global Nrdot Iz Nr;

T= (Iz - Nrdot)/Nr;
K=1/Nr;

v = chi_d_dot - 2*lambda*chi_e - lambda^2 * chi_e_int;
v_dot = chi_d_ddot - 2*lambda*chi_e_dot - lambda^2 * chi_e;

%Defining sliding surface
s = chi_e_dot + 2*lambda*chi_e + lambda^2 * chi_e_int;
```

```
s_dot = 2* lambda*chi_e_dot+ lambda^2 * chi_e;

rho= (abs((v_dot) *T/K) + 1/K * abs(v));

%10% uncertainty for all measurements
eta=rho*1.1;

%Saturation
sign_s=s/(abs(s) + e_smc);

%Control—law
u_smc = (T/K)*v_dot + v/K - (k_s+Nr*0)*s - eta*sign_s;

end
```

LOS

```
function [chi_desired,e,wp_LOSindex] = lookahead_LOS(
    wp_LOSindex,p, WPx,WPy, R, delta)

wp_dx = WPx(wp_LOSindex+1)-WPx(wp_LOSindex);
wp_dy = WPy(wp_LOSindex+1)-WPy(wp_LOSindex);

alfa_k = atan2(wp_dy, wp_dx);

%Cross—track error
e = -(p(1)-WPx(wp_LOSindex))*sin(alfa_k) + (p(2)-WPy(
    wp_LOSindex))*cos(alfa_k);

chi_r = atan(-e/abs(delta));

x_error = -WPx(wp_LOSindex+1) + p(1);
y_error = -WPy(wp_LOSindex+1) + p(2);

%Waypoint switching
if (x_error^2 + y_error^2 <= R^2) && wp_LOSindex<(length(WPx)
    -1)
     wp_LOSindex = wp_LOSindex + 1;
end

%Desired course
chi_desired = alfa_k + chi_r;

end
```

Obstacle avoidance

```
function [a] = in_T_C(sigma,sigma_dot,sigma_min,sigma_max)

if sigma_min<sigma && sigma<sigma_max
    a=true;
elseif sigma<=sigma_min && sigma_dot>=0
    a=true;
elseif sigma<=sigma_min && sigma_dot<0
    a=false;
elseif sigma>= sigma_max && sigma_dot<=0
    a=true;
else
    a=false;
end

end

function [chi_desired,e,wp_LOSindex] = LOS_avoidance(
   wp_LOSindex,p, WPx,WPy, R, delta,Rm,x_c_obstacle,
   y_c_obstacle,weight)
global wpt obstacle_list orig_wpt;
wp_dx = WPx(wp_LOSindex+1)-WPx(wp_LOSindex);
wp_dy = WPy(wp_LOSindex+1)-WPy(wp_LOSindex);

alfa_k = atan2(wp_dy, wp_dx);

%Cross-track error
e = -(p(1)-WPx(wp_LOSindex))*sin(alfa_k) + (p(2)-WPy(
   wp_LOSindex))*cos(alfa_k);

chi_r = atan(-e/abs(delta));

x_error = -WPx(wp_LOSindex+1) + p(1);
y_error = -WPy(wp_LOSindex+1) + p(2);

x_oa_error=p(1)-x_c_obstacle;
y_oa_error=p(2)-y_c_obstacle;

%Waypoint switching
if (x_error^2 + y_error^2 <= R^2) && wp_LOSindex<(length(wpt.
   pos.x)-1)
     wp_LOSindex = wp_LOSindex + 1;
end

%Generate waypoints when closer to obstacle than the next
   waypoint
if ((x_oa_error^2 + y_oa_error^2 <= x_error^2 + y_error^2) &&
     ~any(all(obstacle_list ==[x_c_obstacle;y_c_obstacle])))
    path_replanning(Rm,x_c_obstacle, y_c_obstacle,weight,
        wp_LOSindex,p);
end
```

```
%Desired course
chi_desired = alfa_k + chi_r;
end
```

Path re-planning

```
function [] = path_replanning(Rm,x_c_obstacle, y_c_obstacle,
    weight,wp_index,p)

global wpt obstacle_list path;

y_obstacle= (y_c_obstacle-Rm):(y_c_obstacle+Rm);
x_obstacle=(x_c_obstacle-Rm):(x_c_obstacle + Rm);

x_start  = round(p(1));
y_start= round(p(2));

x_max=max(wpt.pos.x);
y_max=max(wpt.pos.y);

space_x=round((wpt.pos.x(wp_index+1)-x_c_obstacle)/2);
space_y=round((wpt.pos.y(wp_index+1)-y_c_obstacle)/2);

x_end=wpt.pos.x(wp_index+1)- space_x;
y_end=wpt.pos.y(wp_index+1)-space_y;

obstacle = [x_obstacle;y_obstacle];

path= Astar(x_max,y_max,[x_start y_start],[x_end y_end],
    obstacle,weight);

if ~isempty(path)

    x= path(:,2);
    y=path(:,1);
    dangle=diff(diff(y)./diff(x));

    %Find vertices in the path found
    index = find(abs(dangle)>0.1)+1;

    %Include the start point as a waypoint
    if length(index)>2
        if sqrt((x_start-path(index(1),1))^2 - (y_start-path(
            index(1),2))^2)>0
             index = [1; index];
        end
    end

    %Update waypoints and obstacle list
    if ~isempty(index)
        if ~all([wpt.pos.x;wpt.pos.y]==[path(index(1),1)';
            path(index(1),2)'])
            wpt.pos.y=[wpt.pos.y(1:wp_index) path(index,2)'
                wpt.pos.y((wp_index+1):end)];
            wpt.pos.x=[wpt.pos.x(1:wp_index) path(index,1)'
                wpt.pos.x((wp_index+1):end)];

            obstacle_list = [obstacle_list, [x_c_obstacle;
                y_c_obstacle]];
        end
    end
end
end
```

A* algorithm

```
function [path]= Astar(xmax,ymax,start,goal,obstacle,w)

% Author:     Anthony Chrabieh
% Date:       2017-11-06
% Revisions: 2020-03-13

MAP = zeros(xmax,ymax);

%Place obstacles in the map
for i=1:2:size(obstacle,1)
    MAP(obstacle(i,:),obstacle(i+1,:))=inf;
end

%Heuristic Weight
weight = sqrt(w);

%Heuristic Map of all nodes
for x = 1:size(MAP,1)
    for y = 1:size(MAP,2)
        if(MAP(x,y)~=inf)
            H(x,y) = weight*norm(goal-[x,y]);
            G(x,y) = inf;
        end
    end
end

%% initial conditions
G(start(1),start(2)) = 0;
F(start(1),start(2)) = H(start(1),start(2));

closedNodes = [];
```

```
openNodes = [start G(start(1),start(2)) F(start(1),start(2))
    0]; %[x y G F cameFrom]

%% Solve
solved = false;

while(~isempty(openNodes))

    %Find node from open set with smallest F value
    [~,I] = min(openNodes(:,4));

    %Set current node
    current = openNodes(I,:);

    %If goal is reached, break the loop
    if(current(1:2)==goal)
        closedNodes = [closedNodes;current];
        solved = true;
        break;
    end

    %remove current node from open set and add it to closed
        set
    openNodes(I,:) = [];
    closedNodes = [closedNodes;current];

    %For all neighbors of current node
    for x = current(1)-1:current(1)+1
        for y = current(2)-1:current(2)+1

            %If out of range skip
            if (x<1||x>xmax||y<1||y>ymax)
                continue
            end

            %If object skip
            if (isinf(MAP(x,y)))
                continue
            end

            %If current node skip
            if (x==current(1)&&y==current(2))
                continue
            end

            %If already in closed set skip
            skip = 0;
            for j = 1:size(closedNodes,1)
                if(x == closedNodes(j,1) && y==closedNodes(j
                    ,2))
                    skip = 1;
                    break;
                end
            end
            if(skip == 1)
                continue
            end

            A = [];
            %Check if already in open set
            if(~isempty(openNodes))
                for j = 1:size(openNodes,1)
                    if(x == openNodes(j,1) && y==openNodes(j
                        ,2))
                        A = j;
                        break;
                    end
                end
            end

            newG = G(current(1),current(2)) + round(norm([
                current(1)-x,current(2)-y]),1);

            %If not in open set, add to open set
            if(isempty(A))
                G(x,y) = newG;
                newF = G(x,y) + H(x,y);

                newNode = [x y G(x,y) newF size(closedNodes
                    ,1)];
                openNodes = [openNodes; newNode];
                continue
            end

            %If no better path, skip
            if (newG >= G(x,y))
                continue
            end

            G(x,y) = newG;

            newF = newG + H(x,y);
            openNodes(A,3:5) = [newG newF size(closedNodes,1)
                ];
        end
    end
end

if (solved)

    j = size(closedNodes,1);
    path = [];
    while(j > 0)
        x = closedNodes(j,1);
        y = closedNodes(j,2);
        j = closedNodes(j,5);
        path = [x,y;path];
    end
else
    path= [];
end
end
```

Otter model

```
function [xdot,output] = otter(x,n,mp,rp,V_c,beta_c)
% [xdot,U] = otter(x,n,mp,rp,V_c,beta_c) returns the speed U
    in m/s (optionally)
% and the time derivative of the state vector:
%    x = [ u v w p q r x y z phi theta psi ]'
% for the Maritime Robotics Otter USV, see www.
    maritimerobotics.com.
% The length of the USV is L = 2.0 m, while the state vector
    is defined as:
%
% u     = surge velocity          (m/s)
% v     = sway velocity           (m/s)
% w     = heave velocity          (m/s)
% p     = roll velocity           (rad/s)
% q     = pitch velocity          (rad/s)
% r     = yaw velocity            (rad/s)
% x     = position in x direction (m)
% y     = position in y direction (m)
% z     = position in z direction (m)
% phi   = roll angle              (rad)
% theta = pitch angle             (rad)
% psi   = yaw angle               (rad)
%
% The other inputs are:
%
% n = [ n(1) n(2) ]'  where
%    n(1) = propeller shaft speed, left (rad/s)
%    n(2) = propeller shaft speed, right (rad/s)
%
% mp = payload mass (kg), maximum 45 kg
% rp = [xp, yp, zp]' (m) is the location of the payload
% V_c = current speed (m/s)
% beta_c = current direction (rad)
%
% Author:    Thor I. Fossen
% Date:      2019-07-17
% Revisions: 2019-10-14

% Check of input and state dimensions
if (length(x) ~= 12),error('x vector must have dimension 12 !
    '); end
if (length(n) ~= 2),error('n vector must have dimension 2 !')
    ; end

% Main data
g   = 9.81;          % acceleration of gravity (m/s^2)
rho = 1025;          % density of water
L = 2.0;             % length (m)
B = 1.08;            % beam (m)
m = 55.0;            % mass (kg)
rg = [0.2 0 -0.2]'; % CG for hull only (m)
R44 = 0.4 * B;       % radii of gyration (m)
R55 = 0.25 * L;
R66 = 0.25 * L;
T_yaw = 0.5;         % time constant in yaw (s)
Umax = 6 * 0.5144;   % max forward speed (m/s)

% Data for one pontoon
B_pont  = 0.25;      % beam of one pontoon (m)
y_pont  = 0.395;     % distance from centerline to waterline
    area center (m)
Cw_pont = 0.75;      % waterline area coefficient (-)
Cb_pont = 0.4;       % block coefficient, computed from m = 55
     kg



% State and current variables
nu = x(1:6);  nu1 = x(1:3); nu2 = x(4:6);    % velocities
eta = x(7:12);                               % positions
U = sqrt(nu(1)^2 + nu(2)^2);                 % speed
u_c = V_c * cos(beta_c - eta(6));            % current surge
    velocity
v_c = V_c * sin(beta_c - eta(6));            % current sway
    velocity

% Inertia dyadic, volume displacement and draft
nabla = (m+mp)/rho;                          % volume
T = nabla / (2 * Cb_pont * B_pont*L);        % draft
Ig_CG = m * diag([R44^2, R55^2, R66^2]);     % only hull in CG
rg = (m*rg + mp*rp)/(m+mp);            % CG location corrected
     for payload
Ig = Ig_CG - m * Smtrx(rg)^2 - mp * Smtrx(rp)^2;  % hull +
    payload in CO

% Experimental propeller data including lever arms
```

```
l1 = -y_pont;                              % lever arm, left
    propeller (m)
l2 = y_pont;                               % lever arm, right
    propeller (m)
k_pos = 0.02216/2;                         % Positive Bollard,
    one propeller
k_neg = 0.01289/2;                         % Negative Bollard,
    one propeller
n_max =  sqrt((0.5*24.4 * g)/k_pos);     % maximum propeller
    rev. (rad/s)
n_min = -sqrt((0.5*13.6 * g)/k_neg);     % minimum propeller
    rev. (rad/s)

% MRB and CRB (Fossen 2011)
I3 = eye(3);
O3 = zeros(3,3);

MRB_CG = [ (m+mp) * I3   O3
    O3       Ig ];
CRB_CG = [ (m+mp) * Smtrx(nu2)    O3
    O3                 -Smtrx(Ig*nu2)  ];

H = Hmtrx(rg);               % Transform MRB and CRB from CG
    to CO
MRB = H' * MRB_CG * H;
CRB = H' * CRB_CG * H;

% Hydrodynamic added mass (best practise)
Xudot = -0.1 * m;
Yvdot = -1.5 * m;
Zwdot = -1.0 * m;
Kpdot = -0.2 * Ig(1,1);
Mqdot = -0.8 * Ig(2,2);
Nrdot = -1.7 * Ig(3,3);

MA = -diag([Xudot, Yvdot, Zwdot, Kpdot, Mqdot, Nrdot]);
CA  = m2c(MA, nu);

% System mass and Coriolis-centripetal matrices
M = MRB + MA;
C = CRB + CA;

% Hydrostatic quantities (Fossen 2011)
Aw_pont = Cw_pont * L * B_pont;     % waterline area, one
    pontoon
I_T = 2 * (1/12)*L*B_pont^3 * (6*Cw_pont^3/((1+Cw_pont)*(1+2*
    Cw_pont)))...
    + 2 * Aw_pont * y_pont^2;
I_L = 0.8 * 2 * (1/12) * B_pont * L^3;
KB = (1/3)*(5*T/2 - 0.5*nabla/(L*B_pont) );
BM_T = I_T/nabla;        % BN values
```

```
BM_L = I_L/nabla;
KM_T = KB + BM_T;        % KM values
KM_L = KB + BM_L;
KG = T - rg(3);
GM_T = KM_T - KG;        % GM values
GM_L = KM_L - KG;

G33 = rho * g * (2 * Aw_pont);      % spring stiffness
G44 = rho * g *nabla * GM_T;
G55 = rho * g *nabla * GM_L;

G_CF = diag([0 0 G33 G44 G55 0]);   % spring stiffness matrix
     in CF
LCF = -0.2;
H = Hmtrx([LCF 0 0]);               % transform G_CF from CF
    to CO
G = H' * G_CF * H;

% Natural frequencies
w3 = sqrt( G33/M(3,3) ) ;
w4 = sqrt( G44/M(4,4) );
w5 = sqrt( G55/M(5,5) );

% Linear damping terms (hydrodynamic derivaties)
Xu = -24.4 * g / Umax;              % specified using max
    speed
Yv = 0;
Zw = -2 * 0.3 *w3 * M(3,3);         % specified using rel
    damp factors
Kp = -2 * 0.2 *w4 * M(4,4);
Mq = -2 * 0.4 *w5 * M(5,5);
Nr = -M(6,6)/T_yaw;                 % specified using time
    const T_yaw

% Control forces and moments — with propeller revolution
    saturation
Thrust = zeros(2,1);
for i = 1:1:2
    if n(i) > n_max                     % saturation,
        physical limits
        n(i) = n_max;
    elseif n(i) < n_min
        n(i) = n_min;
    end

    if n(i) > 0
        Thrust(i) = k_pos * n(i)*abs(n(i));    % forward
            force (N)
    else
        Thrust(i) = k_neg * n(i)*abs(n(i));    % aft force (N
            )
    end
end

output= [Thrust(1),Thrust(2)];
% Control forces and moments
tau = [Thrust(1) + Thrust(2) 0 0 0 0  -l1 * Thrust(1) - l2 *
    Thrust(2) ]';

% Linear damping using relative velcoities
Xh = Xu * (nu(1) - u_c);
Yh = Yv * (nu(2) - v_c);
Zh = Zw * nu(3);
Kh = Kp * nu(4);
Mh = Mq * nu(5);
Nh = Nr * nu(6);

% Strip theory: cross-flow drag integrals for Yh and Nh
dx = L/10;                      % 10 strips
Cd_2D = Hoerner(B_pont,T);  % 2D drag coefficeint for one
    pontoon
for xL = -L/2:dx:L/2
    v_r = nu(2) - v_c;          % relative sway velocity
    r = nu(6);                  % yaw rate
    Ucf = abs(v_r + xL * r) * (v_r + xL * r);
    Yh = Yh - 0.5 * rho * T * Cd_2D * Ucf * dx;         %
        sway force
    Nh = Nh - 0.5 * rho * T * Cd_2D * xL * Ucf * dx;    % yaw
         moment
end

% kinematics
[Rzyx, Tzyx] = body2ned(eta);

nu_r=nu-[u_c v_c 0 0 0 0]';
% trim: theta = -7.5 deg correponds to 13.5 cm less height
    aft maximum load
g_0 = [ 0 0 0 0 320 0]';

% time derivative of states - numerical integration; see
    ExOtter.m
xdot = [...
    M \ ( tau + [Xh Yh Zh Kh Mh Nh]' - C * nu_r - G * eta -
        g_0)
    Rzyx * nu1
    Tzyx * nu2 ];
end

%% Function [Rzyx, Tzyx] = body2ned(eta)
function [Rzyx, Tzyx] = body2ned(eta)
% computes the rotation and angular velocity transformation
    matrices
```

```
% between BODY and NED
cphi = cos(eta(4));
sphi = sin(eta(4));
cth  = cos(eta(5));
sth  = sin(eta(5));
cpsi = cos(eta(6));
spsi = sin(eta(6));

Rzyx = [...
    cpsi*cth  -spsi*cphi+cpsi*sth*sphi  spsi*sphi+cpsi*cphi*
        sth
    spsi*cth  cpsi*cphi+sphi*sth*spsi   -cpsi*sphi+sth*spsi*
        cphi
    -sth      cth*sphi                  cth*cphi ];

Tzyx = [...
    1  sphi*sth/cth  cphi*sth/cth;
    0  cphi          -sphi;
    0  sphi/cth      cphi/cth ];
end

%% Function S = Smtrx(a
function S = Smtrx(a)
% S = Smtrx(a) computes the 3x3 vector skew-symmetric matrix
   S(a) = -S(a)'.
% The corss product satisfies: a x b = S(a)b.
S = [  0  -a(3)    a(2)
    a(3)     0   -a(1)
    -a(2)   a(1)     0 ];
end

%% Function H = Hmtrx(r)
function H = Hmtrx(r)
% H = Hmtrx(r) computes the 6x6 system transformation matrix
%
% H = [eye(3)     Smtrx(r)'
%      zeros(3,3)    eye(3) ];       Property: inv(H(r)) = H(-
   r)
%
% If r = r_g is the vector from CO to CG, the model matrices
   in CO and CG
% are related by: M_CO = H(r_g)' * M_CG * H(r_g). Generalized
    position and
% force satisfy: eta_CO = H(r_g)' * eta_CG and tau_CO = H(r_g
   )' * tau_CG
H = [eye(3)      Smtrx(r)'
    zeros(3,3) eye(3) ];
end
```