NTNU

Norwegian University of
Science and Technology

Remote Control and Path Following for the ReVolt Model Ship

Albert Havnegjerde

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Morten Breivik, ITK
Co-supervisor: Tom Arne Pedersen, DNV-GL

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This thesis concludes my 2-year master's degree specializing in Navigation, Vessel Control Systems and Robot Engineering at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology.

This thesis has not only given me valuable insight into software development, identification, control and implementation, but also executing a project on my own. Though the project was conducted individually, it would not be possible without the help from certain individuals. I would like to thank my supervisor Morten Breivik, Head of Department of Engineering Cybernetics, and co-supervisor Tom Arne Pedersen, Principal researcher at DNV GL, for their professional guidance and suggestions throughout the project period. An extra thanks to Tom Arne Pedersen for assisting with the simulator, arranging and piloting the follow boat, as well as obtaining necessary equipment regarding experimental sea tests. I would like to thank my friend and fellow student Vegard Kamsvåg for assistance during experimental tests, I would like to thank Stefano Bertelli for arranging transportation of ReVolt to and from the test area, and I would like to thank Glenn Angell at the mechanical workshop for help with debugging and repairing ReVolt's hardware. I would also like to thank Roger Nilsen for help in installing and showing me how to use the simulator for ReVolt. Finally, I would like to thank my family Marianne, Tom, Sandra and Andrea for all their support and encouragement.

The tasks in the specialization project fall of 2017 and the following master's thesis during the spring of 2018 is proposed by DNV GL to support the continuous development of their experimental platform ReVolt Model Ship, hereby referred to as ReVolt. While the specialization project regards exclusively development of a Remote Monitoring and Control (RMC) station for ReVolt, this master's thesis focus mostly on development, implementation and testing of motion control algorithms for path following for ReVolt. In addition, some improvements and new functionality for the RMC station is also presented.

Available information and equipment directly relating to the execution of this master thesis work:

- **Previous Work:** Master's thesis *"Development of a Dynamic Positioning System for the ReVolt Model Ship"* [1] and *"ReVolt User Manual"*.
- **ReVolt:** The 3 meter long physical scale model of the concept ship ReVolt is provided by DNV GL. It has sensors for navigation, three rotatable thrusters and an onboard computer (OBC) installed. New features has been added to ReVolt's control system during the master thesis work and the physical model is described more in Section 6.1.
- **Control System:** Includes DNV GL's thruster allocation, a Dynamic Positioning (DP) system and a simple heading controller. The control system is launched using

the Robot Operating System (ROS) described in Sections 3.1.5-3.1.9.

- **Digital Twin:** In parallel with this project, a simulated model of ReVolt that uses the same control system, was configured and provided by Tom Arne Pedersen with assistance from Roger Nilsen (DNV GL). This software simplified testing of new features developed during this master thesis work, and made it possible to test the control system before doing experiments with the physical model. The simulator is described more in Section 5.1.

For the list of contributions in this thesis, refer to Section 1.4.

Albert Havnegjerde
Trondheim, June 2018

Abstract

The guidance system of the 3 meter long ReVolt is expanded to include a 2-D path following system using the *Line-of-Sight* (LOS) guidance principle and *lookahead-based* steering algorithm for computing the heading reference signal. Reference models for the heading and speed motion is added to smooth the reference signals and create higher order derivatives for the controllers. A separate software for visually placing waypoints in a navigation map and transmitting them ReVolt is developed in C++ using *Qt Creator*. This software is used for remote monitoring and control of ReVolt during simulations and experimental tests. An improved heading controller for maintaining the correct heading during course-keeping and course-changing maneuvers is implemented. A 1st order *Nomoto model* is used to compute the parameters in the model-based feedforward term and the feedback gains. A speed controller consisting of a feedforward and feedback term is implemented as well.

In transit, the two aft azimuth thrusters are constrained to $\pm 45^o$ by the control allocation and the bow thruster remains retracted. This leaves the sway motion uncontrolled, resulting in an underactuated configuration. The source code for controllers and LOS steering algorithm are written in C++ and launched using ROS from the ReVolt onboard computer.

System identification and testing of the guidance and control system are first done using ReVolt's Digital Twin in the simulator developed by DNV GL. Full-scale experimental tests to assess the performance of the implemented solutions with ReVolt are done at Dora 1 harbor basin in Trondheim.

Sammendrag

Reguleringssystemet til den 3 meter lange ReVolt har blitt utvidet til å støtte 2-D banefølging ved å bruke *"Line-of-Sight"* prinsippet med *"lookahead-based"* styringsalgoritme for å beregne referansekurs. Referansemodeller for kurs- og hastighetsdynamikken er utviklet for å glatte referansesignalene og danne høyere ordens deriverte for reguleringssystemet. En separat programvare for visuell plassering av veipunkter i et navigasjonskart og overføring av de til ReVolt, er skrevet i *C++* ved å bruke *"Qt Creator"*. Den utviklede programvaren er brukt til overvåking og fjernstyring av ReVolt under simuleringer og eksperiment. Kontrollsystemet har blitt utvidet med en forbedret regulator for å holde følge korrekt kurs ved kursendringer. En første ordens Nomoto-modell er brukt til å beregne en modell-basert foroverkobling i tillegg til forsterkningskonstanter i bakoverkoblingen. En regulator for ønsket hastighet forover er utviklet, denne benytter også en forover- og bakoverkobling.

I transitt, er de to aktre azimuth-thrusterene begrenset til $\pm 45^o$ i reguleringssystemet og baugthrusteren er hevet. Dette gjør at tverrskips-bevegelsen er ukontrollert, hvilket betyr at det er en underaktuert konfigurasjon. Kildekoden til regulerings- og guidance-systemet er skrevet i *C++* og startes av ROS på datamaskinen ombord.

Systemidentifikasjon og testing av regulerings- og guidance-systemet er først gjort ved bruk av ReVolts digitale tvilling i simulatoren utviklet av DNV GL. Full-skala eksperiment for å vurdere ytelsen til de implementerte løsninngene er gjort i havnebassenget Dora 1 i Trondheim.

Table of Contents

List of Tables

List of Figures

Abbreviations

DNV GL	=	Det Norske Veritas Germanischer Lloyd
GNC	=	Guidance, Navigation and Control
LOS	=	Line-of-Sight
DOF	=	Degree of Freedom
ECI	=	Earth Centered Inertial frame
ECEF	=	Earth Centered Earth Fixed reference frame
NED	=	North, East Down coordinate system
IIR	=	Infinite Impulse Response
FIR	=	Finite Impulse Response
FFT	=	Fast Fourier Transform
RCM	=	Remote Control & Monitoring
OS	=	Operating System
VM	=	Virtual Machine
ROS	=	Robot Operating System
API	=	Application Programming Interface
SDK	=	Software Development Kit
I/O	=	Input/Output
GUI	=	Graphical User Interface
OSI	=	Open System Interconnection
IP	=	Internet Protocol
TCP	=	Transport Control Protocol
UDP	=	User Datagram Protocol
4G	=	Fourth Generation Mobile Broadband
LTE	=	Long Term Evolution
Wi-Fi	=	Wireless Local Area Network
Mutex	=	Mutual Exclusion
VSTS	=	Visual Studio Team Services
USB	=	Universal Serial Bus
IMU	=	Inertial Measurement Unit
GNSS	=	Global Navigation Satellite Systems
FF	=	Feedforward
FB	=	Feedback
PID	=	Proportional Integral Derivative
GPS	=	Global Positioning System
DP	=	Dynamic Positioning
RAM	=	Random Access Memory
CPU	=	Central Processing Unit

Nomenclature

$\{n\}$	=	Denotes the North-East-Down frame
$\{b\}$	=	Denotes the BODY-fixed frame
$\{e\}$	=	Denotes the Earth-Centered Earth-Fixed frame
$\{i\}$	=	Denotes the Earth-Centered Inertial frame
$\boldsymbol{\eta}$	=	Position and attitude vector
$\boldsymbol{\nu}$	=	Linear and angular velocity vector
$\boldsymbol{\tau}$	=	Generalized forces and moments vector
$\boldsymbol{p}$	=	Position vector
$\boldsymbol{\Theta}$	=	Attitude vector (Euler angles)
$\boldsymbol{v}$	=	Linear velocity vector
$\boldsymbol{\omega}$	=	Angular velocity vector
$\boldsymbol{f}$	=	Force vector
$\boldsymbol{m}$	=	Moment vector
x, y, z	=	Positions in $\{e\}$
N, E, D	=	North, East, Down positions in $\{n\}$
u, v, w	=	BODY-fixed linear velocities
l, μ	=	Longitude and latitude
ϕ, θ, ψ	=	Euler angles about the x, y and z-axis
p, q, r	=	Body-fixed angular velocities
X, Y, Z	=	Body-fixed force
K, M, N	=	Body-fixed moments
$s(\cdot), c(\cdot)$	=	$\sin(\cdot), \cos(\cdot)$
$\boldsymbol{R}$	=	Rotation matrix
$\boldsymbol{M}$	=	Inertia matrix
$\boldsymbol{M_{RB}}$	=	Rigid-body inertia matrix
$\boldsymbol{C_{RB}}$	=	Rigid-body Coriolis-centripetal matrix
$\boldsymbol{C_A}$	=	Added mass Coriolis-centripetal matrix
$\boldsymbol{D}$	=	Damping matrix
$\boldsymbol{\Delta}$	=	Relative damping ratio matrix (boldface)
$\boldsymbol{\Omega}$	=	Natural frequency matrix
$\boldsymbol{A}_d$	=	Reference model system matrix
$\boldsymbol{B}_d$	=	Reference model actuator configuration matrix
$\boldsymbol{x}_d$	=	Reference model desired state vector
$\boldsymbol{r}^b$	=	Operator input in $\{b\}$
h	=	Discretization time step
wp	=	List of waypoints in $\{n\}$
p_k^n =	=	Position of waypoint k in $\{n\}$
α_k	=	Path-tangential angle

ϵ	=	Along-track distance and cross-track error vector
e	=	Cross-track error
s	=	Along-track distance
χ_d	=	Desired course angle
χ_p	=	α_k
Δ	=	lookahead distance
ψ_{los}	=	Heading angle from LOS Guidance System
β	=	Sideslip angle
s_{k+1}	=	Distance from waypoint k to $k+1$
R_{k+1}	=	Radius of acceptance of waypoint $k+1$
U_{ref}	=	Reference speed over ground
u_{ref}	=	Reference surge speed
$\boldsymbol{F}$	=	Thruster force produced vector
$\boldsymbol{K}$	=	Force coefficient matrix
$\boldsymbol{u}$	=	Control input to actuator(s)
$\boldsymbol{T(\alpha)}$	=	Thrust configuration matrix
$\boldsymbol{\alpha}$	=	Thruster angles in $\{b\}$
δ_i	=	Thruster angle command for thruster i
l_{x_i}	=	Length to thruster i in x-direction in $\{b\}$
l_{y_i}	=	Length to thruster i in y-direction in $\{b\}$
K_m	=	Thrust coefficient scalar
τ_{m_i}	=	Thruster effort for thruster i
ω_n	=	Natural frequency
ζ	=	Damping ratio
sat$(\cdot)$	=	Saturating element
sgn$(\cdot)$	=	Sign
F_s	=	Sample frequency
F_{pass}	=	End of pass-band frequency
F_{stop}	=	start of stop-band frequency
A_{pass}	=	Pass-band gain
A_{stop}	=	Stop-band gain
$\tau_{i,FF}$	=	Feedforward control input for i
$\tau_{i,FB}$	=	Feedback control input for i
M	=	Mass factor
$\sigma(u_d)$	=	Damping polynomial
K_p	=	Proportional gain
K_i	=	Integral gain
K_d	=	Derivative gain
K	=	Nomoto gain
T	=	Nomoto time constant
s	=	Frequency domain variable
b	=	Slowly varying unknown bias
ω_b	=	Bandwidth

Chapter 1

Introduction

1.1 Motivation

Figure 1.1: DNV GL's experimental platfrom ReVolt.

Autonomous shipping is said to be the future of the maritime industry [2]. To realize remote controlled and autonomous ships, a robust system for path following and low-level controllers is an important part for maneuvering a marine craft autonomously. In the future, ReVolt (Figure 1.1) is expected to be fully autonomous. These results are the next step towards autonomous maneuvering of ReVolt, and enables implementation and testing of more advanced algorithms for decision-making and collision avoidance on a physical model.

1.2 Review

The motion control task of path following have undergone extensive research in recent years. In [3], an overview of the LOS guidance for path following in the 2-D horizontal plane, and the 3-D scenario with a 5-degree of freedom (DOF) underwater vehicle is presented. In [4], the problem of straight-line path following for a fully actuated marine craft is studied. An assumption often made for path following is that marine crafts use only their aft main propellers and rudder for forward speed and steering, leaving sway uncontrolled. This underactuated configuration is presented in [5], where an approach for tracking straight-line segments at high speeds with an unmanned surface vessel (USV) using a surge speed and yaw rate controller during full-scale experiments in Trondheimsfjorden. The underactuated ship problem is also assessed by [6] where tracking problems are tested and solved using surge force and yaw moment as control inputs. In [7], a predictor-based LOS guidance law for path following of an underactuated marine craft is presented. Here, the predictor estimates the sideslip angle with high accuracy in steady state and transient.

1.3 Problem Description

In this thesis, the goal is to expand ReVolt's control system to include a Guidance System which enables ReVolt to follow a predefined path with a desired speed while steering along a LOS vector. This includes the low-level controllers for speed and heading angle as well. Furthermore, a Guidance Management System is to be implemented in the RMC station from the specialization project such that waypoints can be placed in the navigation map and transmitted to ReVolt. The controllers and LOS steering algorithm should be tested using ReVolt's Digital Twin (simulator) before performing experimental tests at sea. The RMC station is used for monitoring and control of ReVolt both during simulations and at sea.

1.4 Contributions

The main contributions in this thesis are:

- Add Guidance Management functionality to the RMC station which includes waypoint generation, path, footprint and obstacle visualization.
- Develop and implement a new heading controller for ReVolt, which includes a reference model, feedforward and feedback term for use in the path following system.
- Develop and implement a surge speed controller for ReVolt, which also includes a reference model, feedforward and feedback term for use in the path following system.

- Implement a Guidance System for path following for ReVolt using the lookahead-based steering algorithm.
- Simulate and assess the new heading and speed controllers' tracking performance separately.
- Simulate and assess the path following performance using the new heading and speed controllers.
- Perform similar experimental tests at sea and assess the performance.
- Improve image streaming problem from [8].

1.5 Outline

This thesis contains a total of **eight** chapters. **Chapter 1** contains the introduction, followed by **Chapter 2** with theory and concepts for path following adapted from [9]. **Chapter 3** presents an introduction to the RMC station developed in the specialization project and the new contributions, preceded by necessary theory regarding networking and software fundamentals. In **Chapter 4**, the implemented Guidance, Navigation and Control (GNC) System for path following is described. This includes choice of control allocation, development of heading and speed controller and implementation in the control system with ROS environment. Simulation results for the heading controller, speed controller, path following and a discussion relating to the results is presented in **Chapter 5**. **Chapter 6** presents the corresponding experimental results and a discussion regarding them. **Chapter 7** presents a discussion implemented solutions. Lastly, a concluding remark and future work is presented in **Chapter 8**.

Chapter 2

Theory and Concepts for Path Following

This chapter is heavily based on the ”Handbook of Marine Craft Hydrodynamics and Motion Control” by Fossen 2011 [9].

2.1 Guidance, Navigation and Control Systems

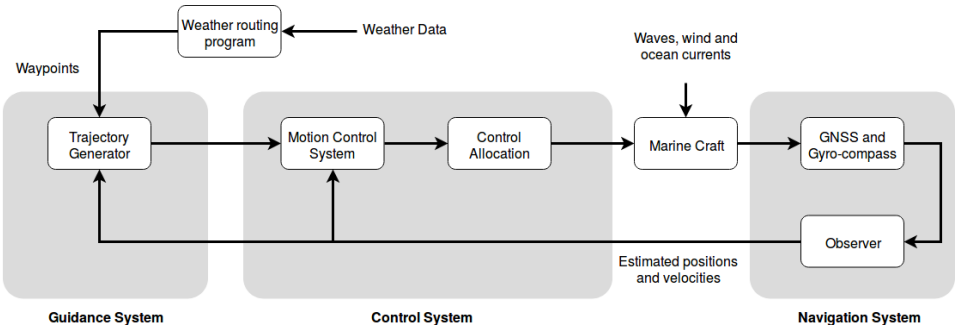


Figure 2.1: Illustration of a generic Guidance, Navigation and Control system [9]

Figure 2.1 shows the three main components of a GNC System. The leftmost block, *Guidance System*, is concerned with generating a desired trajectory for the marine craft. These can be either time-varying or time-invariant. Using time-varying reference signals, known as trajectory tracking, forces the marine craft to track a given reference value at a specific time. Time-invariant relates to *path following* in which a path is predefined and places no

constraint on time, only spatial constraints such as known obstacles. The reference signals can be generated by e.g. a joystick, keyboard, weather data, guidance law algorithms or collision avoidance data. The reference trajectories are designed by using reference models obtained from e.g low-pass filtering or through simulations to ensure feasible tracking (marine craft able to follow). Lastly, the simplest form is called *setpoint regulation*, in which desired position and attitude is constant.

In close cooperation with the guidance block, the *Control System* uses the scenarios explained above to carry out its control objective by generating the correct forces and moments necessary to maneuver the marine craft based on e.g. PID or Linear Quadratic (LQ) Control. These are output from the *Motion Control System* and translated to thruster effort and direction by the *Control Allocation*.

The rightmost block in a GNC system is the *Navigation* block. Here, a signal processing unit checks the raw GPS measurements for wild-points. The filtered positions are transmitted to the state estimator, an algorithm that process sensor and navigation data to provide noise-filtered estimates of both measured and unmeasured states.

2.2 Kinematics

2.2.1 Motion Variables

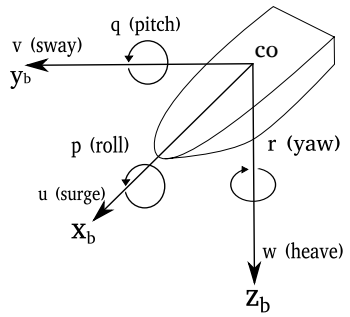


Figure 2.2: Illustration of BODY-fixed velocities in 6 DOF. CO denotes center of origin. Figure adapted from [9]

To determine the position and attitude of a marine craft moving in 6 DOF, six independent coordinates are necessary. The former three states, and their derivatives, correspond to position and translational motion (surge, sway and heave) along the x, y and z axes. The three latter states, and their derivatives, corresponds attitude and angular velocity (roll, pitch and yaw). The velocities are illustrated in Figure 2.2 and listed in table 2.1.

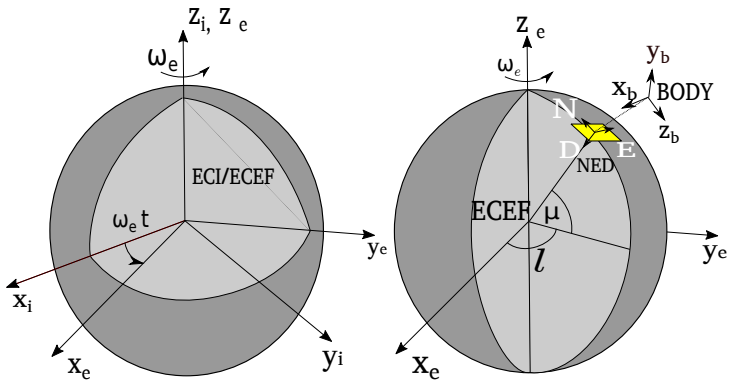
DOF	Description	Forces and moments	Linear and angular velocities	Positions and Euler angles
1	motions in the x direction (surge)	X	u	x
2	motions in the y direction (sway)	Y	v	y
3	motions in the z direction (heave)	Z	w	z
4	rolls about the x axis (roll)	K	p	ϕ
5	rolls about the y axis (pitch)	M	q	θ
6	rolls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

Table 2.1: SNAME notation (1950) for marine vessels [9].

2.2.2 Reference Frames

For GNC and analysis it is convenient to define earth-centered and geographic reference frames.

- **ECI:** Earth-centered inertial frame, denoted $\{\mathrm{i}\} = (x_i, y_i, z_i)$, is a nonaccelerating reference frame, i.e. inertial and Newton's laws apply. Its origin is located at the center of the earth, as shown in Figure 2.3a.
- **ECEF:** Earth-centered Earth-fixed reference frame, denoted $\{\mathrm{e}\} = (x_e, y_e, z_e)$ has the same origin as ECI (see Figure 2.3a), but rotates relative to it with an angular velocity of $\omega_e = 7.2921 \times 10^5$ rad/s [9]. ECEF is often considered inertial for maneuvering of marine crafts and used for GNC during long distance transit [9].
- **NED:** North-East-Down coordinate system denoted $\{\mathrm{n}\} = (x_n, y_n, z_n)$ is usually defined as a tangent plane on the surface of the earth. Its axes x, y and z is always pointing to the true north, east and down normal to the earth's surface, respectively. The NED coordinate system relative to ECEF is determined by two angles latitude μ and longitude l as seen in Figure 2.3b. During navigation, NED is fixed to the earth's surface and considered inertial such that Newton's laws apply [9].
- **BODY:** The Body frame is denoted $\{\mathrm{b}\} = (x_b, y_b, z_b)$ and is fixed to the marine craft. The x-axis of the frame points towards the bow of the ship, while the y and z axes points to the starboard and down, respectively (see Figure 2.2). The position and attitude of a marine craft should be expressed in an inertial frame, e.g. ECEF, NED. The linear and angular velocities of the marine craft should be expressed in BODY.



(a) ECI and ECEF **(b)** NED and BODY

Figure 2.3: ECI, ECEF, NED and BODY coordinate systems. Figure adapted from [9].

2.2.3 Definitions of Course, Heading and Sideslip Angles

For marine crafts it is necessary to define the direction to where it is moving.

Course angle χ: "The angle from the x-axis of the NED frame to the velocity vector of the vehicle, positive rotation about the z-axis of the NED frame by the right-hand screw convention" [10].

Heading angle ψ: "The angle from the NED x-axis to the BODY x-axis, positive rotation about the z-axis of the NED frame by the right-hand screw convention" [10].

Sideslip angle β: "The angle from the BODY x-axis to the velocity vector of the vehicle, positive rotation about the BODY z-axis frame by the right-hand screw convention" [10].

$$\beta \triangleq \chi - \psi, \qquad \beta = \arcsin(\frac{v}{U}) \tag{2.1}$$

where $U = \sqrt{u^2 + v^2}$ is the speed over ground measured by e.g. GNSS.

2.2.4 Notation

For marine craft the following notation is adopted for vectors in the coordinate systems $\{\mathrm{b}\}$, $\{\mathrm{e}\}$, $\{\mathrm{n}\}$:

$\mathbf{v}^e_{b/n}$ =	linear velocity of the point o_b with respect to $\{\mathrm{n}\}$ expressed in $\{\mathrm{e}\}$
$\boldsymbol{\omega}^b_{n/e}$ =	angular velocity of $\{\mathrm{n}\}$ with respect to $\{\mathrm{e}\}$ expressed in $\{\mathrm{b}\}$
$\mathbf{f}^n_b$ =	force with line of action through the point o_b expressed in $\{\mathrm{n}\}$
$\mathbf{m}^n_b$ =	moment about the point o_b expressed in $\{\mathrm{n}\}$
$\boldsymbol{\Theta}_{nb}$ =	Euler angles between $\{\mathrm{n}\}$ and $\{\mathrm{b}\}$

We can now express the different quantities in Table 2.1, defined by SNAME (1950) in a vectorial setting:

ECEF position	$\boldsymbol{p}^e_{b/e} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	Longitude and latitude	$\boldsymbol{\Theta}_{en} = \begin{bmatrix} l \\ \mu \end{bmatrix}$
NED position	$\boldsymbol{p}^n_{b/n} = \begin{bmatrix} N \\ E \\ D \end{bmatrix}$	Attitude (Euler angles)	$\boldsymbol{\Theta}_{nb} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$
Body-fixed linear velocity	$\boldsymbol{v}^b_{b/n} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$	Body-fixed angular velocity	$\boldsymbol{\omega}^b_{b/n} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$
Body-fixed force	$\boldsymbol{f}^b_b = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$	Body-fixed moment	$\boldsymbol{m}^b_b = \begin{bmatrix} K \\ M \\ N \end{bmatrix}$

Using these, the general motion of a 6 DOF marine craft can be described with the following vectors:

$$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}^n_{b/n} \\ \boldsymbol{\Theta}_{nb} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} \boldsymbol{v}^b_{b/n} \\ \boldsymbol{\omega}^b_{b/n} \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{f}^b_b \\ \boldsymbol{m}^b_b \end{bmatrix} \tag{2.2}$$

2.2.5 Transformation Between BODY and NED

The rotation matrix from BODY to NED for a 6 DOF marine craft is:

$$\boldsymbol{R}^n_b(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} \mathrm{c}\psi\mathrm{c}\theta & -\mathrm{s}\psi\mathrm{c}\phi + \mathrm{c}\psi\mathrm{s}\theta\mathrm{s}\phi & \mathrm{s}\psi\mathrm{s}\phi + \mathrm{c}\psi\mathrm{c}\phi\mathrm{s}\theta \\ \mathrm{s}\psi\mathrm{c}\theta & \mathrm{c}\psi\mathrm{c}\phi + \mathrm{s}\phi\mathrm{s}\theta\mathrm{s}\psi & -\mathrm{c}\psi\mathrm{s}\phi + \mathrm{s}\theta\mathrm{s}\psi\mathrm{c}\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{2.3}$$

where s = $\sin(\cdot)$, c = $\cos(\cdot)$ and $\boldsymbol{R}^n_b(\boldsymbol{\Theta}_{nb})$ is element in SO(3).

The body-fixed velocity vector can now be expressed in $\{\mathrm{n}\}$ as:

$$\dot{\boldsymbol{p}}^n_{b/n} = \boldsymbol{R}^n_b(\boldsymbol{\Theta}_{nb})\boldsymbol{v}^b_{b/n} \tag{2.4}$$

For control design, roll ϕ and pitch θ is often neglected simplifying (2.3) to:

$$\boldsymbol{R}^n_b(\boldsymbol{\Theta}_{nb}) = \boldsymbol{R}_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

under the assumption that $\phi = \theta = 0$. This is commonly used in Dynamic Positioning (DP) systems and local path following routes due to its "flat earth" tangent plane.

2.2.6 Transformation Between NED and ECEF

The rotation matrix from NED to ECEF is:

$$\boldsymbol{R}_n^e(\boldsymbol{\Theta}_{en}) = \begin{bmatrix} -\mathrm{c}l\mathrm{s}\mu & -\mathrm{s}l & -\mathrm{c}l\mathrm{c}\mu \\ -\mathrm{s}l\mathrm{s}\mu & \mathrm{c}l & -\mathrm{s}l\mathrm{c}\mu \\ \mathrm{c}\mu & 0 & -\mathrm{s}\mu \end{bmatrix} \tag{2.6}$$

where s = $\sin(\cdot)$, c = $\cos(\cdot)$ and $\boldsymbol{R}_n^e(\boldsymbol{\Theta}_{en}) \in$ SO(3).

The body fixed velocity vector can now be expressed in $\{\mathrm{e}\}$ as:

$$\dot{\boldsymbol{p}}_{b/e}^e = \boldsymbol{R}_n^e(\boldsymbol{\Theta}_{en})\dot{\boldsymbol{p}}_{b/e}^n = \boldsymbol{R}_n^e(\boldsymbol{\Theta}_{en})\boldsymbol{R}_b^n(\boldsymbol{\Theta}_{nb})\boldsymbol{v}_{b/n}^b \tag{2.7}$$

This transformation is necessary when designing global path following system, due to large variations in latitude μ and longitude l.

2.3 Modeling of Marine Crafts

The majority of surface vessel models are based on the 3-DOF model

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{2.8}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \boldsymbol{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_{wind} + \boldsymbol{\tau}_{wave} \tag{2.9}$$

where

$$\boldsymbol{\eta} = \begin{bmatrix} N \\ E \\ \psi \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ r \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} \tag{2.10}$$

$\boldsymbol{\nu}_r$ denotes the relative velocity between the vessel and the water. τ_{wind} and τ_{wave} represent disturbances. The matrix $\boldsymbol{R}(\psi)$ is rotation about the z-axis. $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$ is the inertia matrix consisting of the rigid-body (RB) mass matrix and added mass (A). $\boldsymbol{C}_{RB}$is the rigid-body Coriolis-centripetal matrix and $\boldsymbol{C}_A$ is the Coriolis added mass matrix. $\boldsymbol{D}(\boldsymbol{\nu}_r)$ represents the hydrodynamic damping of the vessel.

2.4 Reference Models

Reference models is part of a guidance system and aim to create feasible trajectories for position, attitude and/or velocity that the marine craft is able to follow. The simplest form of a reference model is an open-loop linear low-pass filter, which are simple to implement as well.

2.4.1 Velocity Reference Model

A velocity reference model should be at least of order 2 to obtain smooth reference signals for desired velocity and acceleration. The second order low-pass filter is

$$\ddot{\boldsymbol{\nu}}_d + 2\boldsymbol{\Delta}\boldsymbol{\Omega}\dot{\boldsymbol{\nu}}_d + \boldsymbol{\Omega}^2\boldsymbol{\nu}_d = \boldsymbol{\Omega}^2\boldsymbol{r}^b \tag{2.11}$$

where $\boldsymbol{\nu}_d$ is the desired velocity, $\dot{\boldsymbol{\nu}}_d$ is the desired acceleration and $\ddot{\boldsymbol{\nu}}_d$ is interpreted as the desired "jerk". $\boldsymbol{\Delta}$ and $\boldsymbol{\Omega}$ are positive definite, diagonal design matrices containing relative damping ratios and natural frequencies

$$\boldsymbol{\Delta} = \text{diag}\{\zeta_1, \zeta_2, ..., \zeta_n\} \tag{2.12}$$

$$\boldsymbol{\Omega} = \text{diag}\{\omega_{n_1}, \omega_{n_2}, ..., \omega_{n_n}\} \tag{2.13}$$

and $\boldsymbol{r}^b$ is the operator input expressed in $\{\text{b}\}$. In steady state (no acceleration or jerk)

$$\lim_{t\to\infty} \boldsymbol{\nu}_d(t) = \boldsymbol{r}^b \tag{2.14}$$

In state-space form this results in

$$\underbrace{\begin{bmatrix}\dot{\boldsymbol{\nu}}_d \\ \ddot{\boldsymbol{\nu}}_d\end{bmatrix}}_{\dot{\mathbf{x}}_d} = \underbrace{\begin{bmatrix}\mathbf{0} & \boldsymbol{I} \\ -\boldsymbol{\Omega}^2 & -2\boldsymbol{\Delta}\boldsymbol{\Omega}\end{bmatrix}}_{\mathbf{A}_d}\underbrace{\begin{bmatrix}\boldsymbol{\nu}_d \\ \dot{\boldsymbol{\nu}}_d\end{bmatrix}}_{\mathbf{x}_d} + \underbrace{\begin{bmatrix}0 \\ \boldsymbol{\Omega}^2\end{bmatrix}}_{\mathbf{B}_d}\underbrace{\boldsymbol{r}^b}_{\mathbf{u}} \tag{2.15}$$

2.4.2 Position and Attitude Reference Model

A third order reference filter for position and/or attitude is:

$$\frac{\eta_{d_i}}{r_i^n} = \frac{\omega_{n_i}^3}{(s+\omega_{n_1})(s^2 + 2\zeta_i\omega_{n_i}s + \omega_{n_i}^3)} \tag{2.16}$$

which is represented in state space formulation as

$$\underbrace{\begin{bmatrix}\dot{\boldsymbol{\eta}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{\eta}}_{\boldsymbol{d}} \\ \dddot{\boldsymbol{\eta}}_{\boldsymbol{d}}\end{bmatrix}}_{\dot{\boldsymbol{x}}_d} = \underbrace{\begin{bmatrix}\mathbf{0} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} \\ -\boldsymbol{\Omega}^3 & -(2\boldsymbol{\Delta}+\boldsymbol{I})\boldsymbol{\Omega}^2 & -(2\boldsymbol{\Delta}+\boldsymbol{I})\boldsymbol{\Omega}\end{bmatrix}}_{\boldsymbol{A}_d}\underbrace{\begin{bmatrix}\boldsymbol{\eta}_d \\ \dot{\boldsymbol{\eta}}_d \\ \ddot{\boldsymbol{\eta}}_d\end{bmatrix}}_{\boldsymbol{x}_d} + \underbrace{\begin{bmatrix}\mathbf{0} \\ \mathbf{0} \\ \boldsymbol{\Omega}^3\end{bmatrix}}_{\boldsymbol{B}_d}\underbrace{\boldsymbol{r}^b}_{\boldsymbol{u}} \tag{2.17}$$

where the matrices $\boldsymbol{\Delta}$ and $\boldsymbol{\Omega}$ have the same properties as (2.12). The reference model also satisfy

$$\lim_{t\to\infty} \boldsymbol{\eta}_d(t) = \boldsymbol{r}^n \tag{2.18}$$

as long as $\boldsymbol{r}^n$ is constant

2.4.3 Discretization

To implement the reference filter on a computer, a discretization is necessary. Using *Discrete Euler Method* [11] yields:

$$\mathbf{x}_{d_{k+1}} = \mathbf{x}_{d_k} + h[\mathbf{A}_d \mathbf{x}_{d_k} + \mathbf{B}_d \mathbf{u}_k] \tag{2.19}$$

where $\mathbf{A}_d \in R^{n \times n}$ is the continuous-time system model matrix, $\mathbf{B}_d \in R^{n \times p}$ is the continuous-time input matrix and h is the sampling time.

2.5 Line-of-Sight Guidance

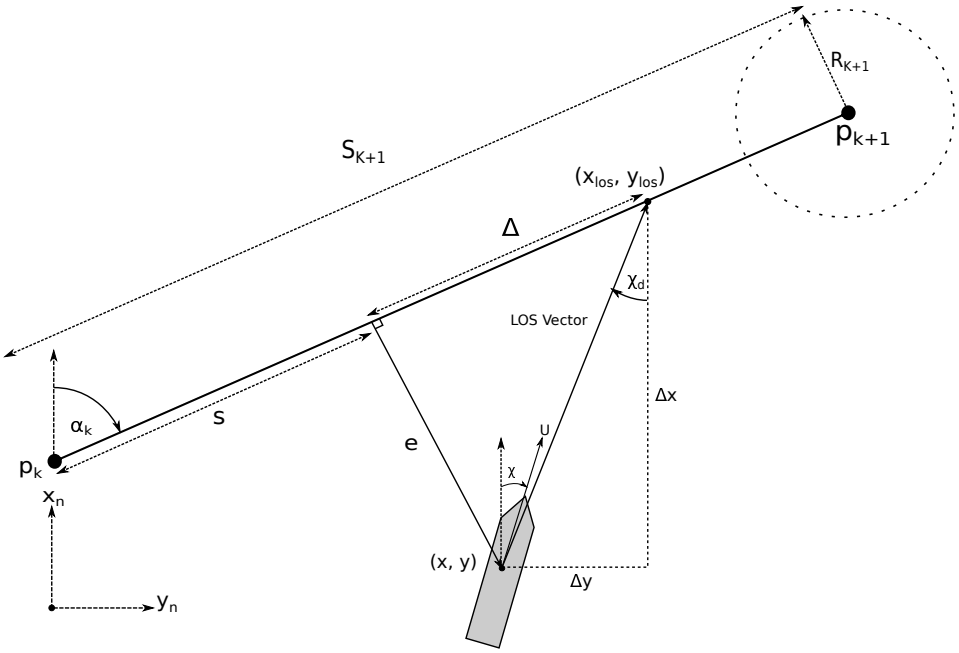


Figure 2.4: LOS guidance illustration expressed in $\{\mathrm{n}\}$. Figure adapted from [9].

Originally used in surface-to-air missiles, the LOS guidance is also applied to marine crafts for path following tasks. "*Path following* is the task of following a predefined path independent of time; that is there are no temporal constraints" [9]. A path is often made up of a sequence of interconnected straight lines defined by a list of waypoints expressed in the $\{\mathrm{n}\}$-frame. The position of waypoint k and the list of waypoints is defined as

$$\boldsymbol{p}_k^n := \begin{bmatrix} x_k & y_k \end{bmatrix}^\top \tag{2.20}$$

$$wp := \begin{bmatrix} \boldsymbol{p}_1^n & \boldsymbol{p}_2^n & \dots & \boldsymbol{p}_n^n \end{bmatrix} \tag{2.21}$$

for a path consisting of n number of waypoints and $n-1$ straight line segments.

In Figure 2.4, the path is defined by a straight line between the two waypoints $\boldsymbol{p}_k^n$ and $\boldsymbol{p}_{k+1}^n$. The path is inherently rotated an angle α_k relative to the x_n axis and is given by

$$\alpha_k = \text{atan2}(y_{k+1} - y_k, x_{k+1} - x_k) \tag{2.22}$$

where atan2(y, x) is the four-quadrant version of atan(y, x) $\in [-\pi/2, \pi/2]$ [12].

By using the angle obtained from (2.22) and the marine craft's position $\boldsymbol{p}^n(t)$ relative to $\boldsymbol{p}_k^n$, we express its position in the path-fixed coordinate system. This has its origin in $\boldsymbol{p}_{\boldsymbol{k}}^{\boldsymbol{n}}$ and rotated α_k degrees, using

$$\boldsymbol{\epsilon}(t) = \boldsymbol{R}_p(\alpha_k)^\top (\boldsymbol{p}^n(t) - \boldsymbol{p}_k^n) \tag{2.23}$$

where

$$\boldsymbol{R}_p(\alpha_k) := \begin{bmatrix} \cos(\alpha_k) & -\sin(\alpha_k) \\ \sin(\alpha_k) & \cos(\alpha_k) \end{bmatrix} \in \text{SO(2)} \tag{2.24}$$

$\boldsymbol{\epsilon}(t) = [s(t), e(t)]^\top$ is the along-track distance and cross-track error (normal to path), respectively. SO(m) states that the matrix is orthogonal, has determinant equal to 1 and is of order m. The control objective becomes

$$\lim_{t \to \infty} e(t) = 0 \tag{2.25}$$

i.e. minimizing the cross-track error. If this statement holds, the marine craft will converge to the path asymptotically.

The LOS vector is projected from the marine craft and intersects a point (x_{los}, y_{los}) on the path defined by the two waypoints (see Figure 2.4). The distance Δ denotes the *Lookahead-distance* and decides how far from the along-track distance $s(t)$ the LOS vector will intersect and the steering law's aggressiveness.

To steer along the LOS vector, two guidance principles can be used [12]:

- *Enclosure-based steering*
- *Lookahead-based steering*

In this thesis, only *Lookahead-based steering* will be presented, due to being less computationally expensive [12].

2.5.1 Lookahead-based Steering

The desired course angle χ_d assignment for *lookahead-based steering* is formulated as

$$\chi_d(e) = \chi_p + \chi_r(e) \tag{2.26}$$

where

$$\chi_p = \alpha_k \tag{2.27}$$

is the angle between the north axis of $\{\mathrm{n}\}$ and the straight line between $\boldsymbol{p}_k^n$ and $\boldsymbol{p}_{k+1}^n$ (see Figure 2.4), while

$$\chi_r(e) := \arctan\left(\frac{-e}{\Delta}\right) \tag{2.28}$$

where e is the cross-track error and $\Delta > 0$ is the lookahead distance usually set to 1.5-2.5 ship lengths [9]. From (2.28), a small Δ yields a large $\chi_r(e)$ causing more aggressive convergence.

To enable a marine craft to follow a predefined path under the influence of ocean currents, the controller must account for the *Sideslip*-angle β. Threating the current as a slowly varying disturbance and adding integral action in the LOS steering (integral LOS), or if velocity measurements are available the output can be

$$\psi_d = \chi_d - \beta \tag{2.29}$$

where β can be calculated as

$$\beta = \arcsin\left(\frac{v}{U}\right) \tag{2.30}$$

v is the sway velocity and U is the speed over ground [9]. In this thesis, velocity measurements are available so that (2.29) can be used.

2.5.2 Switching Mechanism for Waypoints

For changing straight line segments to track during transit, a switching mechanism is needed. One method is to define a *circle of acceptance* with radius R_{k+1} (see Figure 2.4). Switching to the next waypoint occurs when the vessel is inside this circle. The equation is

$$[x_{k+1} - x(t)]^2 + [y_{k+1} - y(t)]^2 \leq R_{k+1}^2 \tag{2.31}$$

However, a criterion that doesn't require the vessel to be inside the circle, uses the along-track distance s for switching such that

$$s_{k+1} - s(t) \leq R_{k+1} \tag{2.32}$$

where s_{k+1} is the distance between waypoint k and $k+1$ (see Figure 2.4). A guideline for deciding the value of R_{k+1} is two times ship length.

Chapter 3

Remote Monitoring & Control

This chapter is an extension of the work done in *"Remote Monitoring & Control of an Autonomous Boat"* [8] as the specialization project TTK4551 fall of 2017. The specialization project describes the software development and implementation at a more detailed manner. However, a brief review of necessary theory, regarding networking and software fundamentals, will be presented here. A description of existing functionality along with the new contributions to the software produced in thesis will also be presented.

3.1 Networking and Software Fundamentals

Open Systems Interconnection or OSI-model for short, is a reference model for network communication. It consists of the 7 layers shown in Figure 3.1 where each layer in the stack provides a service for the layer above. When data is transmitted, it is passed down from the application layer where a header corresponding to the transport layer is added first. This header is used by the transport layer at the receiving end and contains information such as source, destination and error-checking. It then continues down the stack of the senders side, transmitted through a physical link (network cable), before returning up the stack to the receiver [14].

3.1.1 Sockets and Protocols

Sockets are end-points in two-ways communication between threads (3.1.4) or processes (3.1.3) communicating over a packet-switching network, e.g. the Internet.

Transport Control Protocol (TCP) is one of two transport protocols residing in layer 4. It is a connection-oriented and reliable protocol that transmits and receives data between processes through **sockets**. To establish connection, the two processes must perform a

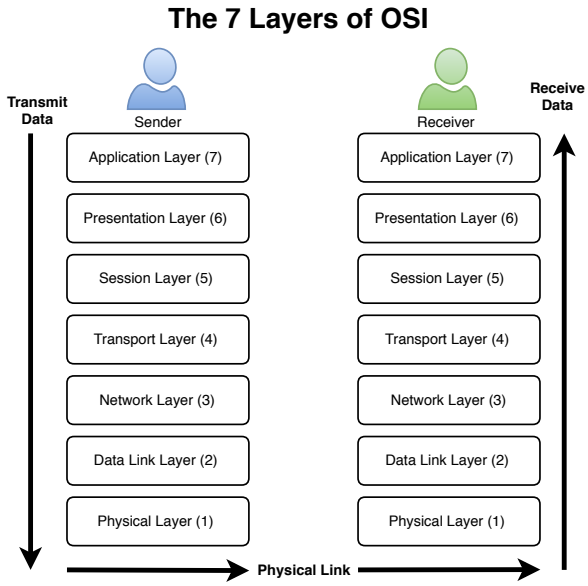


Figure 3.1: Illustration of the 7 layer OSI model. Figure adapted from [13] and [14].

”three-way-handshake”. The sender transmits a ”SYN” (synchronize) packet to the receiver, which then returns a ”SYN-ACK” (acknowledgement) to the sender. The sender then transmits an ”ACK” packet to receiver and connection is established (SYN, SYN-ACK, ACK). TCP supports flow-control (same speed between sender and receiver) and re-transmission of lost packets. The connection remains open until the client closes the connection [14].

User Datagram Protocol (UDP) is the other transport protocol and has no acknowledgement of transmitted packets, or datagrams as they are referred to in UDP. It does not even know if someone is receiving the datagrams sent. This is why UDP is referred to as a connection-less protocol. The lack of ACK-segments sent increases the UDPs throughput, meaning it can transmit a higher amount of data than TCP. The reason to use UDP over TCP is when high transmission rates are necessary and loss of datagrams can be tolerated, e.g. live video stream. Using checksums to detect bit-errors are employed by both UDP and TCP. If one or more bit-errors are detected in the received datagrams or packets they are discarded [14].

The **Internet Protocol (IP)** is not a transport protocol as it resides in the Network layer of the OSI-model. It handles addressing and routing of packets/datagrams throughout the Internet. Every device containing an network interface, e.g. Wi-Fi card, receives a unique IP-address. This protocol along with routing protocols ensures that the packets/datagrams is sent to the correct recipient. There are two IP versions, IPv4 and IPv6. The main difference is the size of the address field. IPv4 uses 32 bit for addressing while IPv6 uses 128 bits. IPv6 is newer and larger due to the lack of IPv4 addresses available [14].

3.1.2 Socket Programming

To establish a connection between two processes in a client-server paradigm, a library called *"Practical C++ Sockets"* can be used. This library is developed for pedagogical reasons by Berkeley University and is a work in progress. It does, however, provide simple means for socket communication by creating a few wrapping classes for a subset of Berkeley C Socket API for TCP and UDP sockets. Their website provides class documentation and a few examples with both TCP and UDP protocols [15]. The interface is supported in both Windows and Unix systems.

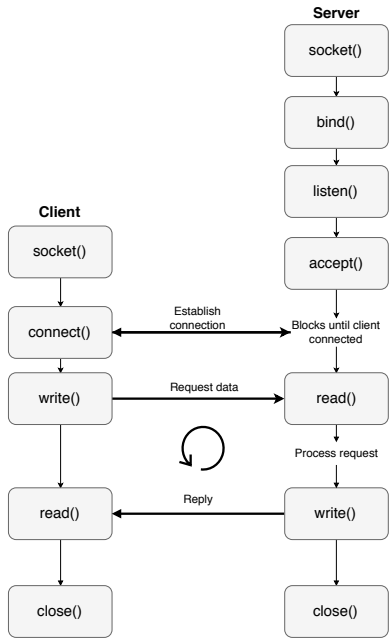


Figure 3.2: Client-Server paradigm for a TCP socket connection. Figure adopted from [8].

Figure 3.2 shows the protocol for a TCP connection between a client and a server. Firstly, a socket-object is created by the server. This object then binds an IP-address and a specific port number to it. The server will then listen for connections on that IP-address and port number. `accept()` will block the execution of that **thread** (see 3.1.4) until a client has connected using the server's IP-address and port number. After the client established connection it then writes a request which is read and processed by the server before returning an appropriate response. This is read/write sequence can be repeated until the client task is completed. TCP is connection oriented and will terminate when the client disconnects [14].

Figure 3.3 shows the protocol for a UDP connection between a client and a server. Since UDP is connectionless, both client and server needs to bind an IP-address and a port num-

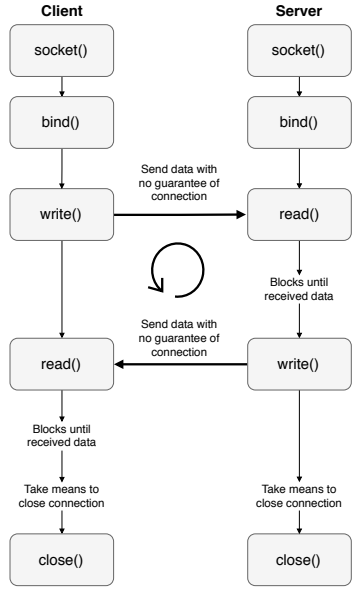


Figure 3.3: Client-Server paradigm for a UDP socket connection. Figure adopted from [8].

ber to receive data from one another. The order of the write/read is arbitrary but needs to be reversed with respect to each other due to blocking the thread when executing a read. When sending data, the sender has no knowledge about ”who” its sending to. This means that the appropriate time to close the sockets needs to be implemented by the programmer, in contrast to TCP where a socket closes when the client disconnects [14].

3.1.3 Process

Several definitions of the term *process* is suggested by [16], some of which are:

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to - and executed on a processor

A process can get *Resource ownership*, meaning it will sometimes be allocated protected ownership/control of resources such as main memory, I/O channels, devices and files. A process has its own virtual address space to hold the process image, where the process image consists of the program, data and stack and a few other attributes. A process state can be described by the models: ”Two-state model” and ”Five-state model”. The former is

the simplest version where a process can either be ”Running” or ”Not running”. The latter is more complex and consists of

- Running
- Ready
- Blocked/Waiting
- New
- Exit

The Kernel, which handles all resource management and hardware interaction in an OS, decides which state to put a process in [16].

3.1.4 Thread

A process contains no less than one thread. Because a process can contain more than one thread, threads are often referred to as **lightweight processes**. Much like a process, a thread is given execution time by the operating system (OS). A thread can only access memory contained in the process' virtual address space. Once a thread is spawned it can have the following states, similar to a process [16]:

- Running
- Ready
- Blocked

Consider a processor with only one core and a process with one thread. Then that thread will get all the execution time by the OS. Now, if the process contains two threads (multi-threading), the OS (thread library) will context switch really fast between execution instructions in both threads, executing them concurrently. Priority and timing options can be added to give one thread more execution time. This is useful for displaying, e.g. a video stream while handling asynchronous button presses in an application. In a multi-threading application, available memory can accessed by multiple threads concurrently. Worst case, a number can be read, while it is changed, resulting in reading a wrong number. This is called a *race condition* (result depends on timing of threads). This must be handled with thread *synchronization* using e.g. a ”Mutex” (Mutual exclusion)[16]. Just before a thread is accessing a shared/global variable the thread ”locks” the resource using the ”Mutex”. All other threads that tries to access this variable are blocked until the accessing thread calls ”unlock”.

3.1.5 Robot Operating System

Developed by Eric Berger and Keenan Wyrobek during their PhDs at Standford, ROS is meant to help researchers and engineers to focus on invention instead of re-invention [17]. Obtaining details of published paper's software was difficult and about 90 % of the time

was spent re-writing source code and about 10 % of the time was used for innovation [18]. The solution was to get funded and hire software engineers to develop critical ”plumbing” software and developer tools that enabled innovators in robotics to build on each other's progress. ROS now has a long list of drivers and tools for common software and hardware used in robotics, e.g. GPS, Inetial Measruement Unit (IMU), motor controllers etc, compatible with their software.

3.1.6 Nodes in ROS

ROS projects consists of nodes. Each node is essentially a *process*, giving it the properties described in 3.1.3. A node performs a task, either periodically or asynchronously [19]. Nodes in ROS projects can consist of either C++ or Python code.

A node's task can be as simple as setting an input to a DC-motor or it can perform complex algorithms. A node usually produce some kind of result. This result is often needed by another node in the project. The producing node can then ”publish” the result to a **topic** (see 3.1.7) making it available for all the other nodes. Nodes that need the result can just ”subscribe” to that topic to receive the message containing the data when it is available. This enables data sharing between processes (interprocess communication), despite separate virtual memory spaces.

3.1.7 Topics in ROS

Topics in ROS are the communication buses that nodes use to communicate (send their results). They transmit **messages** (3.1.8) containing the results they generate. In ROS the *producer/consumer* semantics are referred to as publisher and subscriber. The publisher generates a result and publishes it. It does, however, not know which nodes receives it. Likewise, a node subscribing to a topic does not know which node sent it [20]. A node can publish and subscribe to multiple topics.

3.1.8 Messages in ROS

Messages are data structures of primitive types such as, integer, string, float and boolean etc. Custom messages can be created from a combination of the primitive types and arrays. Variables can then be named improve readability, e.g. `Float64 velocity` instead of the default `Float64 data` [21].

3.1.9 Rosbag Data

Rosbag is a command-line tool for logging *Messages* published to *Topics* in ROS. Files are stored in a .bag-format of which can be imported to Matlab with the *Robotics System Toolbox*.

3.1.10 CartoType Navigation Framework

CartoType provides a framework primarily for developing offline applications using detailed navigation maps. Open source maps obtained from sources such as `http://openstreetmap.org` can be converted to supported formats(.ctm1) and integrated into an application written in C++. The framework implements methods for geo-positioning and street navigation. Platforms supported are Android, iOS, Linux, Mac OS, .NET and Windows. The software development kit (SDK) can be downloaded at [22]. Source code for public use can be found at [23].

3.2 RMC Station Introduction

The RMC station is developed for remote monitoring and control of ReVolt in the specialization project using Qt Creator and runs on Linux. A class diagram of the RMC station software is shown Figure B.1 of Appendix B.

The RMC station software run as a single process (see Section 3.1.3) making use of a threading library to handle synchronous and asynchronous events, while preventing *race conditions* with different types of Mutex (for more info on threads/threading see Section 3.1.4).

To enable transmission of data between the RMC station and ReVolt, the connection-oriented TCP transport protocol, described in Section 3.1.1, is used. This implies that data can be transmitted over Wifi and 4G in a client-server paradigm. A library called *"Practical C++ Sockets"* is used to create communication sockets to transmit and receive data through, in a manner described in Section 3.1.2 where the RMC station acts as the client. This data includes, but are not limited to, a live image stream (UDP) from a camera mounted on ReVolt, navigational information, system data and remote control signals.

The image stream shown in Figures 3.4 and 3.6 uses the best-effort UDP transport protocol due to requiring a higher bandwidth than the other data transmitted. The *MUVI K2 Sport* camera publishes raw images at rate of 10 Hz which are compressed using JPEG-compression algorithm, described more in [8]. The resulting image stream has a 720p resolution.

A navigation map is also implemented such that basic interaction and displaying of ReVolt's location geographically is possible (see [8] for creation and implementation of the navigation map for ReVolt). ReVolt can be controlled from the RMC station in *heading autopilot mode*, by setting a heading reference and thruster effort. Alternatively, in *dynamic positioning mode*, by setting a desired position and heading.

The RMC station is, however, never tested at sea in the specialization project and all software written solely in C++.

Figure 3.4 shows the RMC station's Graphical User Interface (GUI). The red rectangles and numbering correspond to different data:

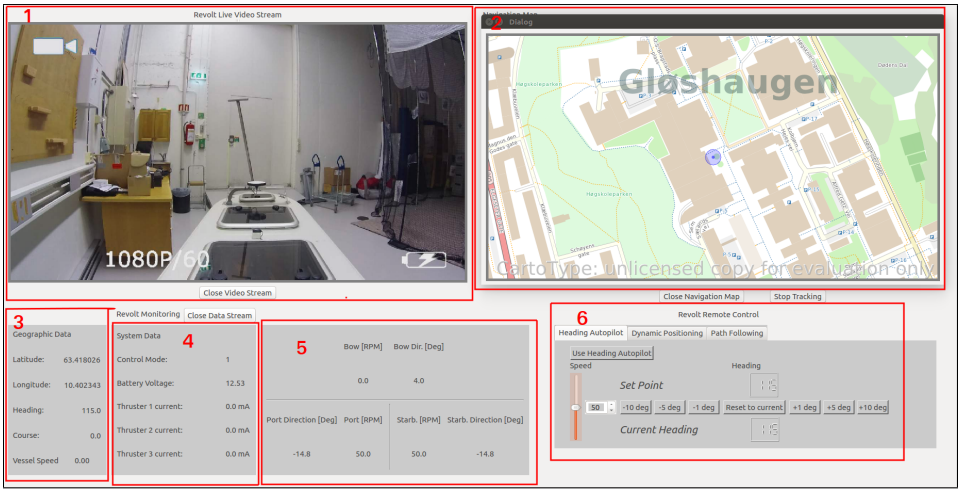


Figure 3.4: Screenshot of the RMC station during development, fall of 2017. Note that the "1080p/60" watermark is not the actual resolution of the received image stream.

1. Live image stream from ReVolt
2. Interactive navigation map with live view of ReVolt's position and footprint
3. ReVolt's navigational data
4. ReVolt's system state
5. ReVolt's actuator states
6. Remote Control of ReVolt (Heading Autopilot, DP, LOS Guidance)

3.3 RMC Station Contributions

The Guidance Management (*"Autonomous Control"* in Figure 3.5) is a new contribution to the RMC station in this thesis. This allows the operator to place a desired number of waypoints (shown in Figure 3.5 and Figure 3.6 as red filled squares with numbering) and generate a path (shown as black dashed lines) between each consecutive pair. Once the operator has placed all desired waypoints, clicking "Execute Route" transmits the list containing all waypoint pairs to ReVolt via Wi-Fi/4G. ReVolt uses its Guidance System, developed in this thesis, to track the path, leaving a green footprint. The development and implementation of the GNC system for path following is presented in Chapter 4.

Figure 3.5: LOS Guidance in the RMC station tested on DNV GLs simulator.

3.3.1 Guidance Management

The CartoType Navigation Framework, described in Section 3.1.10, forms the basis for the Guidance Management. The open-source code includes, but are not limited to, event-handlers and utility functions that enables panning, zooming, rotating and dragging the map. By using event-handlers, the pixel-coordinates of a mouse-click can be converted into a geodetic latitude and longitude in the map. For the purpose of waypoints, these positions are drawn in the map and stored for later transmission.

For every event (e.g. zooming, clicking) occurring in the navigation map needs to be re-drawn to display the event's result. The method that draws/updates the map is provided by the CartoType API. However, methods regarding waypoint placement, path drawing, footprint trace and obstacle visualization needs to be integrated in this method. The positions of waypoints, paths, footprint and obstacles are stored in lists because they need to be redrawn every time a map-event occurs. Source code for drawing of waypoints, paths, footprint and obstacle is listed in Appendix B. The entire source code for the RMC station can be made available at the Git repository at *Visual Studio Team Services* (VSTS).

3.3.2 Transmitting List of Waypoints

Using the established TCP connection, the `Transceiver`-class in the RMC station receives the list of waypoints from the `MapForm`-class. The list is parsed into a `String`, with values separated by ":". Adding an identifier "GC:" (Guidance Control) to the start

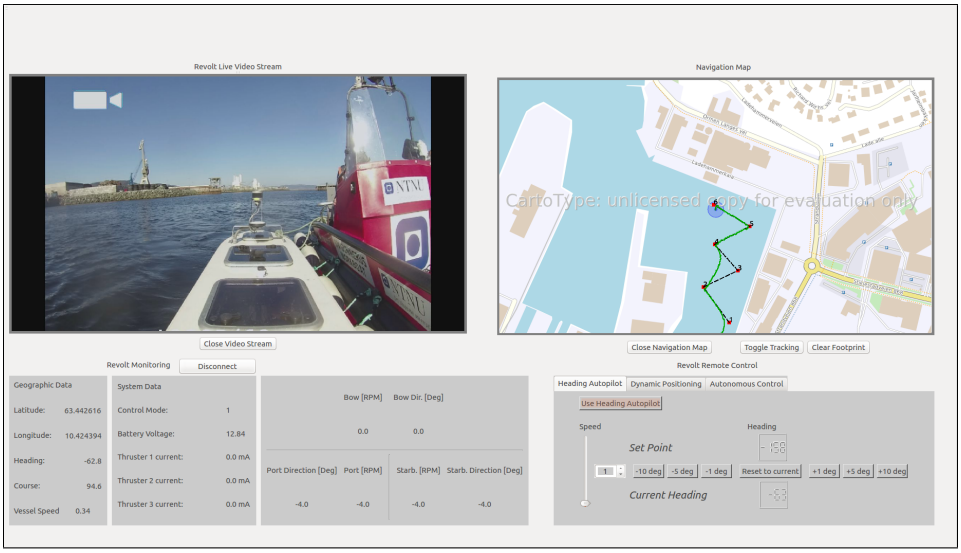


Figure 3.6: LOS Guidance in the RMC station after finishing a run in Dora. A waypoint is skipped due to an implementation error causing too fast iterations though waypoints. Map update was disabled at the end of the run, meaning latitude and longitude in GUI does not match exact position in map at the time of screenshot capture. Also, a setting on the camera caused the different display format.

of the string before it is transmitted, enables ReVolt's `/TCPDatatransceiverNode` (see Section 3.1.6) to know which mode to request (Heading autopilot, DP, Guidance Control) and pass the waypoints to the `/GuidanceLawNode` to generate a path and perform necessary control action.

3.3.3 Image Stream Frame Distortion Removal

An issue from the specialization project regarding the image stream showed that every few seconds the received images would suffer from distortion (see Figure 3.7).

The reason for this being a timing-issue. The solution was to let the `VideoReceiver`-class `emit` a signal every time a full image is received and ready to be displayed, inheriting the rate ($\sim$10 Hz) of the transmission from ReVolt.

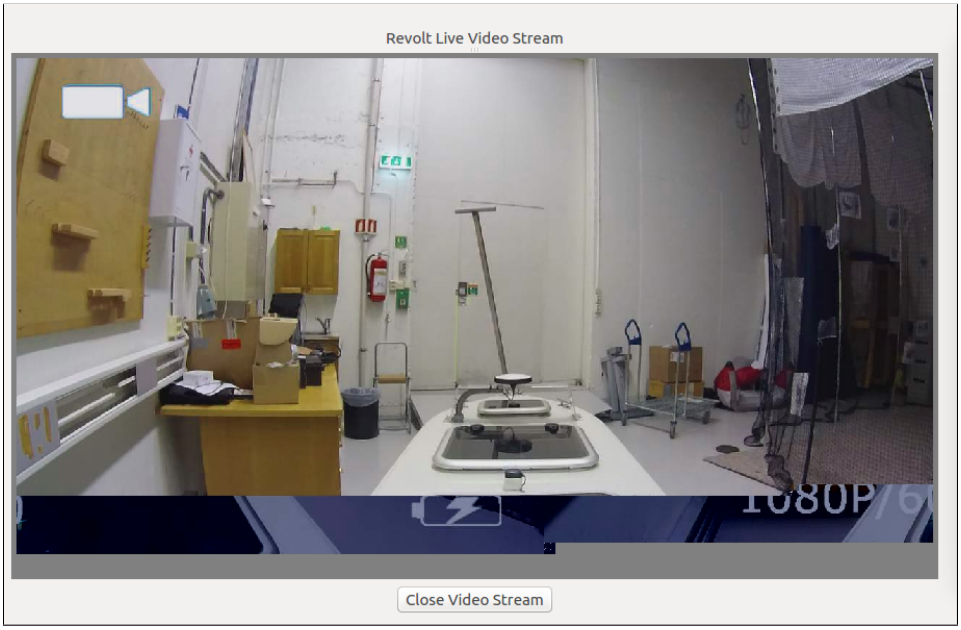


Figure 3.7: Screenshot of image stream distortion [8].

Chapter 4

GNC Implementation for Path Following

In this chapter, a GNC system enabling ReVolt to follow a predefined path, determined by waypoints, is designed and implemented. The waypoints are placed by an operator at the RMC station as described in Chapter 3. In Section 4.1 the GNC system is presented w.r.t the generic structure of Section 2.1. In Section 4.2 the Guidance System is presented. The control allocation for turning and forward speed is presented in Section 4.3. For maintaining forward speed, a surge speed controller is developed and presented in Section 4.4, this includes a feed forward and feedback term. For course-keeping and course-changing maneuvers, a heading controller is developed and presented in Section 4.5. The heading controller also includes a feedforward and feedback term. The system identification in this thesis is done using ReVolt's digital twin (see Section 5.1), developed by DNV GL. Lastly, the implementation in the ROS environment on board ReVolt is described in Section 4.6. A discussion regarding development and implementation is presented in Chapter 7. Simulations and experimental results are presented in Chapter 5 and 6, respectively.

4.1 Guidance, Navigation and Control System

Figure 4.1 shows a block diagram of the implemented GNC System on ReVolt for path following.

The **Guidance System** uses the LOS Guidance, described in Section 2.5 with the *lookahead-based steering* principle described in Section 2.5.1. *Lookahead-based steering* is chosen in favor of *enclosure-based steering* due to less computational requirements [12]. The Guidance system inputs a list of waypoints that make up piece-wise interconnected straight lines expressed in the $\{n\}$-frame. It also inputs navigational data such as the vessel's posi-

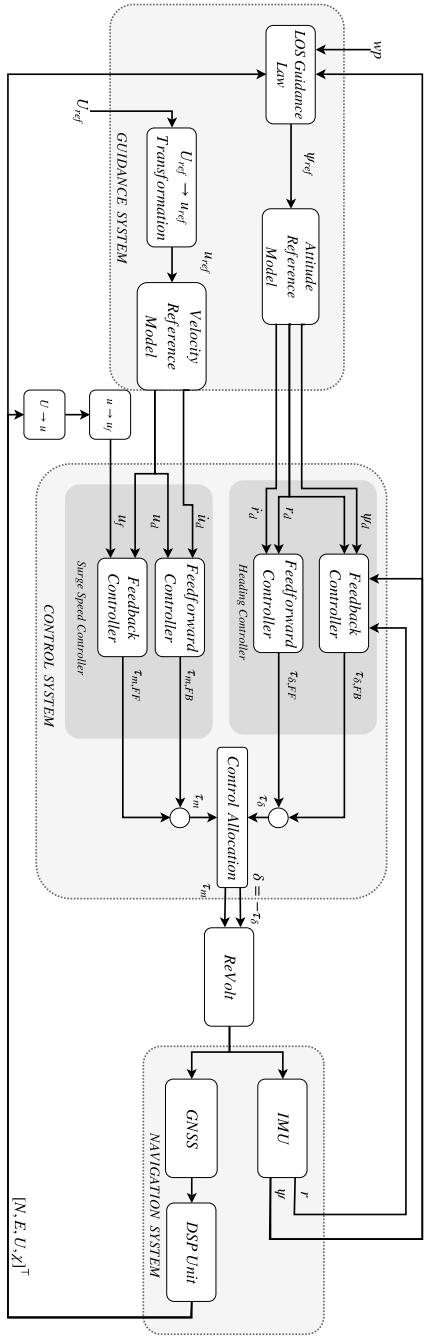


Figure 4.1: Block diagram illustration of the Guidance, Navigation and Control System for path following as implemented on ReVolt.

tion in the {n}-frame, speed over ground, heading and course. It then computes required reference values for the heading and surge speed controllers regulating the cross-track error $e(t)$ to zero. Switching of waypoints is handled using solely the along-track distance in (2.32) as described in Section 2.5.2. Sideslip, caused by environmental disturbances, is corrected using velocity measurements given by the GNSS (see Section 4.2). Reference models for heading and speed controllers smooth the step changes $\psi_{ref} \rightarrow \psi_d$ and $u_{ref} \rightarrow u_d$ and create higher order derivatives for feedforward control. In Figure 4.1 the *Velocity Reference Model* and *Attitude Reference Model* is contained in the Guidance System. In the **implementation** however, the reference models for the controllers exist in their respective controller's source code. This is for practical reasons as controlling ReVolt manually using just the heading controller requires the reference model as well. For the description of the reference models see Sections 4.4.1 (speed) and 4.5.3 (heading).

The **Control System** block contains two low-level controllers, one for controlling the surge speed and one for heading angle. The control objectives are to track the desired values passed by the Guidance System using a model-based feedforward and PID feedback for heading. Similarly, model-based feedforward using desired states and PI feedback for surge speed. The control allocation is configured to use the two aft thrusters constrained to an angle of $\delta_{max/min} = \pm 45^o$.

The **Navigation System** remains the same, and collects the data from the GNSS and IMU which are passed to the steering law and low-level controllers. Wild-point filters are already implemented for the GNSS position measurement by [1].

4.2 Guidance System

The LOS Guidance scheme described in Section 2.5 and the *lookahead-based steering* principle from Section 2.5.1 is used to implement a path following system for ReVolt's control system.

A list containing the geodetic positions (longitude and latitude) of the waypoints is obtained as described in Chapter 3. The positions are transformed to the {n}-frame using an algorithm from [24], resulting in the waypoint list from (2.21). The path-tangential angle a_k is found using (2.22) with the positions of the current waypoint pair and express the marine craft's vessel in a path-fixed coordinate system with origin at p_k^n and x-axis pointing along the path. The along-track distance and cross-track error $(s(t), e(t))$ are the vessels coordinates in that frame.

The **control objective** is to asymptotically regulate the cross-track error $e(t)$ to zero, that is

$$\lim_{t \to \infty} e(t) = 0 \tag{4.1}$$

The course angle assignment from (2.26) is

$$\chi_d(e) = \alpha_k + \chi_r(e) \tag{4.2}$$

where

$$\chi_r(e) := \arctan 2\left(\frac{e}{\Delta}\right) \tag{4.3}$$

is the steering law from (2.28) and $\Delta = \mathbf{9}$ [m] is the lookahead distance. To account for ocean currents, the sideslip-angle β is calculated using (2.30) restated here

$$\beta = \arcsin\left(\frac{v}{U}\right) \tag{4.4}$$

and the desired course χ_d is adjusted to output the LOS heading angle

$$\psi_{los} = \chi_d + \beta \tag{4.5}$$

Desired heading ψ_d and its higher order derivatives is obtained by passing ψ_{los} through the reference model in Section 4.5.3. Switching of waypoints is done using the along-track distance as described in Section 2.5.2 with (2.32) restated here

$$s_{k+1} - s(t) \leq R_{k+1} \tag{4.6}$$

with $R_{k+1} = \mathbf{16}$ [m].

An operator inputs a speed over ground reference speed U_{ref} , while the surge speed controller sets a surge speed reference u_{ref}. A transformation $U_{ref} \rightarrow u_{ref}$ is given by the decomposing the total reference speed with the sideslip angle β

$$u_{ref} = U_{ref}\cos(\beta) \tag{4.7}$$

inserting (4.4) into (4.7) yields

$$u_{ref} = U_{ref}\cos(\arcsin\left(\frac{v}{U}\right)) \tag{4.8}$$

which, when $U \rightarrow U_{ref}$, can be written as

$$u_{ref} = \sqrt{U_{ref}^2 - v^2} \tag{4.9}$$

Note that (4.9) is not implemented for now, i.e. $U > U_{ref}$ is possible. Since v is subtracted before passing through the surge speed reference model, ensuring that the reference model captures the change in v is crucial. Otherwise, v must be subtracted after filtering, causing noise in u_d as no state estimator is implemented. The simulation and experimental results for the GNC system for path following is presented in Sections 5.4 and 6.4, respectively.

4.3 Control Allocation

ReVolt is described more in Section 6.1. However, a short description of propulsion system is presented to understand the control allocation. ReVolt has two freely rotating (azimuth) stern thrusters and a bow thruster constrained to $\pm$ 270^o. DNV GL's thrust allocation

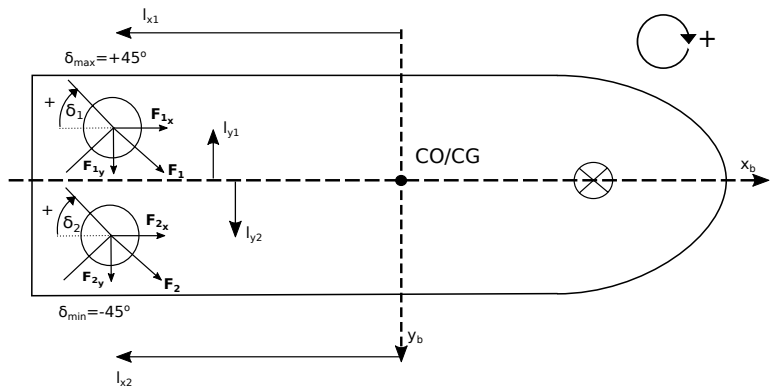


Figure 4.2: Illustration of the setup. The two stern thrusters is used for propulsion and turning

software is already available for in *Dynamic Positioning*. This allows independent control forces and moments in 3-DOF. Regardless, [12], [6], [25] and [26] has shown that a configuration with thrust force and moment in surge and yaw (underactuated) can be used for "high speed" tracking/path following with only a yaw/yaw rate controller and surge speed controller. That is, sway motion is not controlled directly. The Guidance Law outputs a LOS heading ψ_{los} and a speed reference U_{ref} is set by an operator. Therefore, a decoupled thrust effort and thruster angle is used instead of configuring the control system to use DNV GL's control allocation with an underactuated configuration.

A linear relationship between the input value to the thruster and the force produced can be described as [9]

$$\mathbf{F} = \mathbf{K}\mathbf{u} \tag{4.10}$$

where $\mathbf{F} \in R^n$ is the force produced, $\mathbf{K} \in R^{r \times r}$ is the force coefficient matrix and $\boldsymbol{u} \in R^r$ is the control input to the actuator. The generalized control forces $\boldsymbol{\tau} \in R^n$ is described as

$$\boldsymbol{\tau} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{F} \tag{4.11}$$

and inserting (4.10) for $\boldsymbol{F}$ yields

$$\boldsymbol{\tau} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{K}\boldsymbol{u} \tag{4.12}$$

where $\boldsymbol{\alpha}$ is a column vector of azimuth angles and $\boldsymbol{T}(\boldsymbol{\alpha}) \in R^{n \times r}$ is the thrust configuration matrix.

ReVolt's configuration for path following use an identical thruster angle command $\delta \in [-45^o, 45^o]$ for each thruster. That is

$$\boldsymbol{\alpha} = [\delta_1 \;\; \delta_2]^\top \tag{4.13}$$

The the configuration vectors $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ for thruster 1 and 2 could be obtained from

Figure 4.2 as

$$\boldsymbol{T}_1(\delta_1) = \begin{bmatrix} \cos(\delta_1) \\ \sin(\delta_1) \\ -|l_{y_1}|\cos(\delta_1) - |l_{x_1}|\sin(\delta_1) \end{bmatrix} \tag{4.14}$$

$$\boldsymbol{T}_2(\delta_2) = \begin{bmatrix} \cos(\delta_2) \\ \sin(\delta_2) \\ -|l_{y_2}|\cos(\delta_2) - |l_{x_2}|\sin(\delta_2) \end{bmatrix} \tag{4.15}$$

Combining the two column vectors from (4.14) yields the thrust configuration matrix

$$\boldsymbol{T}(\delta_1, \delta_2) = \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_2 \end{bmatrix} = \begin{bmatrix} c(\delta_1) & c(\delta_2) \\ s(\delta_1) & s(\delta_2) \\ |l_{y_1}|c(\delta_1) - |l_{x_1}|s(\delta_1) & -|l_{y_2}|c(\delta_2) - |l_{x_2}|s(\delta_2) \end{bmatrix} \tag{4.16}$$

Both thrusters received identical input effort $\tau_{1,m} = \tau_{2,m} = \tau_m \in [0 \;\; 100]$ with unit [%], such that

$$\boldsymbol{u} = [\tau_m \;\; \tau_m]^\top \tag{4.17}$$

and since the control system also produces the same input angle for both thrusters, $\delta_1 = \delta_2 = \delta$. Due to symmetry $l_{x_1} = l_{x_2} = l_x$, $l_{y_1} = -l_{y_2}$ and $\boldsymbol{K} \to K_m > 0$, the resulting generalized forces and moments $\boldsymbol{\tau}$ from (2.10) is

$$\boldsymbol{\tau} = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} = \begin{bmatrix} 2\cos(\delta) \\ 2\sin(\delta) \\ -2|l_x|\sin(\delta) \end{bmatrix} K_m \tau_m \tag{4.18}$$

The thrust coefficient K_m is found with linear regression with some of the data collected with ReVolt at SINTEF Ocean during a DNV GL summer internship of 2017 (see Appendix D.2). By using `polyfit` in Matlab with the data contained in Table 4.1 a linear

Thruster Effort [%]	25	50	75	100
Force Produced [N]	4	7	13.5	18

Table 4.1: Results from a single test with ReVolt at SINTEF Ocean. Force produced by a single thruster at different efforts.

correlation between effort applied and force produced, is obtained:

$$F(\tau_m) = 0.218\tau_m - 3.5 \tag{4.19}$$

The figure shows the data points from Table 4.1 versus the fitted linear function (4.19) can be seen in Figure 4.3. Inserting (4.19) into (4.18) yields

$$\boldsymbol{\tau} = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} = \begin{bmatrix} 2\cos(\delta) \\ 2\sin(\delta) \\ -2|l_x|\sin(\delta) \end{bmatrix} F(\tau_m) \tag{4.20}$$

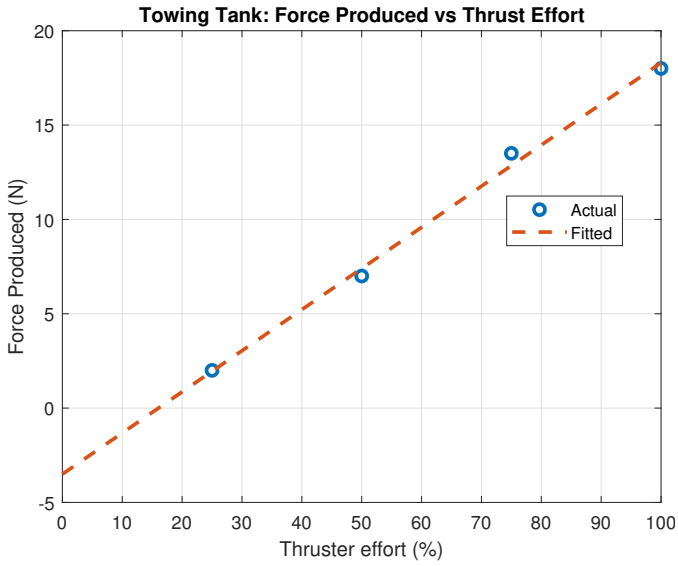


Figure 4.3: Mapping of thruster effort versus the corresponding force produced. Results obtained from SINTEF Ocean.

4.4 Surge Speed Controller

The control objective is to asymptotically track a reference surge speed u_{ref}. This is done using a PI controller with a feedforward term and reference model supplying a desired velocity and acceleration, as seen in the control structure of Figure 4.7. The feed forward term consists of a damping and inertia term. Proportional-Integral Feedback corrects deviations in surge speed caused by environmental disturbances and modeling erros.

4.4.1 Reference Model

To avoid large and unnecessary control efforts from the PI controller during step changes, the second order low-pass filter from (2.11) is implemented to smooth the reference signal in the surge motion. This now takes the form

$$\ddot{u}_d + 2\zeta\omega_n\dot{u}_d + \omega_n^2 u_d = \omega_n^2 u_{ref} \tag{4.21}$$

Using Matlab's *System Identification Toolbox* the values for ζ and ω_n is found. Applying a step corresponding to roughly 1 m/s surge speed and curve fitting a second order generic transfer function to this response yields the necessary coefficients

$$\omega_n = 0.1583 \tag{4.22}$$

$$\zeta = 0.9994 \tag{4.23}$$

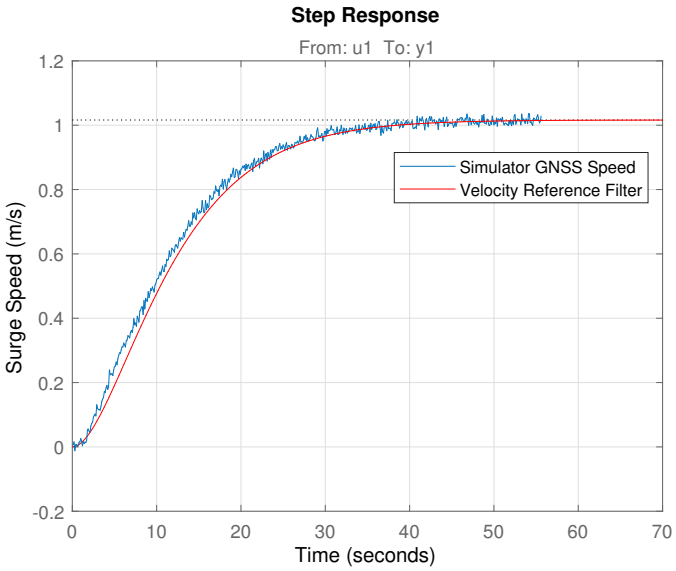


Figure 4.4: Velocity Reference Model 1 m/s step response, without saturation.

with 96.29% accuracy, corresponding to the second order reference filter

$$\ddot{u}_d + 0.3164\dot{u}_d + 0.0251u_d = 0.0251u_{ref} \tag{4.24}$$

In state-space representation this becomes

$$\begin{bmatrix} \dot{u}_d \\ \ddot{u}_d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.0251 & -0.3164 \end{bmatrix} \begin{bmatrix} u_d \\ \dot{u}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 0.0251 \end{bmatrix} u_{ref} \tag{4.25}$$

The reference model also includes saturating elements to limit the desired surge acceleration

$$\dot{u}_d \leq \dot{u}_{max} \tag{4.26}$$

Hence

$$\dot{u}_d \rightarrow \text{sat}(\dot{u}_d) \tag{4.27}$$

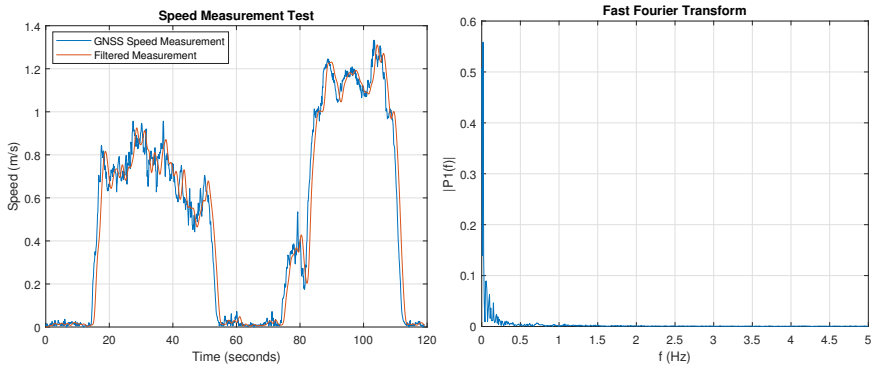
where

$$\text{sat}(x) := \begin{cases} \text{sgn}(x)x_{max} & \text{if } |x| \geq x_{max} \\ x & \text{else} \end{cases} \tag{4.28}$$

Discretization of the filter is done using (2.19) from Section 2.4.3.

4.4.2 Low-Pass Filtering of Velocity Measurement

A 5.order Infinite Impulse Response (IIR) low-pass filter is implemented to smooth the surge speed measurement and suppress some noise from entering the control loop. A IIR-filter is chosen in favor of Finite Impulse Response (FIR)-filter due to it requiring less



(a) Time domain (t) **(b)** Frequency domain (f)

Figure 4.5: (a) GNSS speed measurement versus a low-pass filtered version of the signal. (b) The Fast Fourier Transform (FFT) identifies which frequencies occurs in the signal. The test was done by dragging ReVolt on its carrier/wagon outside and logging the speed.

computing power [27]. The equation for the filter is given by

$$y_k = \frac{1}{a_0}\left(b_0 x_k + b_1 x_{k-1} + \ldots + b_M x_{k-M} - a_1 y_{k-1} - \ldots - a_N y_{k-N}\right) \tag{4.29}$$

where b_i and a_i are the $(M+1) = \mathbf{6}$ and $N = \mathbf{5}$ filter coefficients, respectively. Here $y_k = u_f(t)$ is the filtered surge speed output of the filter at time step k. $x_k = u(t)$ is the input surge speed at time step k. Note that all y_k and x_k prior to time step 0 is equal to 0. The filter coefficients are listen in Appendix D.3.

Figure 4.5a shows the measured speed versus the filter speed. Note the slight delay in the filtered signal. This is due to the filter time constant, and the downside of using a low-pass filter as opposed to a state estimator. Figure 4.5b shows the occurrence of the different frequencies in the measured signal. These were obtained by using Fast Fourier Transform (FFT) function in Matlab `fft`. By inspecting frequencies in the FFT, a filter can be designed using Matlab's `FilterDesigner` with parameters listed in Table 4.2 and selecting the minimal order necessary to realize the filter.

Parameter	Value
F_s	10 Hz
F_{pass}	0.5 Hz
F_{stop}	2 Hz
A_{pass}	1 dB
A_{stop}	60 dB

Table 4.2: Parameters used in the speed filter

Note that having a small difference between F_{stop} and F_{pass} or large difference between A_{stop} and A_{pass} increases the order of the filter (number of coefficients) which increases

the complexity of the implementation in practice. A compromise is necessary to filter noise with minimal delay.

4.4.3 Control Objective

The surge error is defined as

$$\tilde{u}(t) \triangleq u_d(t) - u_f(t) \tag{4.30}$$

where $u_d(t)$ is the time-varying, desired surge speed supplied by the reference filter in Section 4.4.1 and $u_f(t)$ is the low-pass filtered velocity measurement from Section 4.4.2. The control objective is to minimize $\tilde{u}(t)$ such that

$$\lim_{t\to\infty} \tilde{u}(t) = 0 \tag{4.31}$$

in the presence of wind and ocean currents. Wave filtering is not part of this thesis due to rare occurrence of waves in test area. The control law (output of controller) is formulated as

$$\tau_m = \tau_{m,FF} + \tau_{m,FB} \tag{4.32}$$

where $\tau_{m,FF} \in [0 \;\; 100]^\top$ [%] is the feedforward term that ensures a forward speed trajectory and $\tau_{m,FB} \in [0 \;\; 100]^\top$ [%] is the feedback term that corrects errors caused by environmental disturbances and modeling uncertainties.

4.4.4 Feedforward Term

The identification of the feed forward term is obtained with the Digital Twin and consists of an acceleration and velocity feedforward

$$\tau_{m,FF} = M\dot{u}_d + \sigma(u_d) \tag{4.33}$$

where $M\dot{u}_d$ is the inertia term and $\sigma(u_d)$ is the steady-state polynomial damping term.

The damping term $\sigma(u_d)$ is calculated by applying a series of steps to the stern thrusters and recording the steady state surge speeds for each step. These are listed in table 4.3.

Thruster Effort [%]	10	20	30	40	50	60	70	80	90	100
Surge Speed [m/s]	0.42	0.60	0.74	0.86	0.96	1.07	1.16	1.24	1.32	1.40

Table 4.3: Effort applied to the stern thrusters and the corresponding steady state surge speed. Results obtained with the Digital Twin in simulator.

Using `polyfit` in Matlab, with the two rows of table 4.3 as vectors, the following second order polynomial mapping function is obtained (see Figure 4.6):

$$\sigma(u_d) = 46.9590u_d^2 + 6.3054u_d - 0.3211 \tag{4.34}$$

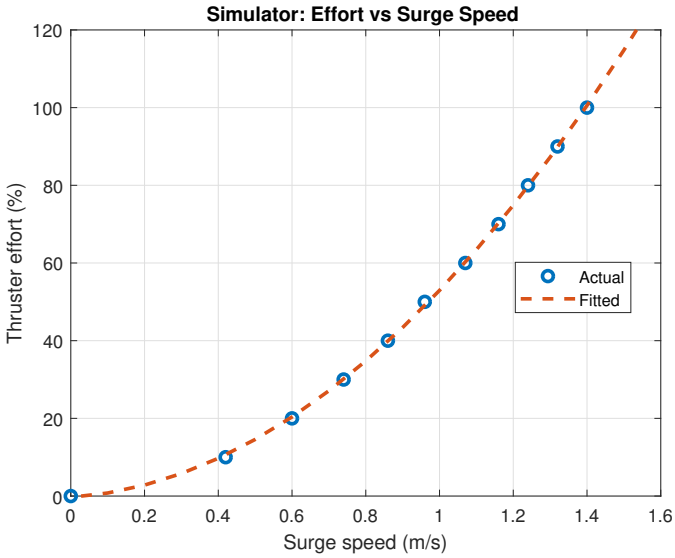


Figure 4.6: Second order mapping function using values from table 4.3

In (4.34), the desired velocity u_d is mapped into an effort and applied to the thrusters as part of the feed forward. With ReVolt in practice, a thruster effort lower that approximately 15% is not enough to overcome the friction in the bearings. Further processing of the polynomial (see [25]) is not part of this thesis.

M in the inertia term is found by step response simulations using $\tau_{m,FF}$ only. $M =$ **800** gives sufficient results (see Figure 5.12) The total feedforward control effort becomes

$$\tau_{m,FF} = 800u_d + 46.9590u_d^2 + 6.3054u_d - 0.3211 \tag{4.35}$$

4.4.5 Feedback Term

A PI controller is chosen as feedback due to its simplicity and robustness. The time-continuous PI controller for the speed controller takes the form

$$\tau_{m,FB} = K_p\tilde{u}(t) + K_i \int_0^t \tilde{u}(\tau)d\tau \tag{4.36}$$

where $\tau_{m,FB} \in [0, 100]$ is the feedback effort, $\tilde{u}(t) = u_d - u_f(t)$, $K_p > 0$ and $K_i > 0$ are the proportional and integral gain, respectively.

To implement the PI controller a discretization is necessary, this takes the form

$$\tau_{m,FB}[k] = K_p\tilde{u}[k] + K_iT_s \sum_{k=0}^{n} \tilde{u}[k] \tag{4.37}$$

where k is the current time step and T_s is the sample time.

4.4.6 Combined Feedforward and Feedback

Combining (4.35) and (4.37) the total control effort which yields

$$\tau_m = \tau_{m,FF} + \tau_{m,FB} \tag{4.38}$$

$$\tau_m = M\dot{u}_d + \sigma(u_d) + K_p\tilde{u}(t) + K_i \int_0^t \tilde{u}(\tau)d\tau \tag{4.39}$$

where $\tau_m \in [0, 100]$.

A block diagram of the surge speed controller including saturation elements is shown in Figure 4.7. Simulations results for the surge speed controller are presented in Section 5.3.

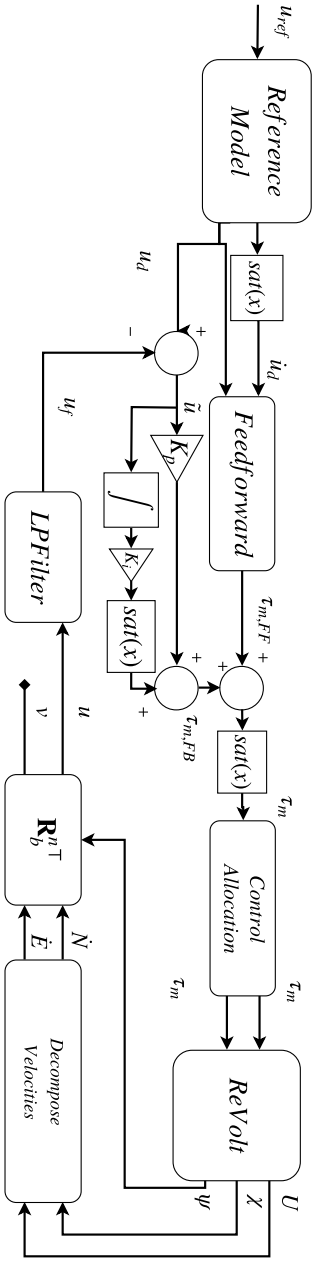


Figure 4.7: Speed Controller block diagram: Reference Filter, feed forward and PI-feedback

4.5 Heading Controller

The heading controller in this thesis is a new contribution to the control system. The previous heading controller consisted solely of PD controller with heading angle feedback and low-pass filtering of the error. Due to noisy control output from the controller (see Appendix D.1), a new one is desired. The new controller uses 1.order Nomoto model for pole-placement of feedback and feedforward computation and receives feedback from both heading angle and yaw rate. A 3.order reference model is implemented to compute desired heading and higher order derivatives (desired yaw rate and yaw acceleration). The simulations and experimental results are presented in Sections 5.2 and 6.3, respectively.

4.5.1 Nomoto Models

The heading controller is based on the 1.order Nomoto model, which is obtained from the 2.order model given by [9]:

$$\frac{r}{\delta}(s) = \frac{K(1+T_3 s)}{(1+T_1 s)(1+T_2 s)} \tag{4.40}$$

Setting

$$T := T_1 + T_2 - T_3 \tag{4.41}$$

yields the 1.order model

$$\frac{r}{\delta}(s) = \frac{K}{1+Ts} \tag{4.42}$$

The models (4.40) and (4.42) describes yaw rate r response due to change in rudder angle (in this case thruster angle command δ) and has decoupled sway-yaw motion. They are are derived from the *Yaw subsystem [9]*

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{N}(u_0)\boldsymbol{\nu} = \boldsymbol{b}\delta \tag{4.43}$$

which has coupled surge-sway motions and assumes constant surge speed u_0 and $\boldsymbol{\nu} = [v \;\; \psi]^\top$. In (4.40), (4.42) and (4.43), the heading angle is a pure integrator of yaw rate, i.e. $\dot{\psi} = r$.

4.5.2 Choosing Nomoto Gain and Time Constant

To determine the gain and time constant K and T in (4.42), Matlab's *System identification Toolbox* is used. By inputting one of the yaw rate responses r in Figure 4.8, with the corresponding thruster angle δ. Selecting a desired number of zeros and poles (e.g. 0 and 1 for first order model) yields a transfer function containing the parameters for the selected yaw rate response.

The parameters for the first and second order models in Figures 4.9a and 4.9b are listed in Table 4.4

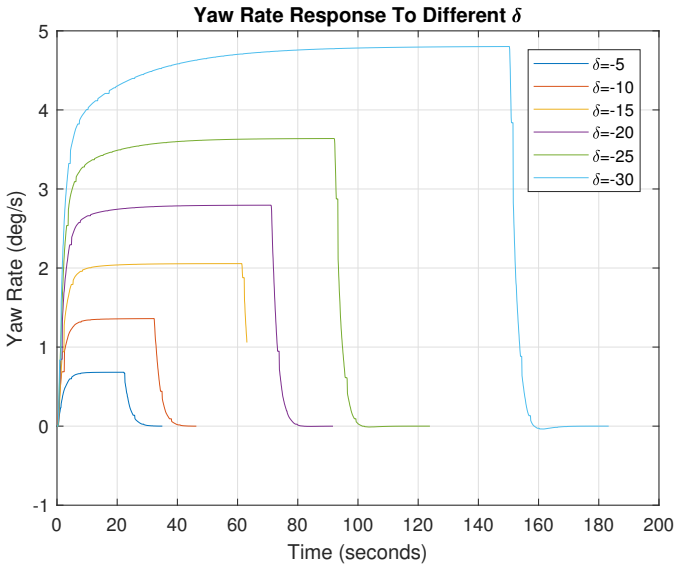


Figure 4.8: Yaw rate responses for different thruster angle commands δ. Thruster effort is set to $\tau_m = \mathbf{60}$ [%], corresponding to a constant forward speed of $u_0 = \mathbf{1.07}$ [m/s]. Response is obtained from DNV GL's ReVolt Simulator with the Digital Twin. Premature stop in logging caused drop in yaw rate at $\delta = -15^o$.

1. order $\delta = -5^o$	1. order $\delta = -15^o$
$K = -0.136174$	$K = -0.137094$
$T = 2.117039$	$T = 2.445675$

Table 4.4: First and second order Nomoto parameters for steps of $\delta = -5^o$ and $\delta = -15^o$ thruster angle commands.

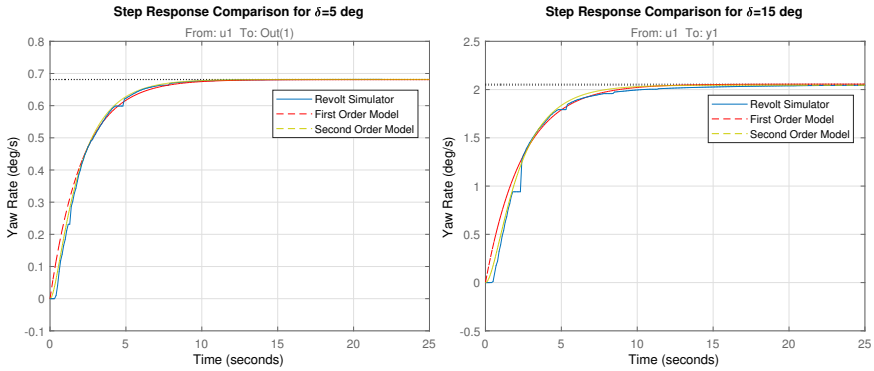
For control design, parameters for the 1.order transfer function with a step of $\delta = -5^o$ is chosen, such that (4.42) yields

$$\frac{r}{\delta}(s) = \frac{-0.1361}{1 + 2.1170s} \tag{4.44}$$

4.5.3 Reference Model

The 3.order reference filter described in Section 2.4.2, is used to compute the desired states ψ_d, r_d and $\dot{r}_d$ needed for turning. The reference model for the heading reference signal is as follows

$$\frac{\psi_d}{\psi_{ref}}(s) = \frac{\omega_n^3}{(s + \omega_n)(s^2 + 2\zeta\omega_n s + \omega_n^2)} \tag{4.45}$$



(a) $\delta = -5^o$ step input. **(b)** $\delta = -15^o$ step input.

Figure 4.9: Comparison between first and second order Nomoto model and ReVolt Simulator for two different thruster angle commands. Note that the ReVolt simulator yaw rate drops out at e.g. $t \approx 2$ in Figure 4.9b, this is due to communication issues between the simulator and ReVolt control system. The yaw rate in the simulator was observed to be smooth.

where ψ_{ref} is the operator input (or ψ_{los} for LOS guidance), ζ is the damping ratio and ω_n is the natural frequency. From (4.45) it can be seen that for a constant ψ_{ref}

$$\lim_{t\to\infty} \psi_d(t) = \psi_{ref} \tag{4.46}$$

and that $\dot{\psi}_d = r_d$ and $\ddot{\psi}_d = \dot{r}_d = a_d$ is smooth and bounded for steps in ψ_{ref} [9]. A saturating element is added to limit the desired yaw rate $|r_d| \leq r_{max}$ [rad/s]. An additional saturation is added to limit the desired yaw acceleration $|a_d| \leq a_{max}$ [rad/s^2] as well. The saturating element is defined as

$$\text{sat}(x) := \begin{cases} \text{sgn}(x)x_{max} & \text{if } |x| \geq x_{max} \\ x & \text{else} \end{cases} \tag{4.47}$$

The state space equation for the third order reference filter for from input ψ_{ref} to ψ_d is as stated in [9]

$$\dot{\psi}_d = \text{sat}(r_d) \tag{4.48}$$
$$\dot{r}_d = \text{sat}(a_d) \tag{4.49}$$
$$\dot{a}_d = -(2\zeta + 1)\omega_n \text{sat}(a_d) - (2\zeta + 1)\omega_n^2 \text{sat}(r_d) + \omega_n^3(\psi_{ref} - \psi_d) \tag{4.50}$$

choosing $\zeta = 1$ yields a critically damped system, leaving ω_n as the tuning parameter. All parameters for the reference model are listen in Table 4.5

Mapping from $\langle -\pi, \pi \rangle$ to $\langle -\infty, \infty \rangle$: The reference signals from the operator at the RMC station and the LOS Guidance algorithm is discontinuous in the $\langle -\pi, \pi \rangle$ junction. This causes unwanted behavior from the reference model and inherently the vessel. As the reference signal passes the discontinuity from e.g. positive π, the reference will then be

Reference Model Parameters	
ω_n	0.60
ζ	1.00
r_{max}	3
a_{max}	0.5

Table 4.5: Parameters used in the reference model

$> -\pi$, causing the reference filter to "reverse" back to towards $-\pi$ through the origin. The problem is solved by the use of a mapping-algorithm from [5].

Mapping from $\langle -\infty, \infty \rangle$ to $\langle -\pi, \pi \rangle$: For use in the heading controller the desired heading angle output by the reference model needs to be mapped back to $\langle -\pi, \pi \rangle$. This is also done using a mapping algorithm from [5].

4.5.4 Control Objective

Error states for heading and yaw rate is defined as

$$\tilde{\psi} \triangleq \psi_d - \psi \tag{4.51}$$

$$\tilde{r} \triangleq r_d - r \tag{4.52}$$

where ψ_d is the desired heading and r_d is the desired yaw rate. These are time-varying and supplied by the reference model. ψ and r are the heading angle and yaw rate measured by the IMU. The controller needs to ensure that

$$\lim_{t \to \infty} \tilde{\psi} = 0 \tag{4.53}$$

$$\lim_{t \to \infty} \tilde{r} = 0 \tag{4.54}$$

and work in the presence of wind and current. A wave filter is omitted from this thesis as wave forces are generally weak in the test area. The control law is formulated as

$$\tau_\delta = \tau_{\delta,FF} + \tau_{\delta,FB} \tag{4.55}$$

consisting of a feedforward (FF) term, to ensure better tracking during course-changing maneuvers. The feedback (FB) term corrects errors caused by model uncertainties and environmental disturbances.

4.5.5 Feedforward Term

To obtain better tracking performance during course-changing maneuvers, a feedforward term is proposed by [9]. The term is obtained from the basis of (4.42), which in the time domain is written as

$$\tau_{\delta,FF} = \frac{T}{K}\left(\dot{r}_d + \frac{1}{T} r_d\right) \tag{4.56}$$

where r_d is the desired yaw rate from the reference model.

Considering the 1. order Nomoto model with an unknown bias b

$$T\dot{r} + r = K\delta + b \tag{4.57}$$

where $\dot{b} \approx 0$ and $\tau_{\delta,FF} = \delta$. Inserting for δ yields

$$T\dot{r} + r = K\frac{T}{K}\left(\dot{r}_d + \frac{1}{T}r_d\right) + b \tag{4.58}$$

and it follows that

$$-b = T\dot{\tilde{r}} + \tilde{r} \tag{4.59}$$

At steady state $\dot{\tilde{r}} = 0 \rightarrow r = -b$, i.e. a steady state error is present if $b \neq 0$.

4.5.6 Feedback Term

To counteract the unknown bias b a PID feedback is implemented as

$$\tau_{\delta,FB} = -\left(K_p\tilde{\psi}(t) + K_i\int_0^t \tilde{\psi}(\tau)d\tau + K_d\tilde{r}(t)\right) \tag{4.60}$$

the values for K_p, K_i and K_d is chosen from the following algorithm with ω_n and ζ is chosen as in Table 4.5 where

Algorithm 1 Pole-placement algorithm
1: Specify the bandwidth $\omega_b > 0$ and relative damping ratio $\zeta > 0$
2: Compute the natural frequency $\omega_n = \frac{\omega_b}{\sqrt{1-2\zeta^2+\sqrt{4\zeta^4-4\zeta^2+2}}}$
3: Compute the P gain: $K_p = m\omega_n^2$
4: Compute the D gain: $K_d = 2\zeta\omega_n m - d$
5: Compute the I gain: $K_i = \frac{\omega}{10}K_p$

Table 4.6: Algorithm for pole-placement [9].

$$m = \frac{T}{K}, \quad d = \frac{1}{K} \tag{4.61}$$

In [9] its stated that the bandwidth ω_b should be around 0.01 rad/s for larger vessels and 0.1 rad/s for smaller vessels. As ReVolt is probably even smaller, further increase in bandwidth could be desirable. Using $\omega_n = 0.6$ sets the bandwidth at about $\omega_b \approx 0.64 \times \omega_n = 0.384$ rad/s. The resulting controller gains are listed in Table 4.7

4.5.7 Combined Feedforward and Feedback

Using a combined feedforward feedback controller results in the control law:

$$\tau_\delta = \frac{T}{K}\left(\dot{r}_d + \frac{1}{T}r_d\right) - \left(K_p\tilde{\psi}(t) + K_i\int_0^t \tilde{\psi}(\tau)d\tau + K_d\tilde{r}(t)\right) \tag{4.62}$$

Controller Gains	
K_p	-6.422
K_i	-0.385
K_d	-14.113

Table 4.7: Controller gains for feedback control

illustrated in a block diagram in Figure 4.10 (next page). In ReVolt the control law is "flipped" before its applied to the actuators, i.e. $\delta = -\tau_\delta$. A block diagram of the heading controller is shown in Figure 4.10. Simulation and experimental results for the heading controller are presented in Sections 5.2 and 6.3, respectively.

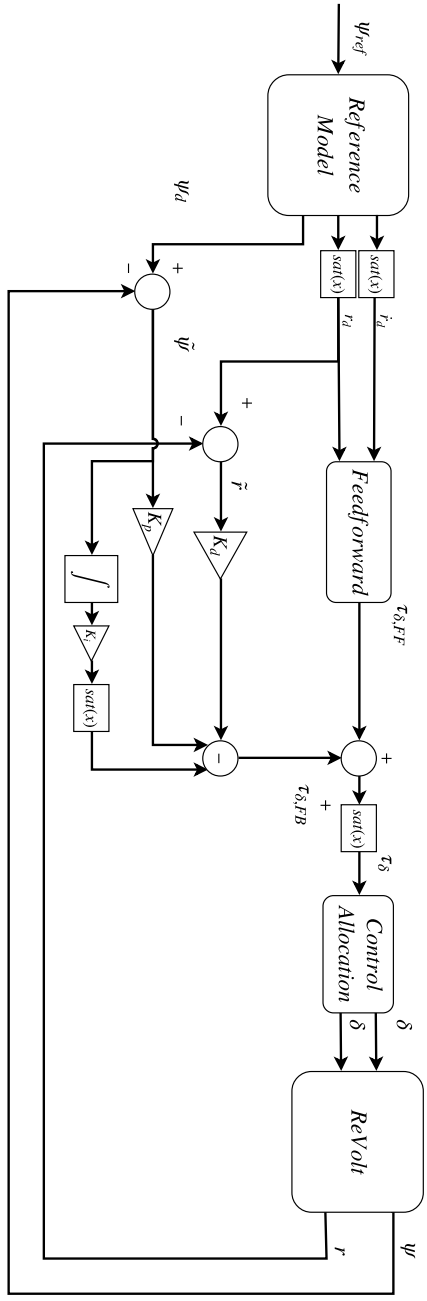


Figure 4.10: Block diagram of the heading controller as implemented on ReVolt.

4.6 Implementation in ROS Environment

Figure 4.11 shows the relationship between the `GuidanceLawNode` and the main node, `ControllerNode` in a custom made ROS graph with most nodes omitted. The latter handles the central logic of the control system, i.e. which mode to select (DP, path following, emergency stop etc.). It subscribes to topics containing necessary sensor data to perform each of the tasks expected from the active mode and publishes the required state and/or setpoints to their respective controllers.

The `GuidanceLawNode` is written in C++ and contains the algorithm for *lookahead-based steering*. It subscribes to the topic `guidance_law_input`, published by the main node at a controlled rate of 10 Hz. The topic has a message containing the state-variables for heading ψ, north position N, east position E, speed over ground U and course over ground χ. Every time this message is received, a callback function executes the algorithm with results depending on the contents of the message (vessel state) and also based on the topic containing the list of waypoints `list_of_waypoints`. The list of waypoints is received from the navigation map in the RMC station through a TCP connection. The `GuidanceLawNode` then publishes reference values for the heading and speed controller through `heading_controller_input` and `speed_controller_input`.

`SpeedControllerNode` and `HeadingControllerNode` responds to these topics by performing the corresponding callback functions containing reference models, feedforward and feedback control algorithm for both controllers. Matrix and vector arithmetics is performed using `Eigen` library. For an excerpt from the source code to the controllers and guidance law see Appendix A.

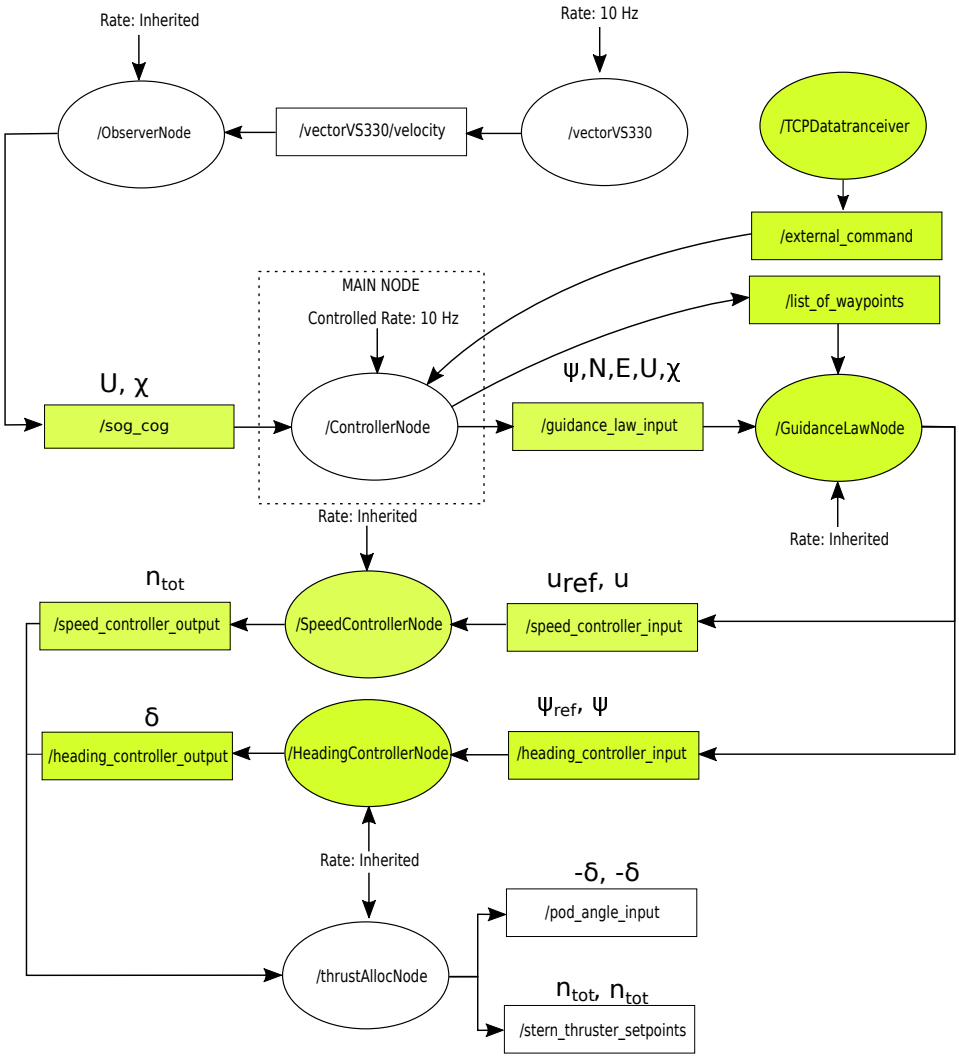


Figure 4.11: Guidance, Navigation and Control in ROS environment for Path Following as implemented onboard ReVolt. Colored nodes (ellipses) and topics (rectangles) are new to the control system. /TCPDataTransceiver was added during the specialization project Fall 2017. However, new functionality to support Guidance Management in the RMC station is added in this thesis.

Chapter 5

Simulation Results

5.1 Simulation Platform

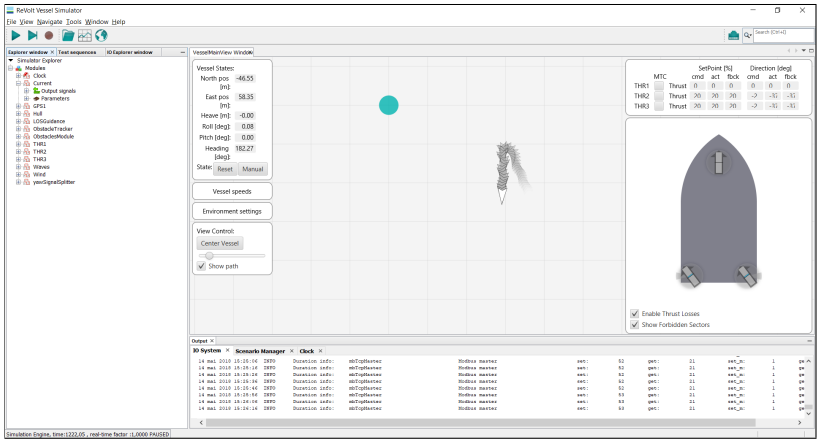


Figure 5.1: Screenshot of DNV GL's ReVolt Vessel Simulator.

All simulation results is obtained through the use the actual ReVolt control system in connection with DNV GL's ReVolt Vessel Simulator (see Figure 5.1). This is a program for Windows that contains a model of the ReVolt and have been updated with the data from the Towing tank results performed with ReVolt at SINTEF Ocean during the summer of 2017 which the author helped produce. The simulator contains simulated modules for the HULL, IMU, GPS, thruster configuration and much more which allows testing of the ReVolt control system without needing the actual HULL, IMU, GPS or thrusters. A preset of environmental disturbances can also be added. Since the simulator communicates with

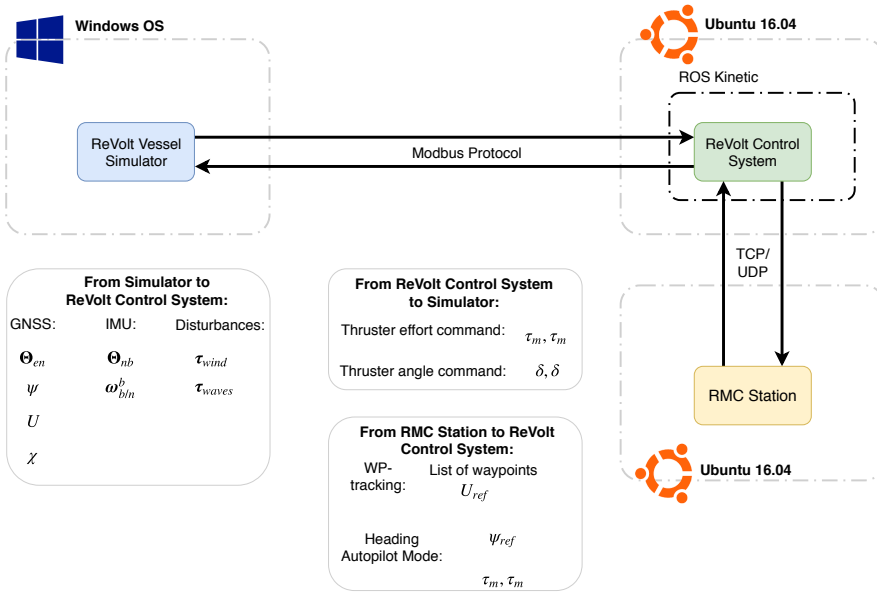


Figure 5.2: Illustration of setup for performing the simulations. Simulator and ReVolt can transmit/receive more data than shown in the figure.

the actual control system, all simulations must run in real-time. Communication is done through the Modbus protocol as the control system runs on another computer using Ubuntu 16.04. Data is logged using *rosbag* tool in ROS, described briefly in Section 3.1.9. An illustration of the setup for obtaining simulation results is shown in Figure 5.2.

An important thing to note with the simulator is that the during its development it was observed that the vessel model, Digital Twin, was observed to be unstable in yaw for $\delta = 0^o$ at any speeds. Similar tests with the physical vessel had not been performed so it was assumed that the instability was due to the Digital Twin's small dimensions (the models are actually meant for larger vessels). A virtual rudder was added to make it stable. Later, it was discovered that the physical model was unstable as well. However, the virtual rudder made system identification in this thesis possible.

5.2 Heading Controller Performance

The tracking performance of the heading controller developed in Section 4.5 is presented here. Three different simulations is performed

1. Feedforward control only (no disturbances)
2. Feedback control only (no disturbances)
3. Combined Feedforward and Feedback control with varying winds up to 2 m/s

For all heading controller simulations, the steps are 10^o, 45^o and 90^o, returning to 0^o between each step. Resulting in a total of six steps. The section contains plots of the tracking performances for the heading and the corresponding deviation from the desired state. Furthermore, tracking performances for the desired yaw rate is also present, with a corresponding deviation from the desired state. Lastly, there exists plots of the control inputs that resulted in the tracking performances in question.

5.2.1 Feedforward Control Only

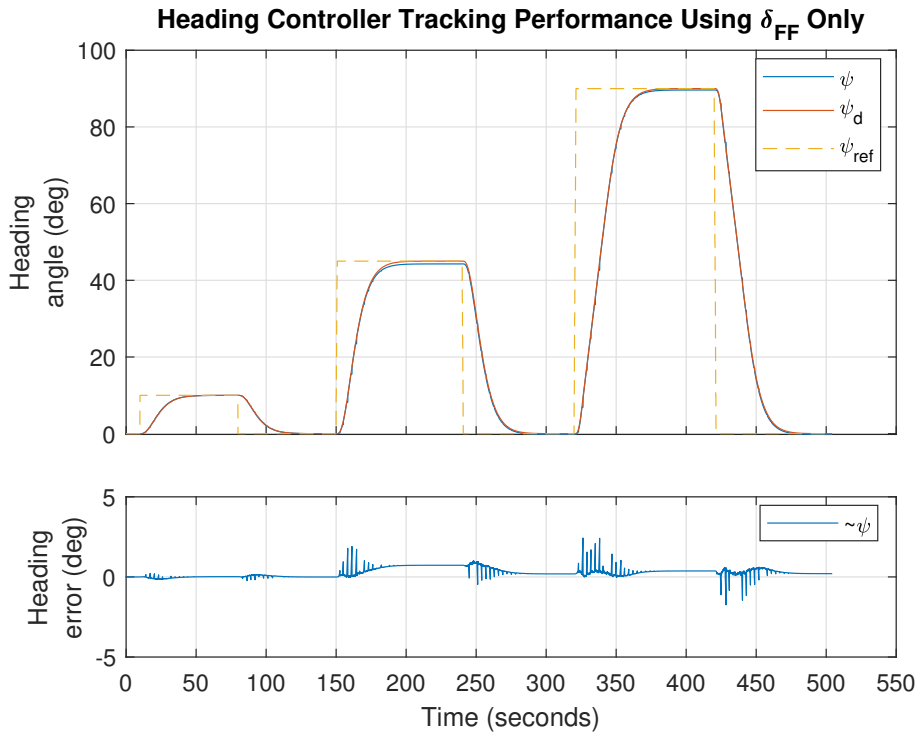


Figure 5.3: Heading controller tracking performance using only model-based feedforward as control input.

In Figure 5.3 a slight steady state error can be observed at the steps of 45^o and 90^o which can be explained by the control input plot in Figure 5.5. Observe that the *Heading Error* shows spikes at step changes. This is because of communication issues between the simulator and control system, as it was observed to be continuous in the simulator during testing. Feedforward parameters are listed in Table 5.1.

Feedforward Parameters	
K	-0.1361
T	2.1170
ω_n	0.35
ζ	1

Table 5.1: Feedforward parameters used in simulation.

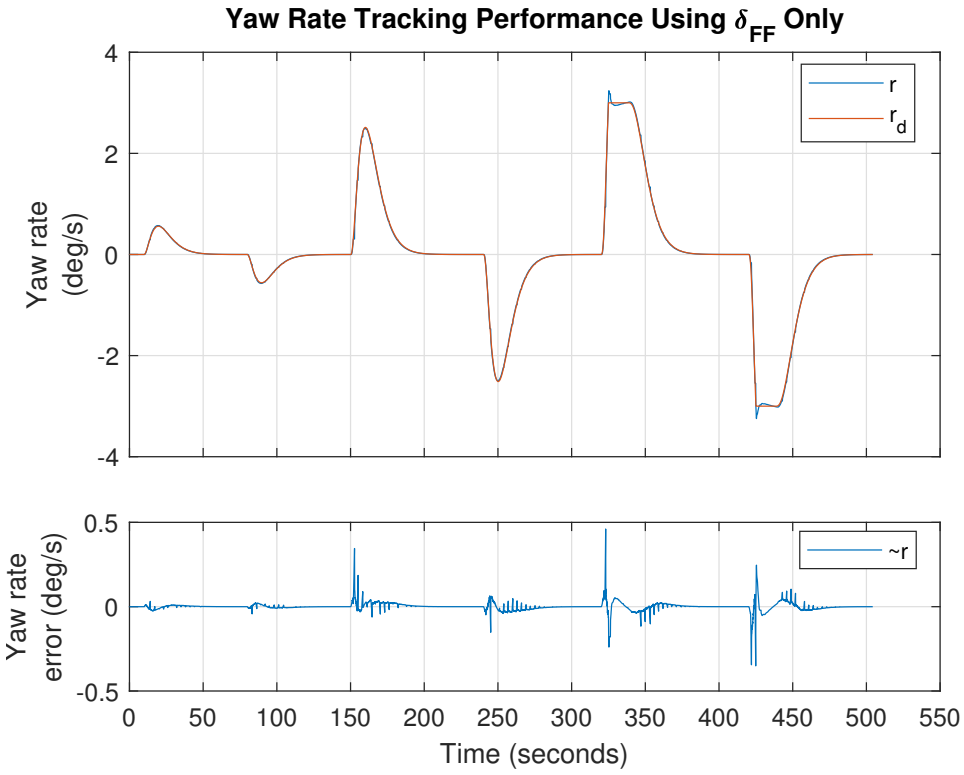


Figure 5.4: Yaw rate tracking performance using only model-based feedforward as control input.

In Figure 5.4 the corresponding yaw rate tracking performance is shown. At $t \approx 325$ the reference filter saturates at $3\ ^o/s$ and a slight overshoot appears. In the implementation, the yaw acceleration is set to 0 (instant steady-state) when yaw rate reaches its limit and the vessel's yaw acceleration is not able to follow.

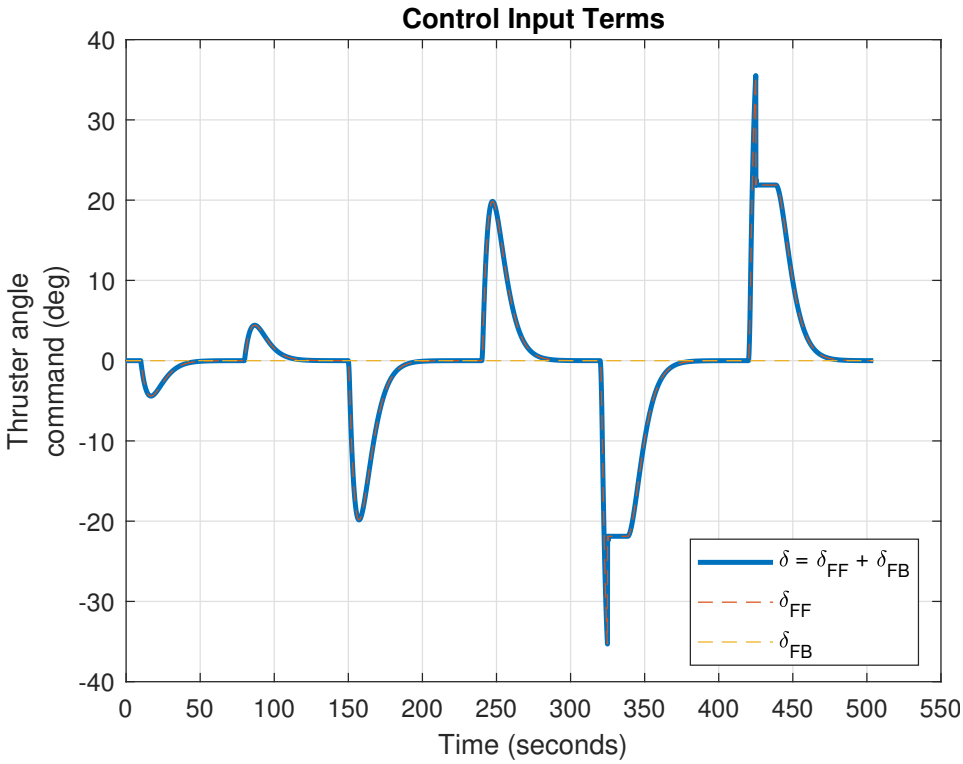


Figure 5.5: Thruster angle commands generated by the feedforward term.

Figure 5.5 shows the control input using only the feedforward term with desired states generated by the reference filter. The model is most accurate for steps in thruster angle close to $\delta = \pm 5^o$. The input can be observed to be larger causing the small offset in Figure 5.3. At $t \approx 325$ and 425 (top of the "spikes") the desired yaw rate saturates, setting $\dot{r}_d = 0$ causing the decrease and flattening control input.

5.2.2 PD Feedback Control Only

The controller gains are generated as described in Section 4.5.6. Initially however, the reference model and feedback gains where chosen with a bandwidth ω_b such that $\omega_n = 0.35$ rad/s which resulted in the following gains used in the heading controller simulations. Note that the integral gain K_i is set to zero due to no environmental disturbances.

Feedback Gains	
K_p	-2.1853
K_i	0
K_d	-5.1932

Table 5.2: Controller gains used in simulations with feedback control only.

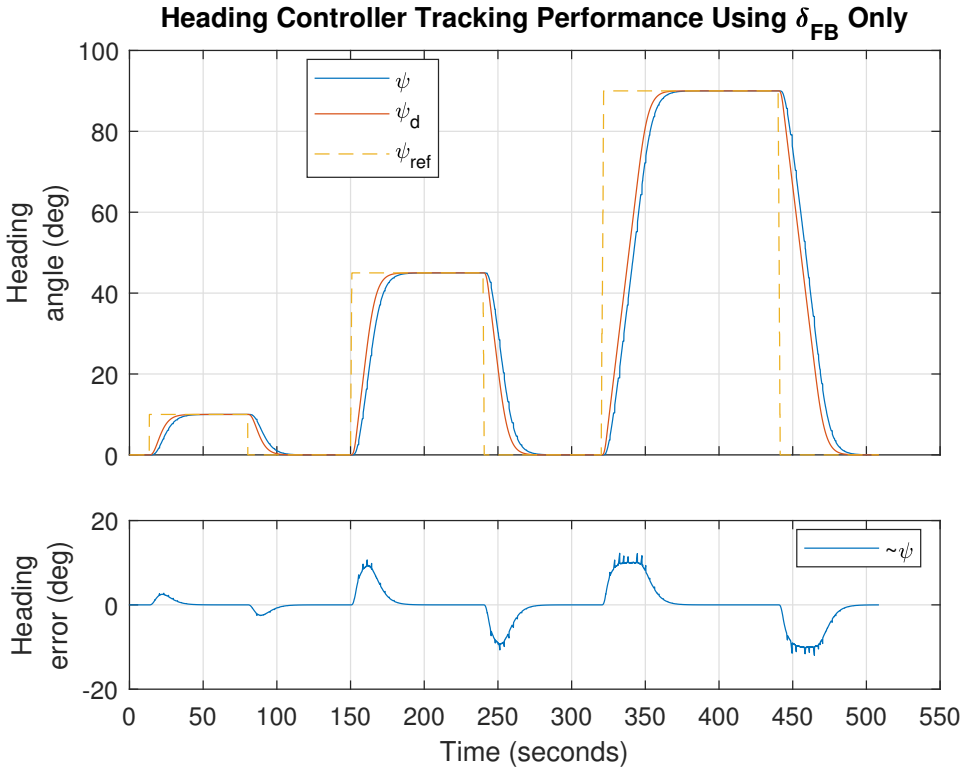


Figure 5.6: Heading controller tracking performance using only PD feedback as control input.

In Figure 5.6 the tracking performance using only feedback is shown. As can be expected from a PD, the response lags a few seconds behind. This is because during course changing, the PD will only act if there is a deviation from the desired states. It does however, converge nicely for all steps.

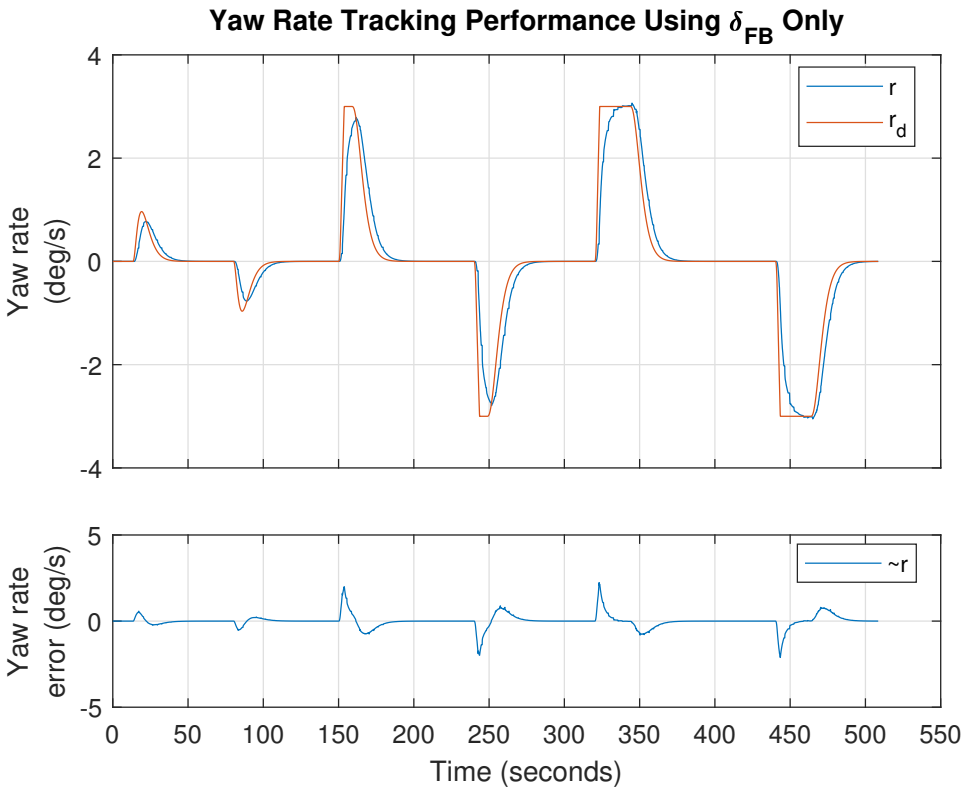


Figure 5.7: Yaw rate tracking performance using only PD feedback as control input.

In Figure 5.7, the yaw rate also lags behind when only using feedback for control before converging.

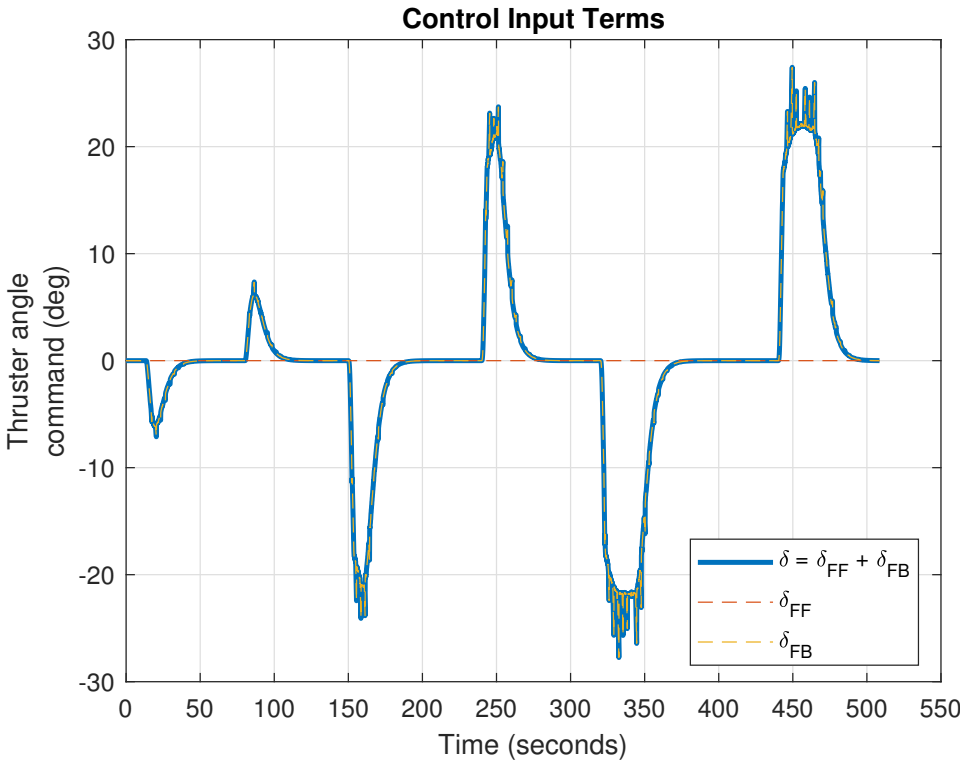


Figure 5.8: Thruster angle commands generated by the feedback term.

The control input is generated by yaw and yaw rate errors, which suffers from signal ”dropout” for a few milliseconds (see error plots of Figures 5.6 and 5.7). As a result, the control input also suffers accordingly as seen clearly at response maxima and minima in Figure 5.8.

5.2.3 Combined Feedforward and PID Feedback s.t. Wind

Combined FF+FB simulation parameters	
K_p	-2.185
K_i	-0.076
K_d	-5.193
ω_n	0.350
ζ	1.000
K	-0.136
T	2.117

Table 5.3: Controller gains used in feedback simulations

For the combined Feedforward (FF) + Feedback (FB) simulations, a time-varying wind disturbance ≈ 2 m/s is added with direction -90^o in $\{n\}$. Figure 5.9 shows the tracking performance for six steps. Notice the longer simulation time as the response needs longer time to stabilize, and the nonzero integral gain. In the steady-state region of $\psi = 90^o$, the wind directions is parallel to the vessel and bow-facing.

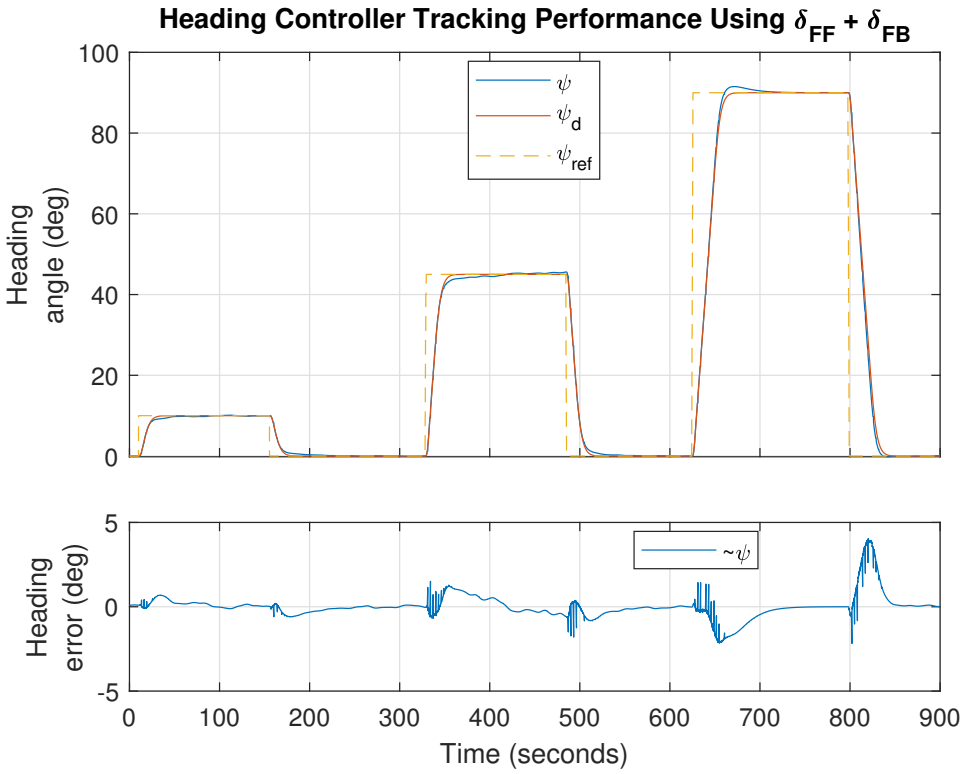


Figure 5.9: Heading controller tracking performance using model-based feedforward and PID feedback as control input.

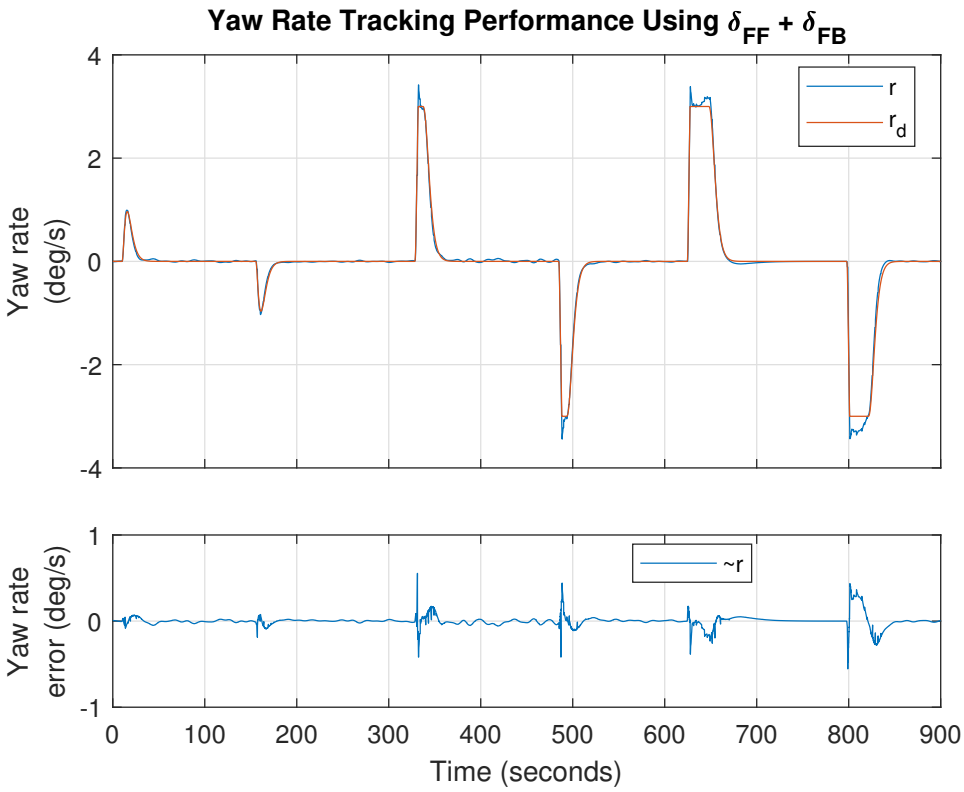


Figure 5.10: Yaw rate tracking performance using model-based feedforward and PID feedback as control input.

In Figure 5.10, a combined feedforward and feedback improves tracking during course-changing maneuvers (compared to Figure 5.7) as well as rejecting the errors caused the wind.

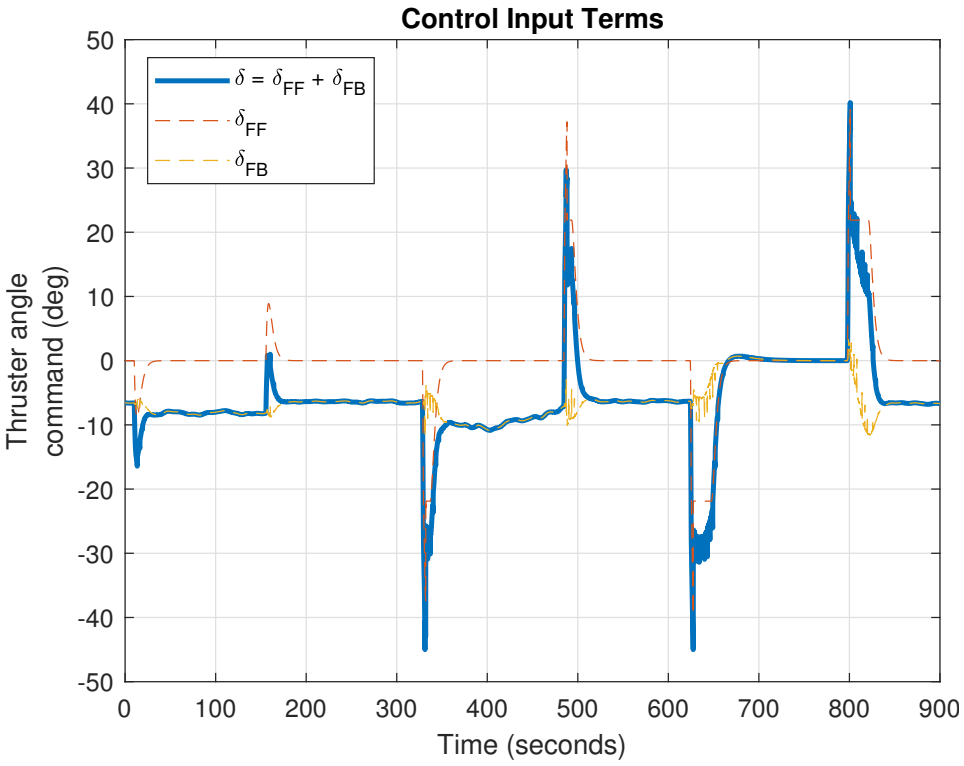


Figure 5.11: Thruster angle commands generated by the sum of model-based feedforward and PID feedback term.

In Figure 5.11, the combined FF and FB effort is shown. The FB term is constantly working to reject the disturbance caused by the wind, except at the steady-state region from $t \approx 700$ to $t \approx 800$ as the wind is directly facing the vessel.

5.3 Speed Controller Performance

The Speed Controller's tracking performance, developed in Section 4.4, is presented here. A total of three steps is performed, starting at 0 to 0.3 m/s, 0 to 0.5 m/s and lastly, 0 to 1.0 m/s. Each of these steps is performed using:

1. Feedforward control only (no disturbances)

2. Combined Feedforward and Proportional-Feedback Control (no disturbances)

3. Combined Feedforward and PI-Feedback Control (subject to ocean current)

For each simulation, a plot of the tracking performance and the corresponding deviation from the desired state is presented. Furthermore, a plot of the control input that resulted in the tracking performance in question. Note that the Digital Twin's state is reset after $u \rightarrow u_{ref}$ for a period of time such that next step change and simulation is displayed instantly in the figures (see e.g. Figure 5.13) and that the vessel's heading is zero $\forall\ t$.

5.3.1 Feedforward Control Only

In this simulation, no environmental disturbance is present.

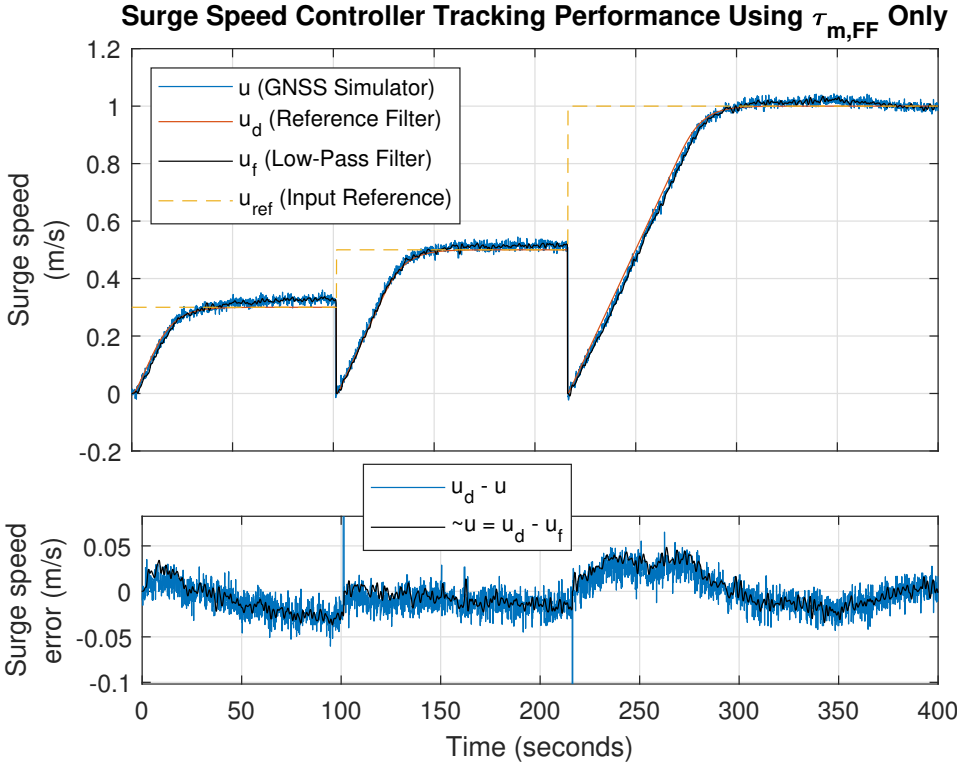


Figure 5.12: Surge speed tracking performance using only model-based feedforward as control input. No disturbances present.

Figure 5.12 shows the tracking performance for the surge speed controller using $\tau_{m,FF}$ only, along with the surge speed error for the speed measurement and low-pass filtered measurement. Notice a deviation at the first step and also a slight overshoot in the last. This is caused by modeling errors in the damping term $\sigma(u_d)$ and inertia Mu_d. Sudden drop in surge speed is the result of resetting the simulator between the steps, also causing the spikes in *Surge speed error* (they do not enter the control loop in practice). See Section 4.4.4 for development feedforward term.

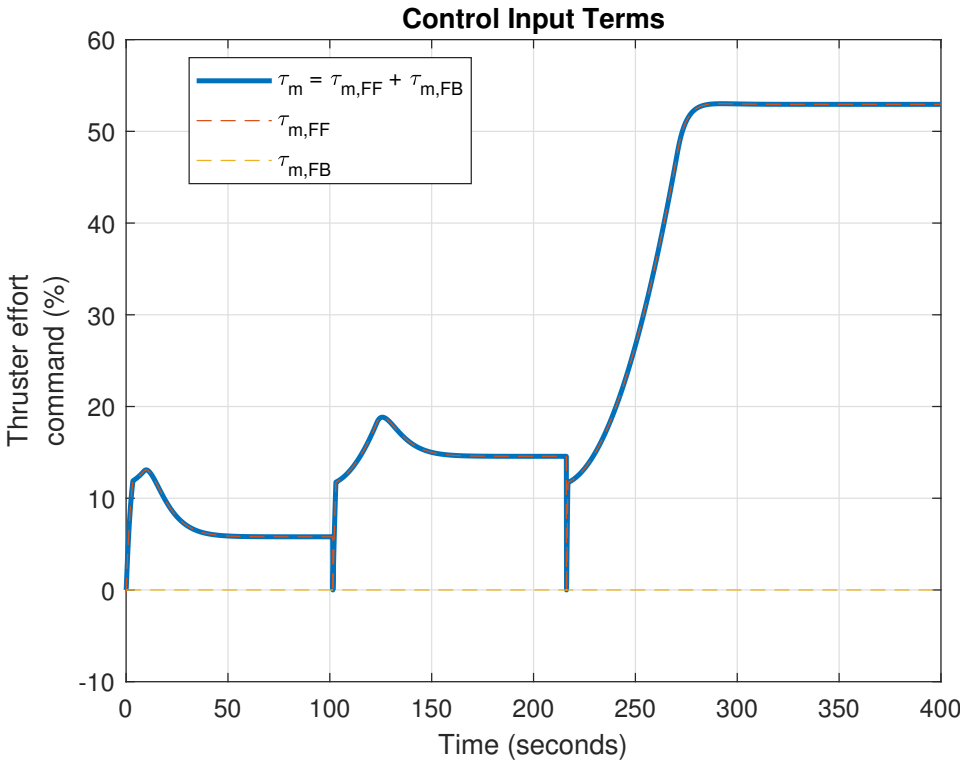


Figure 5.13: Thruster effort commands generated by the model-based feedforward term only. No disturbances present.

Figure 5.13 shows the control input for the surge speed steps of 0.3, 0.5 and 1 m/s (returning to zero after each step convergence) generated by the feedforward term only.

5.3.2 Feedforward and Proportional Feedback Control

Parameters for controller is listed in Table 5.4. In this simulation, no environmental disturbance is present.

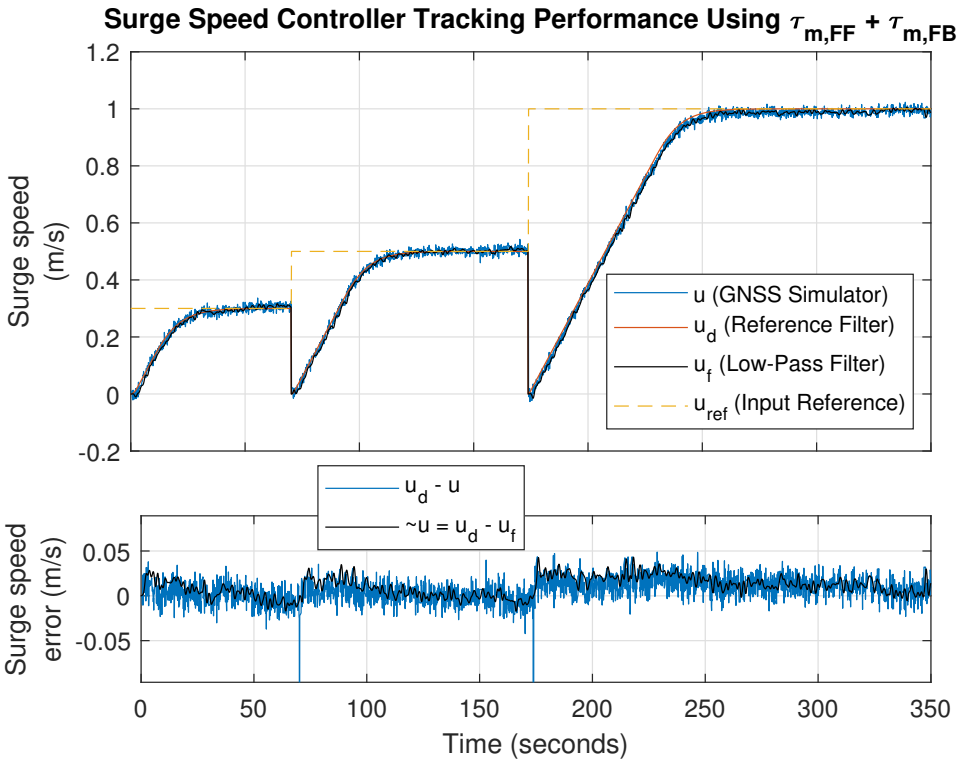


Figure 5.14: Surge speed tracking performance using model-based feedforward and proportional feedback as control input. No disturbances present.

Combining feedforward and proportional feedback eliminates the steady-state error and overshoot.

Parameter	Value
K_p	50
K_i	0
M	800
$\sigma(u_d)$	see (4.34)

Table 5.4: Parameters used in controller

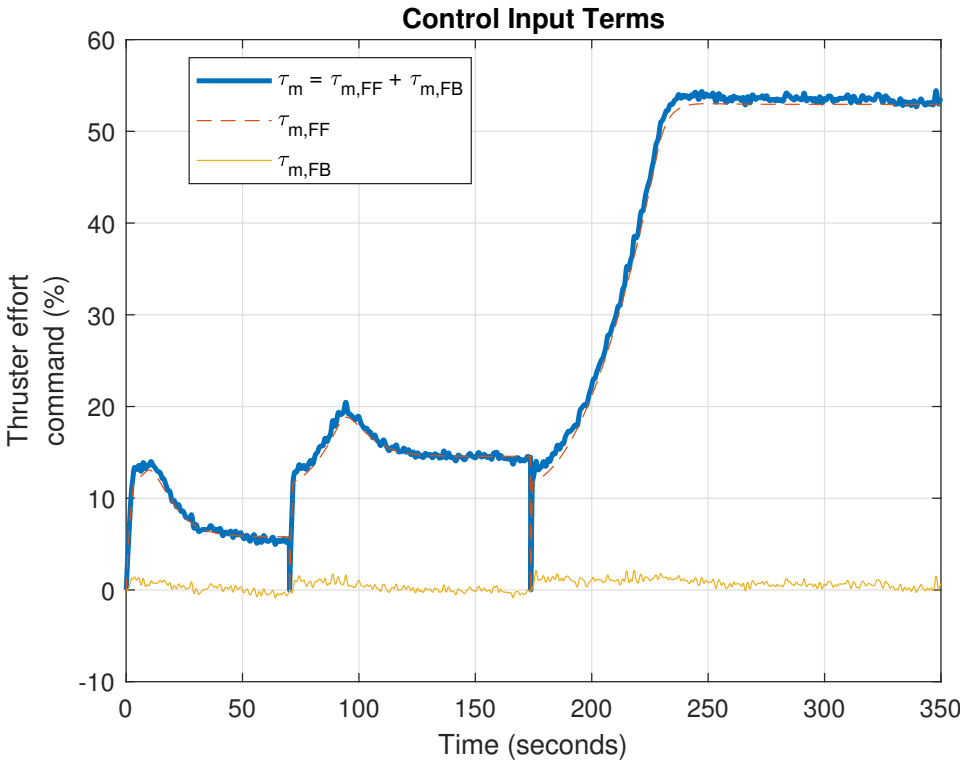


Figure 5.15: Thruster effort commands generated by the model-based feedforward and proportional feedback. No disturbances present.

The proportional feedback accounts for just a small portion of control effort applied when not subject to disturbances.

5.3.3 Feedforward and PI Feedback s.t. Ocean Current

Disturbance Parameters:

- Current Speed: 0.3 m/s
- Current Direction: 180^o in $\{n\}$

The current is directed towards the vessel such that no sideslip occurs.

Controller Parameters:

Parameter	Value
K_p	100
K_i	7.5
M	800
$\sigma(u_d)$	see (4.34)

Table 5.5: Controller gains used in this simulation for feedforward and feedback control.

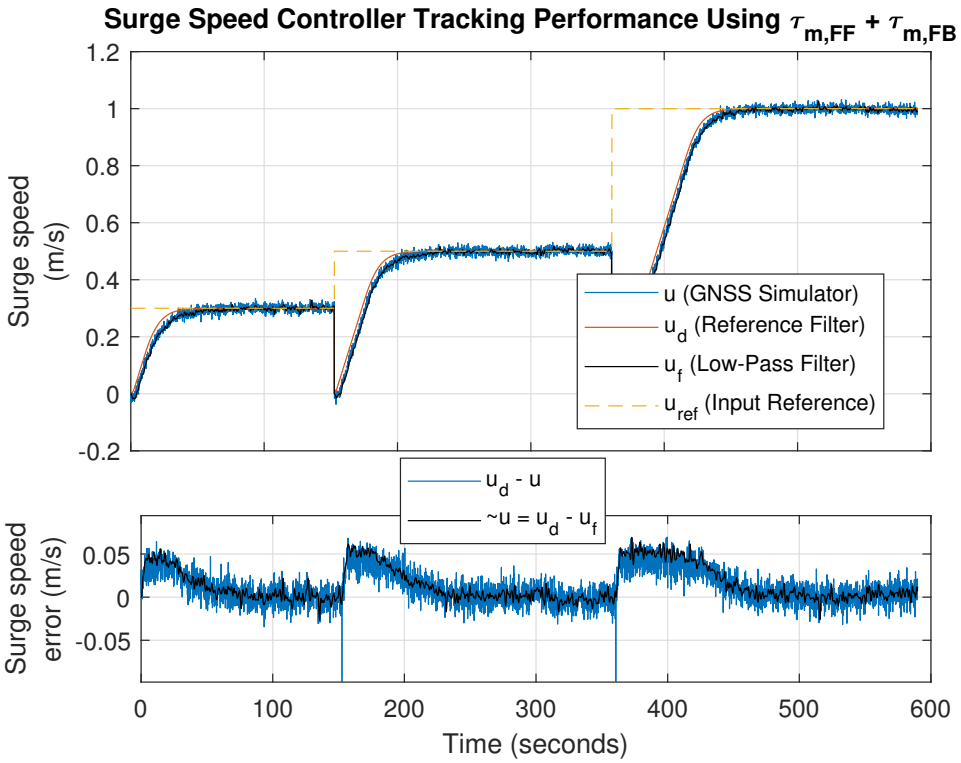


Figure 5.16: Surge speed tracking performance using model-based feedforward and proportional-integral feedback as control input.

In Figure 5.16, a small offset appears in the transient response for each step. There is a slight delay due to the low-pass filtering of the velocity measurement and the initial conditions of the surge speed is nonzero unlike the filter.

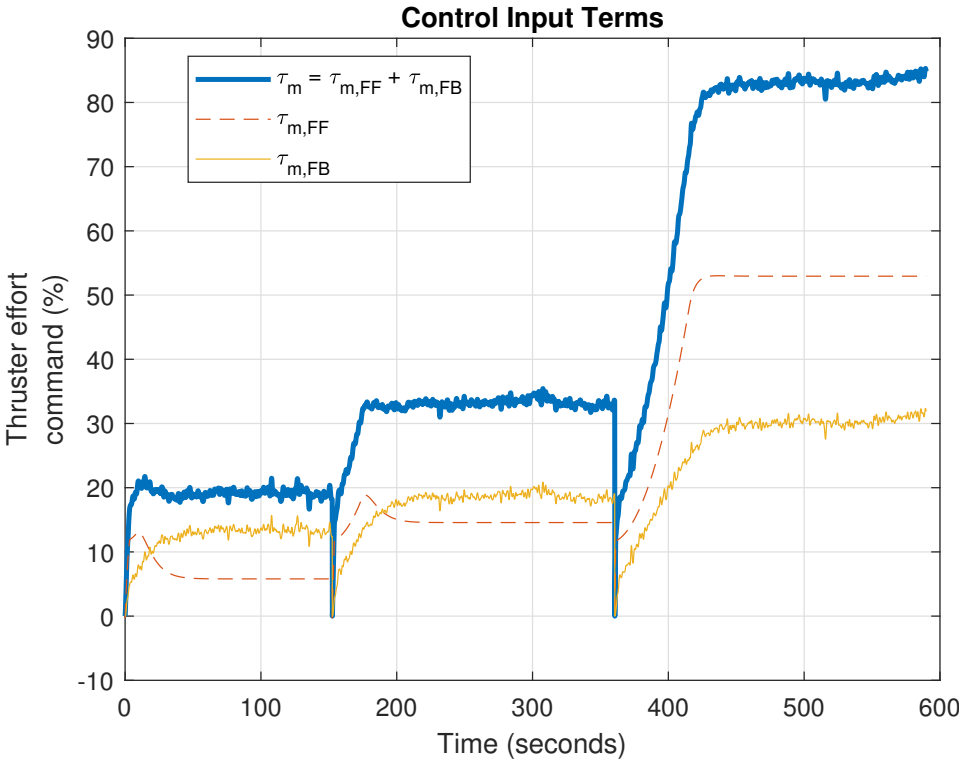


Figure 5.17: Thruster effort commands generated by the model-based feedforward and proportional-integral feedback.

In Figure 5.17, the combined feedforward feedback control input applied to the vessel is shown. The constant ocean current is directed at the bow of the vessel increasing the control effort needed to counteract the disturbance.

5.4 Performance of Guidance System for Path Following

The simulation results for the LOS Guidance system developed in Section 2.5, with the speed and heading controller developed in Section 4.4 and 4.5 is presented here. Two LOS Guidance simulations is presented:

1. Subject to wind and ocean current

2. Subject to both stronger wind and ocean current

For both simulations, the wind is time-varying and ocean current is constant in $\{n\}$.

LOS Guidance Parameters [m]	
Δ	12
R_{k+1}	12

Table 5.6: LOS Guidance parameters.

Table 5.6 lists the parameters used in the LOS Guidance algorithm for these simulations. Remember that Δ is the lookahead distance from (2.28) which decides the aggressiveness of the steering law. A small Δ yields a faster convergence. R_{k+1} is the circle of acceptance radius used to decide when to switch waypoints. It is however, not necessary for the vessel to be inside the circle to switch waypoints, which is done using the along-track distance from (4.6).

The heading controller parameters for the following simulations, obtained from Section 4.5 are restated in Table 5.7.

Heading Controller Parameters	
K_p	-6.422
K_i	-0.385
K_d	-14.113
ω_n	0.600
ζ	1.000
K	-0.136
T	2.117

Table 5.7: Heading controller parameters for LOS Guidance.

The Speed controller's parameters remains the same.

5.4.1 Subject to Wind and Ocean Current

Environmental disturbance data:

- Wind Speed: Varying around ~ 2 m/s
- Wind Direction: 180^o in $\{n\}$
- Current Speed: 0.2 m/s
- Current Direction: -45^o in $\{n\}$

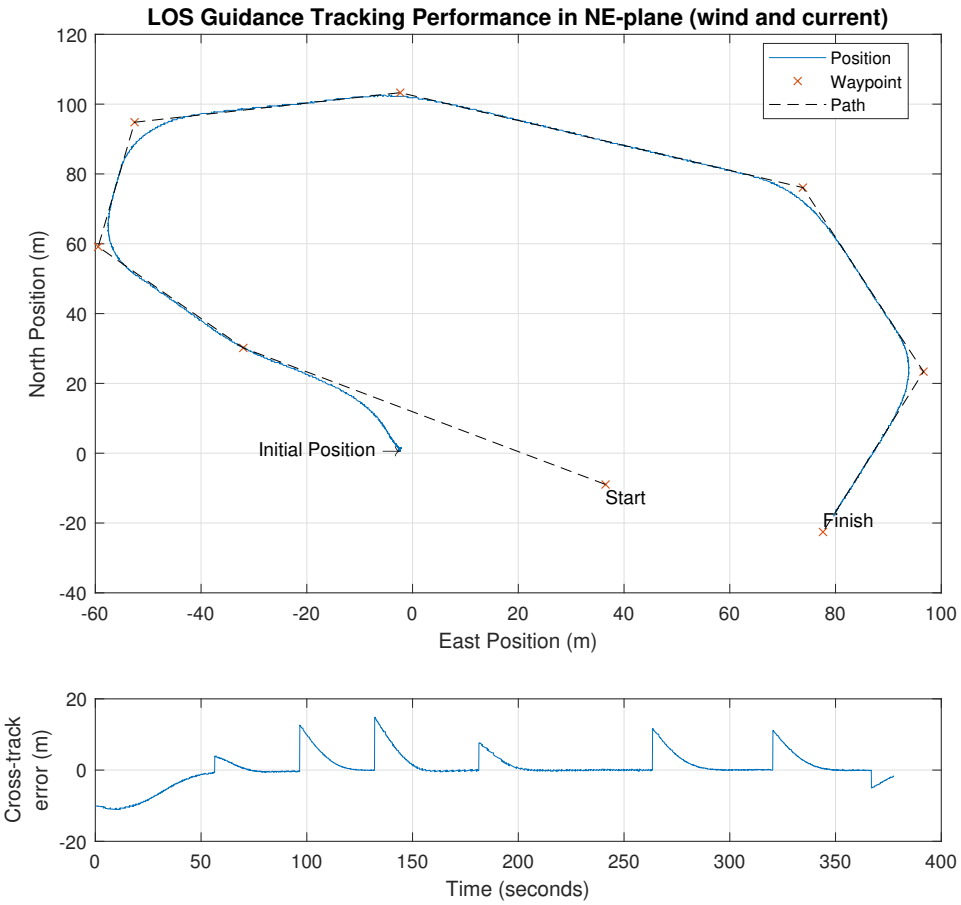


Figure 5.18: Line-of-sight Guidance simulation with 7 waypoints and cross-track error subject to environmental disturbances.

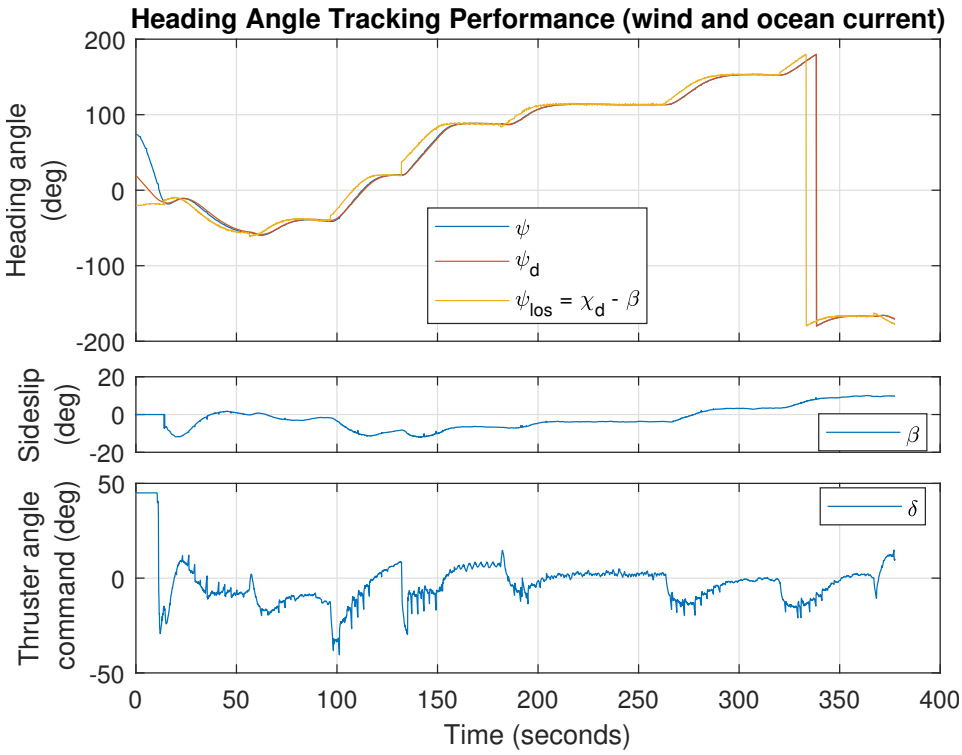


Figure 5.19: Heading controller tracking performance with LOS heading ψ_{los} desired heading ψ_d generated by the reference model and current heading ψ. Sideslip angle β and total thruster angle command $\delta = -\tau_\delta$.

Figure 5.18 shows the a LOS simulation by positions in the NE-plane with disturbances. It also shows the cross-track error $e(t)$ converging to zero after waypoint switching. Cross-track error after $t \approx 370$ can be neglected as waypoint iteration is already complete.

In Figure 5.19, the LOS angle ψ_{los} is passed through the reference filter to obtain ψ_d and its higher order derivatives. Notice that the desired heading ψ_d generated by the filter is discontinuous at $\langle -\pi, \pi \rangle$, yet the filter's desired heading chooses the shortest path. This is due to the wrapping schemes described in Section 4.5.3. The heading angle ψ tracks the desired heading ψ_d and converges to ψ_{los}. Also, notice a sudden step in β at $t \approx 20$ due to the sideslip compensation being disabled for *speed over ground* $U < 0.2$ as $u \not\gg v$ in the initial phase. The *Thruster angle command* initial angle is saturated due to the difference in initial state of the vessel versus the reference model. Furthermore, it suffers from the same noise as in the Heading controller simulations of Section 5.2. However, far less noisy compared to when using the previous heading controller for LOS Guidance (see Appendix D.1).

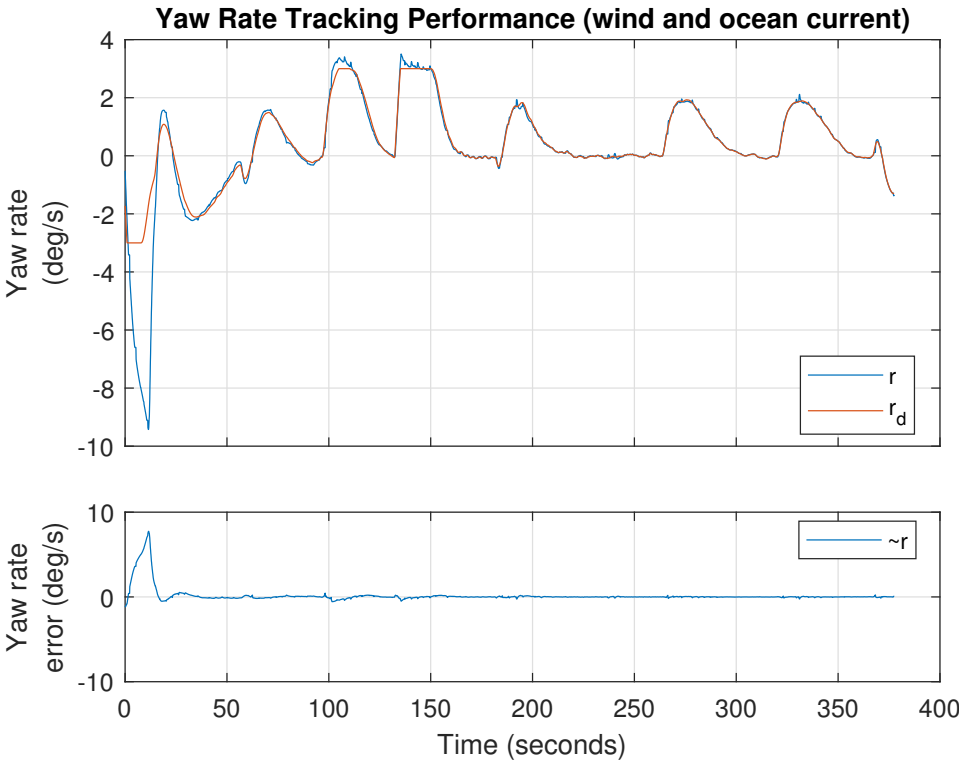


Figure 5.20: Yaw rate tracking performance with desired yaw rate r_d generated by the reference model and current yaw rate r. Along with the corresponding yaw rate error $\tilde{r} = r_d - r$.

The desired yaw rate r_d in Figure 5.20 is obtained from the reference filter. From $t > 25$ the yaw rate tracks the desired yaw rate in a satisfactory manner. Initially, the yaw rate is far off its trajectory due to the difference in initial states between the vessel and reference model (see Figure 5.19).

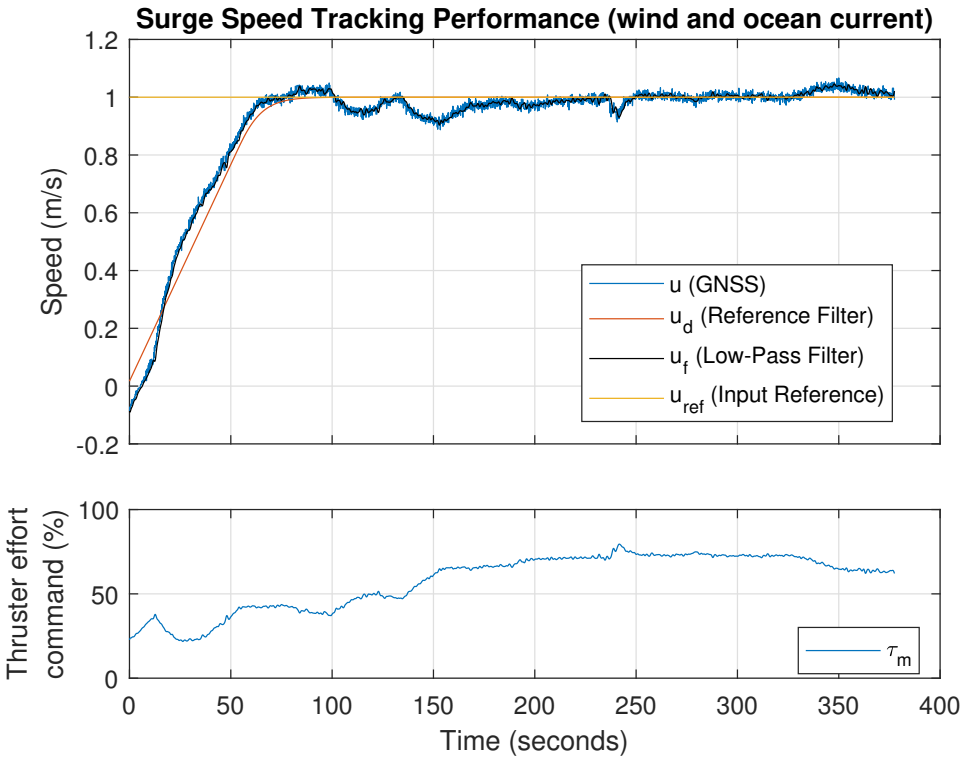


Figure 5.21: Surge speed tracking performance subject to wind and current.

Figure 5.21 shows the surge speed tracking performance during a LOS simulation with reference speed 1 m/s. The transient has some deviation, contributed by the filter initial conditions not being identical to the vessel's. Also, sideslip compensation kicks in at $U \geq 0.2$ m/s and that u/u_f decreases during turning. Due to occurring waypoint switches, the response don't always have time to reach steady state. The distance between waypoints could have been increased, but waypoints were placed inside *Dora Test Pool* (see e.g. Figure 3.5). Alternatively, a smaller U_{ref} gives the response more time to converge.

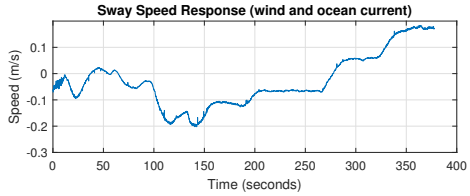


Figure 5.22: Sway speed response for the LOS simulation subject to wind and current.

In Figure 5.22, the sway speed v response is shown. This response would contribute to the velocity transformation $U_{ref} \rightarrow u_{ref}$ from (4.9) not implemented here.

5.4.2 Subject to Both Stronger Wind and Ocean Current

Environmental disturbance data:

- Wind Speed: Varying around ~ 4 m/s
- Wind Direction: 180^o in $\{n\}$
- Current Speed: 0.4 m/s
- Current Direction: -45^o in $\{n\}$

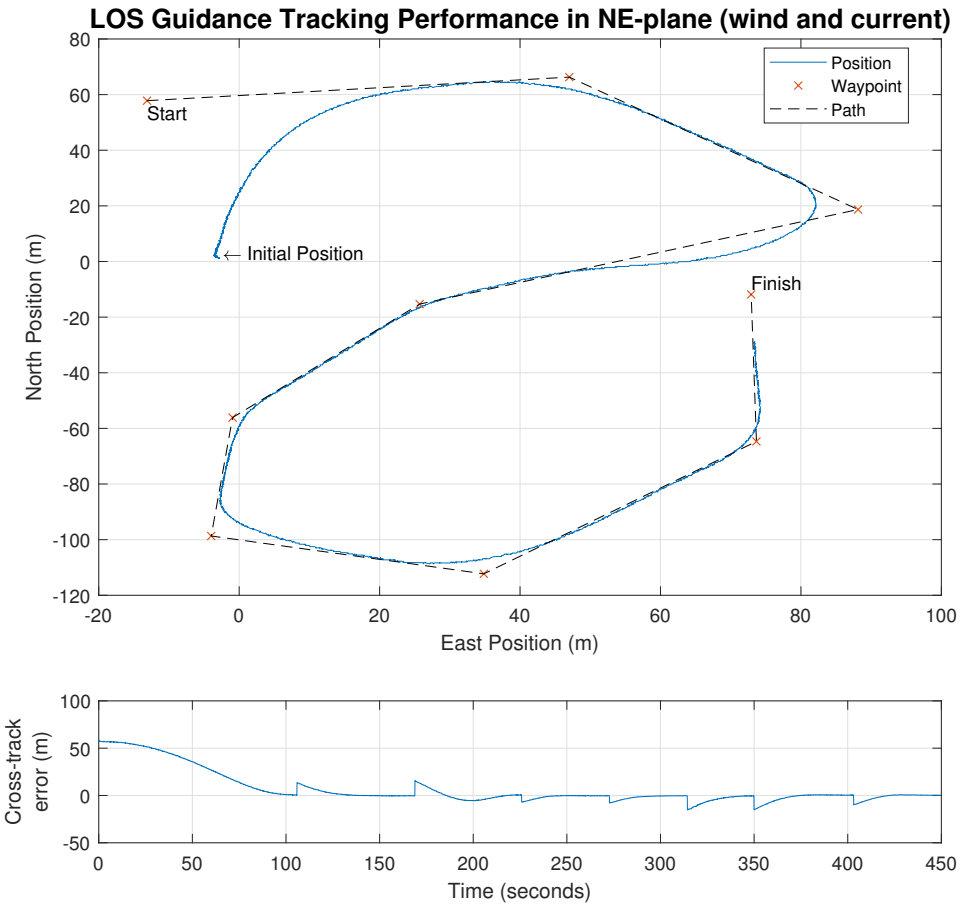


Figure 5.23: Line-of-sight Guidance simulation with 7 waypoints and cross-track error e.

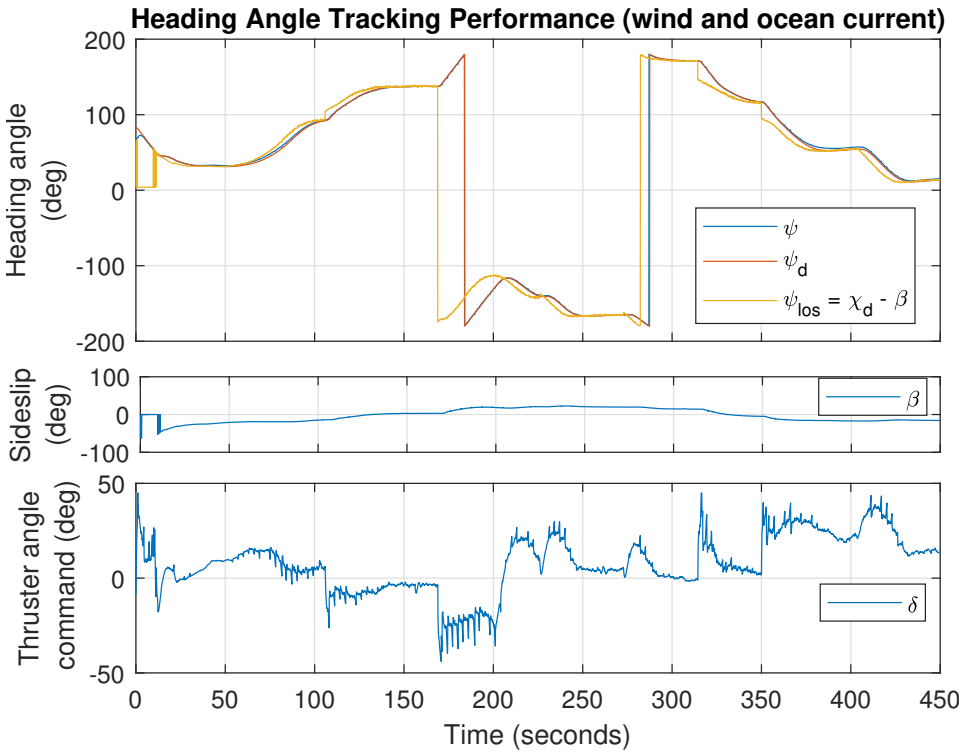


Figure 5.24: Heading controller tracking performance with LOS heading ψ_{los} desired heading ψ_d generated by the reference model and current heading ψ. Sideslip angle β and total thruster angle command δ is also shown. Here subject to stronger disturbances.

In Figure 5.23 the LOS tracking performance subject to stronger disturbances is shown. Despite the extra disturbance the vessel still tracks the desired path, the *lookahead distance* Δ could however be smaller giving a faster convergence to the path in the initial phase and through waypoint 3. One could argue that the angle between waypoint 2 and 4 is to steep or that cruising speed is to high. A speed profile for each waypoint based on steepness and length could resolve this issue.

In Figure 5.24, the vessel's heading angle ψ still tracks the desired heading ψ_d sufficiently with the reference ψ_{los} and added sideslip β compensation. Notice also the low noise in *Thruster angle command*, compared to that in Appendix D.1.

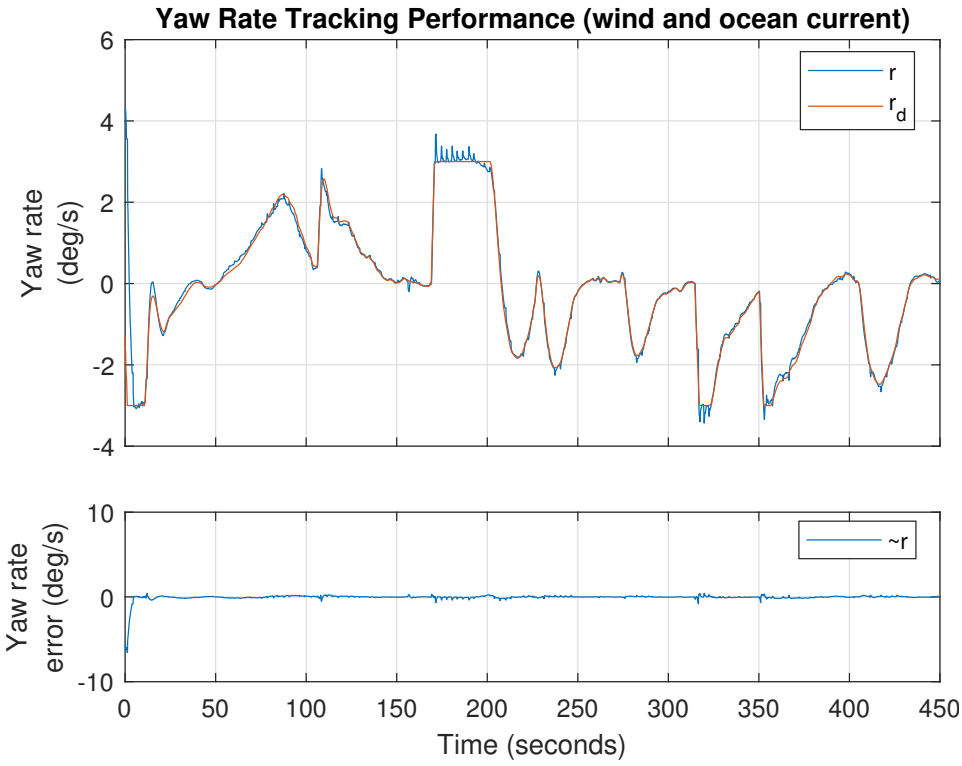


Figure 5.25: Yaw rate tracking performance with desired yaw rate r_d generated by the reference model and current yaw rate r. Along with the corresponding yaw rate error $\tilde{r} = r_d - r$. Here subject to stronger disturbances.

Figure 5.25 shows yaw rate tracking and yaw rate error when subject to stronger disturbances.

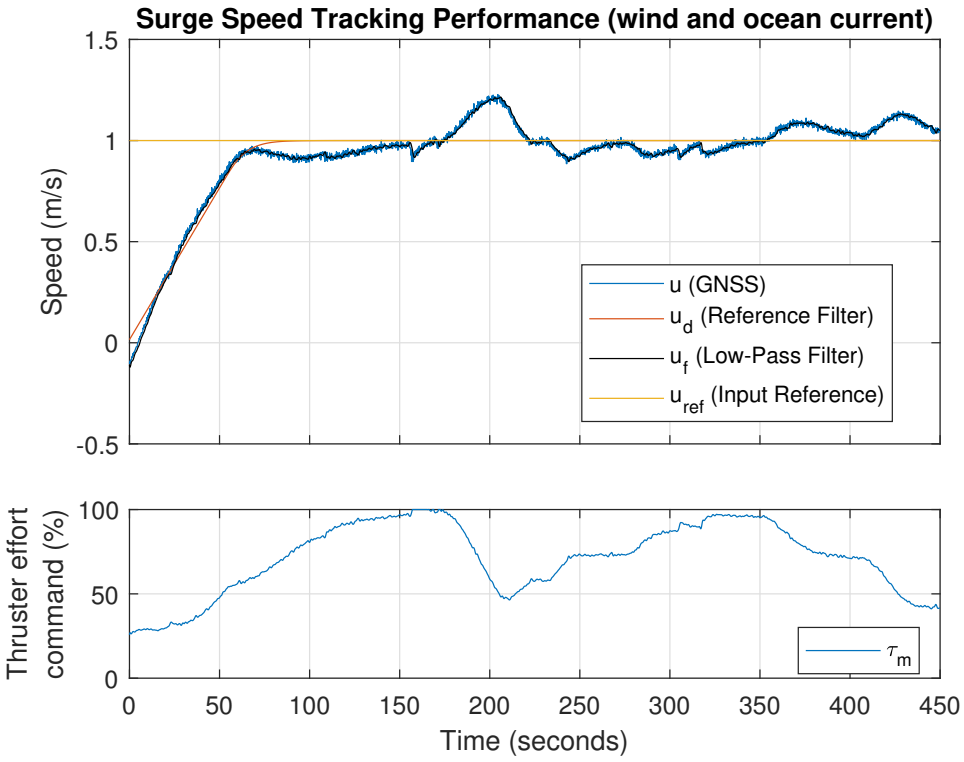


Figure 5.26: Speed controller tracking performance with input reference u_{ref}, desired surge speed u_d generated by the velocity reference model, low-pass filtered velocity measurement u_f and surge speed u decomposed from speed over ground U. Here subject to stronger disturbances.

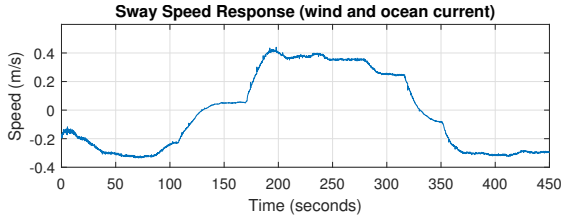


Figure 5.27: The indirectly controlled surge speed response v. Here subject to stronger disturbances.

In Figure 5.26, the surge speed convergence is interrupted by the changes in heading due to waypoint switching. Also, the velocity reference transformation (4.9) in Section 4.2 is not implemented here, causing the reference surge speed u_{ref} to be constant, i.e. trying to maintain a surge speed of 1 m/s while having a non-zero sway speed (see Figure 5.27), causing $U > u_{ref}$.

5.5 Discussion

The controllers for heading and speed are developed to be used by the Guidance System for path following and their performance directly affect the results of that. By performing different step responses for speed and heading, the dynamics during course-keeping and course-changing maneuvers are presented. The heading controller showed that using FF only, a satisfactory tracking could be obtained. Adding disturbance in the form of wind was corrected by a PID feedback controller and resulted in good tracking despite not having wind FF.

The surge speed controller also tracked the desired trajectory, but due to some modeling errors, some deviations were observed. This was corrected using a proportional feedback, which resulted in satisfactory performance. The integral feedback was added to counteract the ocean currents.

Combining these two controllers, the Guidance System tracking performance proved to work well, with some reduction in speed controller performance. Given more time, the speed controller would converge as well. However, during course-changing maneuvers, the surge speed drops and sway speed increases. A transformation $U_{ref} \rightarrow u_{ref}$ would lower the surge speed reference during turning, such that $U = U_{ref}$ and reducing extra demand for control effort (see (4.9) in Section 4.2).

Chapter 6

Experimental Results

All experimental tests were performed at Dora I in Trondheim, from which the image in Figure 6.1 is taken. A map of the test area is shown in Figure 6.3. The location is sheltered from waves, and the ocean current is fairly weak further inside the test area. A brief presentation of the experimental platform ReVolt is presented first, followed by the information about the test area and routines. The experimental results obtained, are discussed in Section 6.5. In this chapter the following test are presented

1. Heading Controller Experimental Performance
2. Performance of Guidance System for path following with combined heading and speed controller

6.1 Experimental Platform

Figure 6.1: ReVolt at Dora Test Pool in Trondheim.

A brief description of the experimental platform ReVolt is presented here. For a thorough explanation of ReVolt's sensors, actuators and embedded computerized control platform refer to [1].

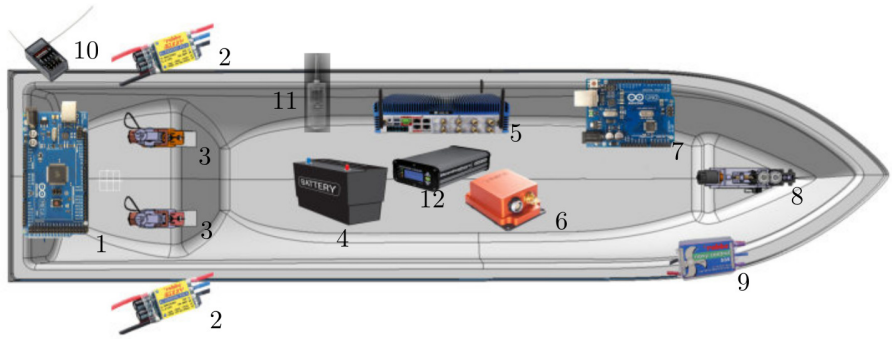
6.1.1 Background

ReVolt is a 1:20 scale-model of the concept ship ReVolt. The concept ship was designed by the international certification body and classification society, DNV GL in 2014. It is also unmanned, fully electric and designed for speed around 6 knots (concept ship). ReVolt (scale) is built by Stadt Towing Tank (STT).

ReVolt (scale) specifications is listed in Table 6.1

Length	3m	Draft	0.23m
Width	0.72m	Battery voltage	12V
Weight	257kg	Max engine power	360W
Top speed	2 knots	Battery capacity	900Wh

Table 6.1: ReVolt (scale) Specifications from [1].



No.	Name	No.	Name
1	Arduino Mega	2	ESC - AC Robbe Roxxy
3	Stern Thruster	4	Batteries
5	Tank-720 PC	6	Xsens MTI-G-710
7	Arduino Uno	8	Bow Thruster
9	ESC - DC Robbe Roxxy	10	Spektrum AR400 RC Receiver
11	Satel Radio Link	12	Hemisphere Vector VS330

Figure 6.2: Main components and their placements on ReVolt. Figure from Stadt Towing Tank and [1].

6.1.2 Main Components

Figure 6.2 shows and lists the main components of ReVolt. Providing position and heading, a **Hemisphere Vector VS330** GNSS. This two antennas to determine position and heading as well as additional correction data such as Satellite Based Augmentation System (SBAS) and Real Time Kinematic (RTK). GNSS measurements for the experimental tests used SBAS as RTK was not available at the time.

The **Xsens MTI-G-710** IMU also provides position in addition to roll, pitch and yaw motion. Note that the yaw and yaw rate used in the heading controller is provided by the IMU and not the GNSS as this was the default configuration in the control system.

Two stern azimuth thrusters made up of two AC motors for thrust and stepper motors for direction. The retractable and smaller bow thruster uses a DC motor for thrust. The OBC that runs the control system is a **Tank-720-PC** with Ubuntu 16.04. A radio controller (RC) receiver is connected to enable RC remote control. ReVolt uses two 12V batteries for power.

6.2 Test Area

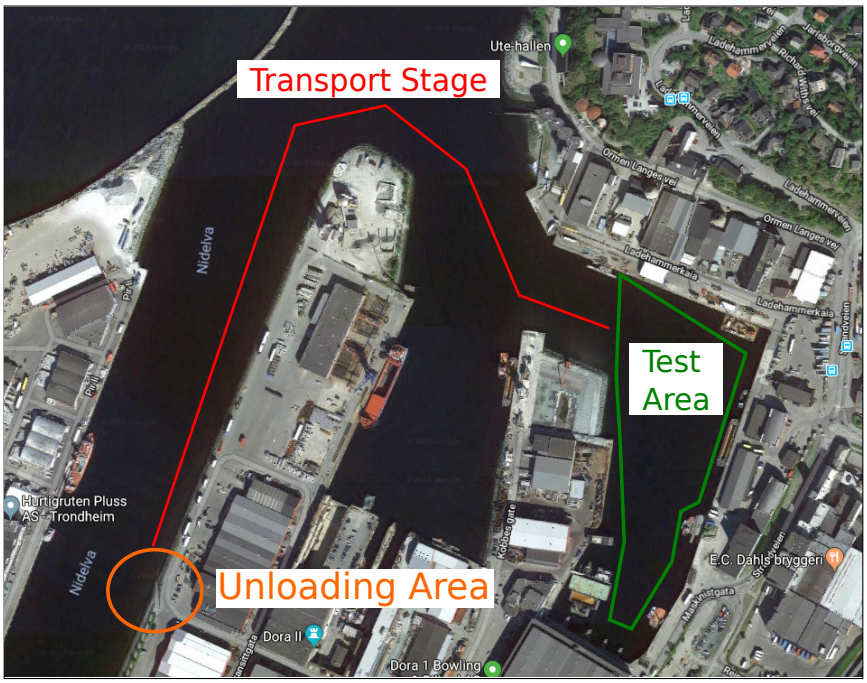


Figure 6.3: Map of the test area, from Google, with unloading area and transport stage marked.

At test-day, ReVolt is unloaded from transportation at the *Unloading Area* next to the launch-ramp. Pre-launch checks of hull, sensors and actuators are performed w.r.t the ReVolt's User Manual for testing. After launch, ReVolt is transported to Dora 1 using *Gunnerus Workboat*, from now on referred to *Workboat*, shown in Figure 6.4. A challenge during transportation is the sea state rounding the pier. For rough sea states, a drogue is connected to ReVolt for stability, and it is towed 3-5 m behind *Workboat* (see Appendix C). For calms sea states ReVolt could simply be moored along side *Workboat*.

Figure 6.4: Follow boat Gunnerus Workboat. Courtesy of Tom Arne Pedersen.

6.3 Heading Controller Experimental Performance

The tracking performance for experimental tests of the heading controller is presented here. The plots visualizes the data in the same way as with the heading controller simulations in Section 5.2. In the simulations, FF, FB and FF+FB is performed. Here, however, only two experiments with combined FF+FB is presented. The latter adds integral action. ReVolt is unstable for thruster angles of 0^o, this may be due to weight balancing, so using FF only is ineffective.

Date: May 9th, 2018

Weather conditions:

- Sunny, 23.5^{o}C
- Wind, 6.9 m/s (moderate breeze) from south
- No waves, weak ocean current

The feedback gains used in the experimental results differ slightly from that of the simulations. The parameters and gains are listed in Table 6.2.

Heading Controller Parameters and Gains	
K_p	-10.000
K_i	0.000
K_d	-5.000
ω_n	0.600
ζ	1.000
K	-0.136
T	2.117

Table 6.2: Heading controller parameters for LOS Guidance.

Due to time and battery life the feedback gains were selected as in the previous heading controller. This is because they are known to stabilize ReVolt. New tests with parameters in Table 5.3 and 5.7 is desirable.

6.3.1 Combined Feedforward and PD Feedback

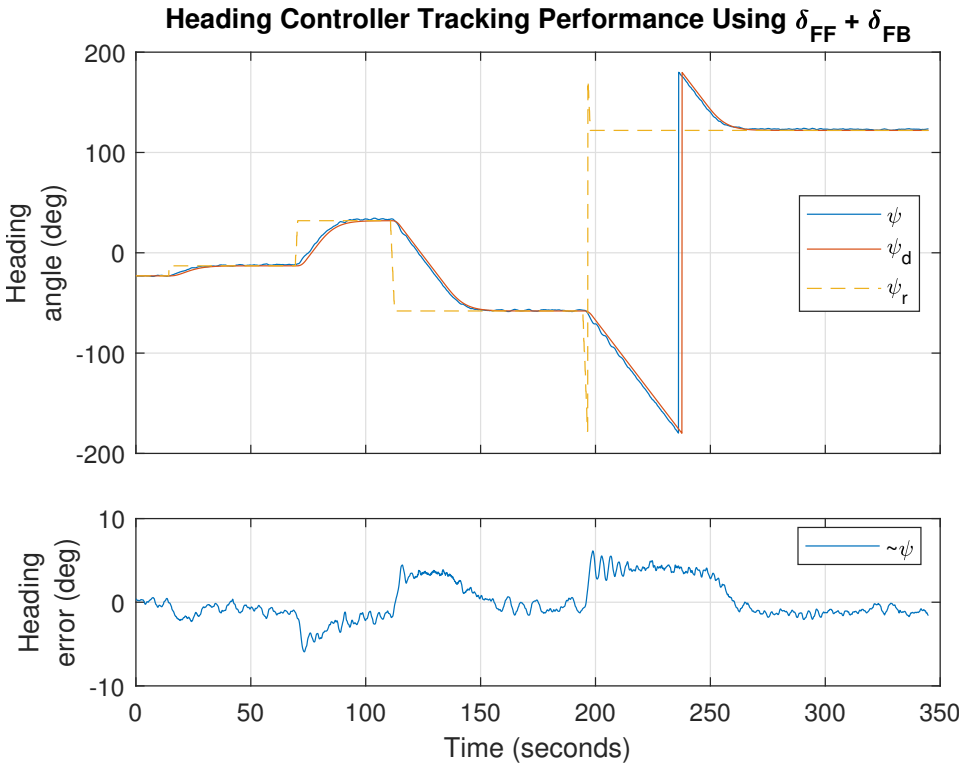


Figure 6.5: Heading controller tracking performance using model-based feedforward and PD feedback as control input. The reference ψ_r cannot be set directly from the RMC station, but has to be incremented quickly in steps of 10 at e.g. $t \approx 190$.

Figure 6.5 shows an experiment with four step changes. The relative sizes are 10^o, 45^o, -90^o and -180^o. Step sizes similar to those in simulations where selected, with some differences due to test area limits. No integral action is present during this test, and the largest errors occur during course-changing maneuvers.

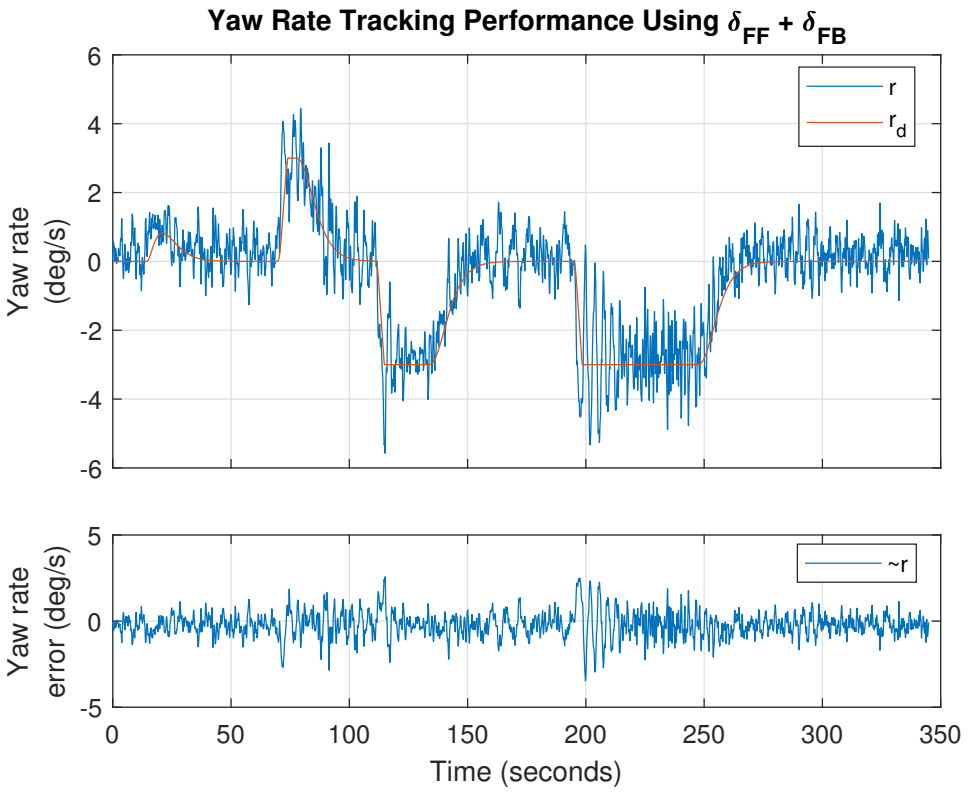


Figure 6.6: Yaw rate tracking performance using model-based feedforward and PD feedback as control input.

Figure 6.6 shows the yaw rate tracking performance. Comparing to Figure 5.10, the yaw rate has more difficulty stabilizing but still tracks the desired trajectory.

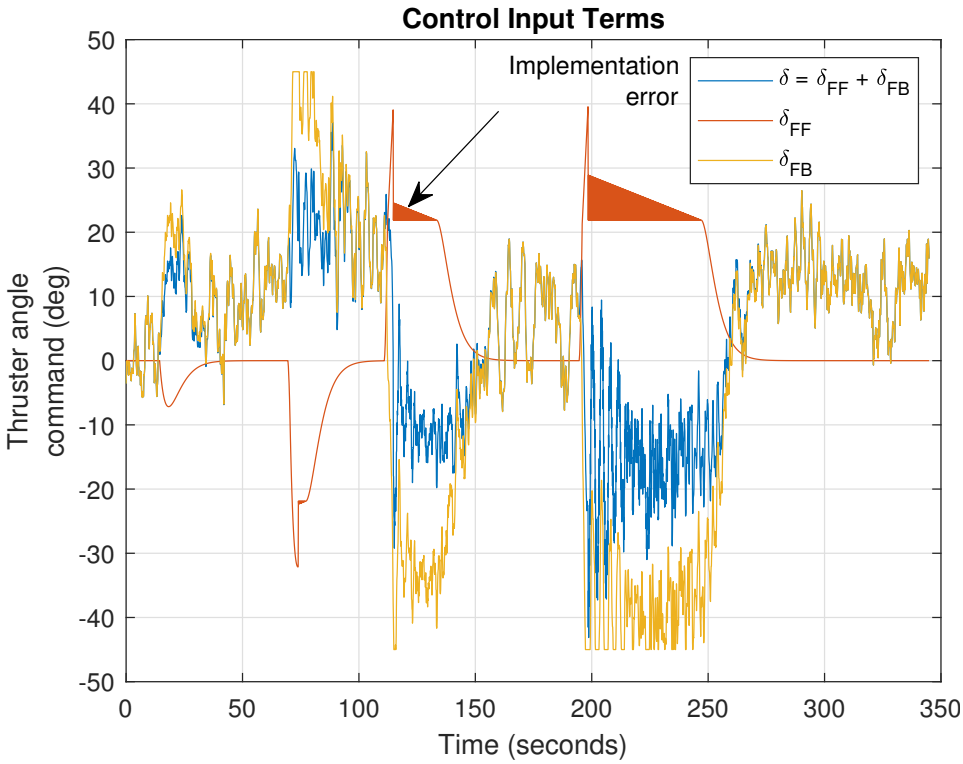


Figure 6.7: Thruster angle commands generated by the model-based feedforward and PD feedback term. The implementation errors are the red triangles, caused by repeated incrementation of r_d ($r_d > r_{max}$) at time step k and saturating again at time step $k+1$ ($r_d = r_{max}$). The error is resolved in LOS experiments.

Compared to the Figure 5.11, the feedback term (with different gains) in Figure 6.7 needs to work a lot more to stabilize ReVolt.

6.3.2 Combined Feedforward and PID Feedback

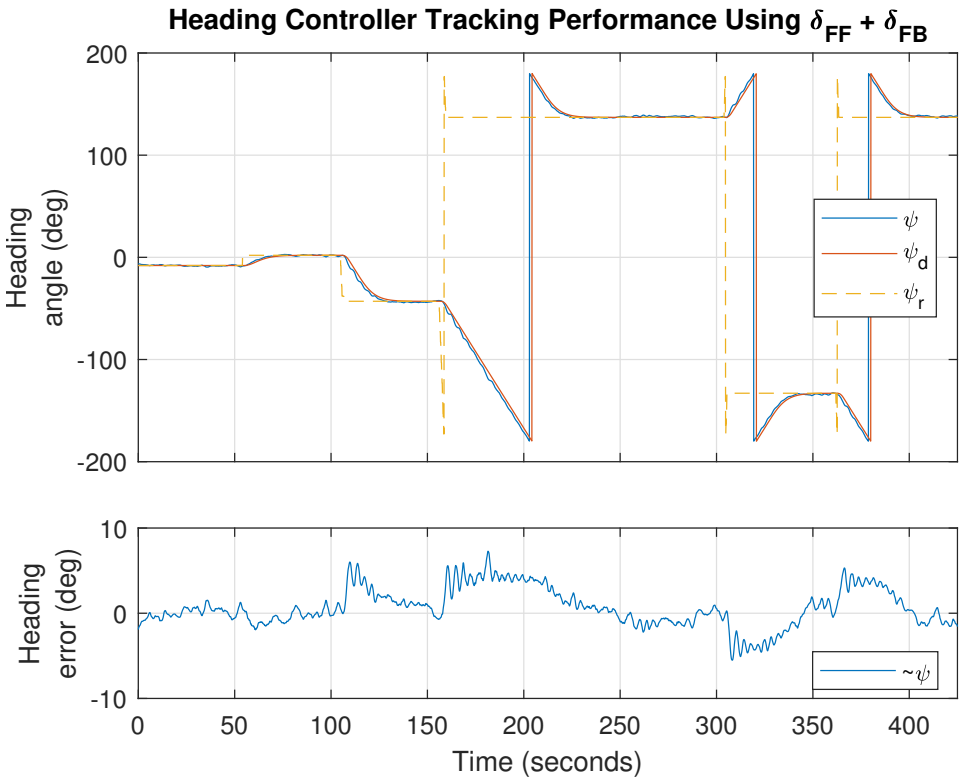


Figure 6.8: Heading controller tracking performance using model-based feedforward and PID feedback as control input.

In Figure 6.8, the same parameters as in Table 6.2 is used, except with integral gain $K_i = -1.0$.

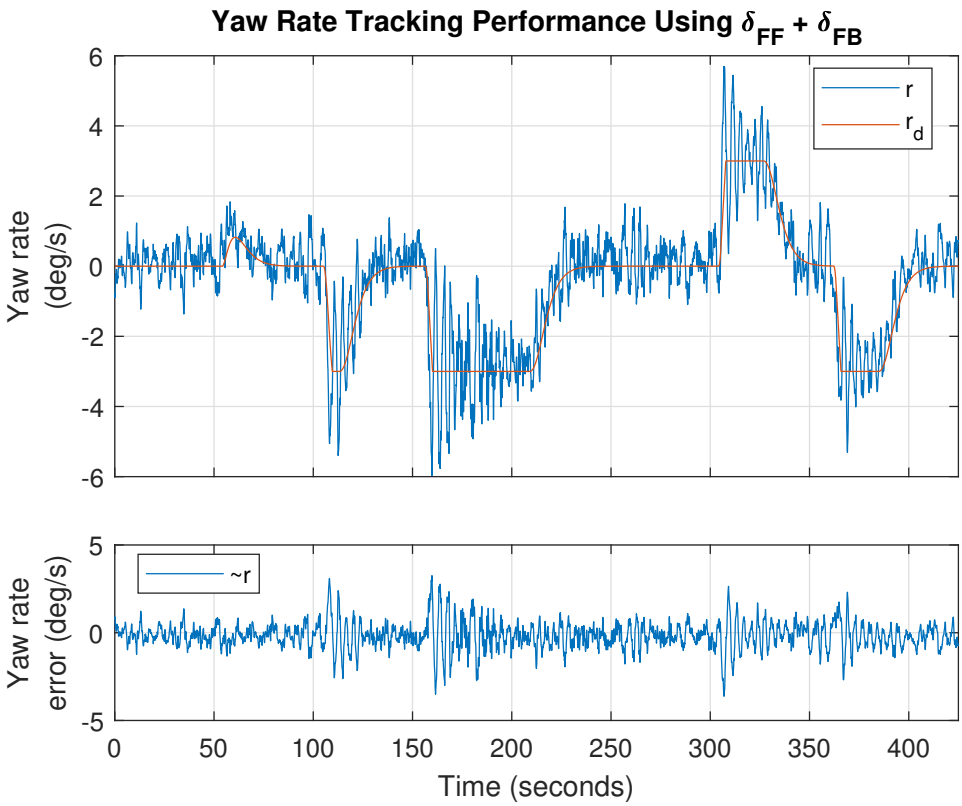


Figure 6.9: Yaw rate tracking performance using model-based feedforward and PID feedback as control input.

Figure 6.9 shows the yaw rate tracking response with added integral action.

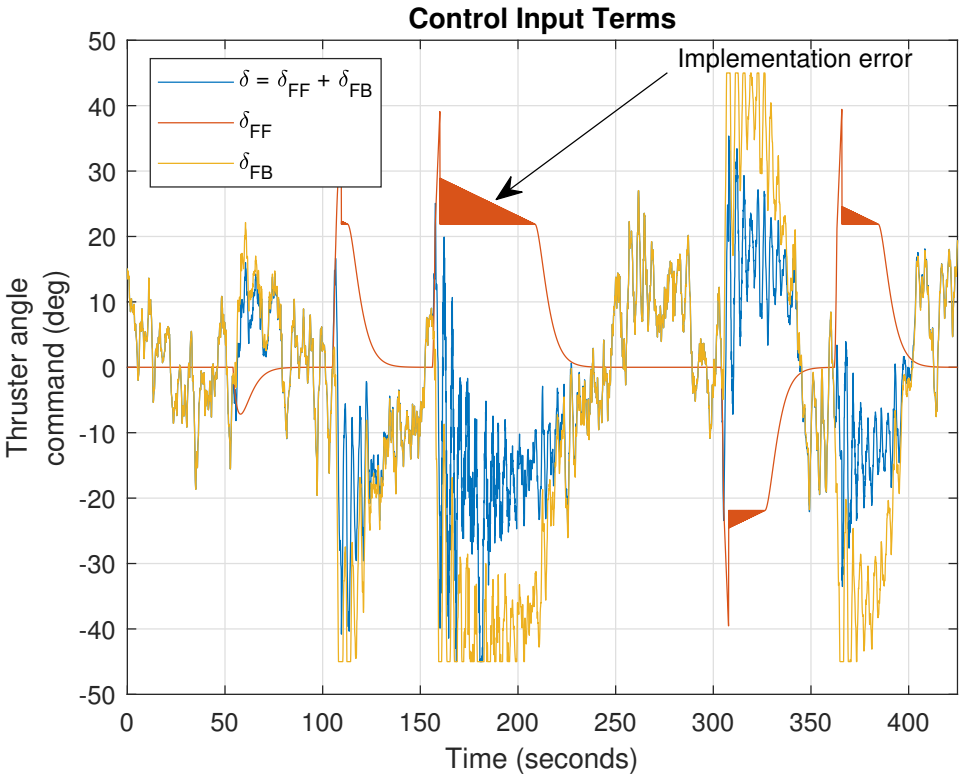


Figure 6.10: Thruster angle commands generated by the model-based feedforward and PID feedback term. The implementation errors are the red triangles, caused by repeated incrementation of r_d ($r_d > r_{max}$) at time step k and saturating again at time step $k+1$ ($r_d = r_{max}$). The error is resolved in LOS experiments.

Figure 6.10 shows the control input terms. The implementation error causes the ripples which appear as a triangle in the figure.

6.4 LOS Guidance Experimental Performance

The experimental results for the LOS Guidance System developed in Section 2.5, with the speed and heading controller developed in Sections 4.4 and 4.5 is presented here.

Date: May 28th, 2018

Weather conditions:

- Cloudy, 19.8^{o}C
- Wind, 2.8 m/s (light breeze) from southwest
- No waves, weak ocean current

The following results are from a single experiment. Table 6.3 shows the parameters used

LOS Guidance Parameters [m]	
Δ	9
R_{k+1}	16

Table 6.3: LOS Guidance parameters.

in the LOS Guidance algorithm. Note the changes from the simulations in Table 5.6 and that sideslip compensation is not used, due to no ocean currents present. The lookahead distance is decreased, i.e. more aggressive convergence. The distance from waypoint before switching occurs is increased, i.e. more time to approach next path.

Heading Controller Parameters and Gains	
K_p	-10.000
K_i	0.000
K_d	-5.000
ω_n	0.600
ζ	1.000
K	-0.136
T	2.117

Table 6.4: Heading controller parameters and gains for the LOS Guidance experimental test.

Chapter

Parameter	Value
K_p	100
K_i	7.5
M	800
$\sigma(u_d)$	see (4.34)

Table 6.5: Speed controller parameters and gains for the LOS Guidance experimental test.

Tables 6.4 and 6.5 lists the parameters and gains in the heading and speed controller.

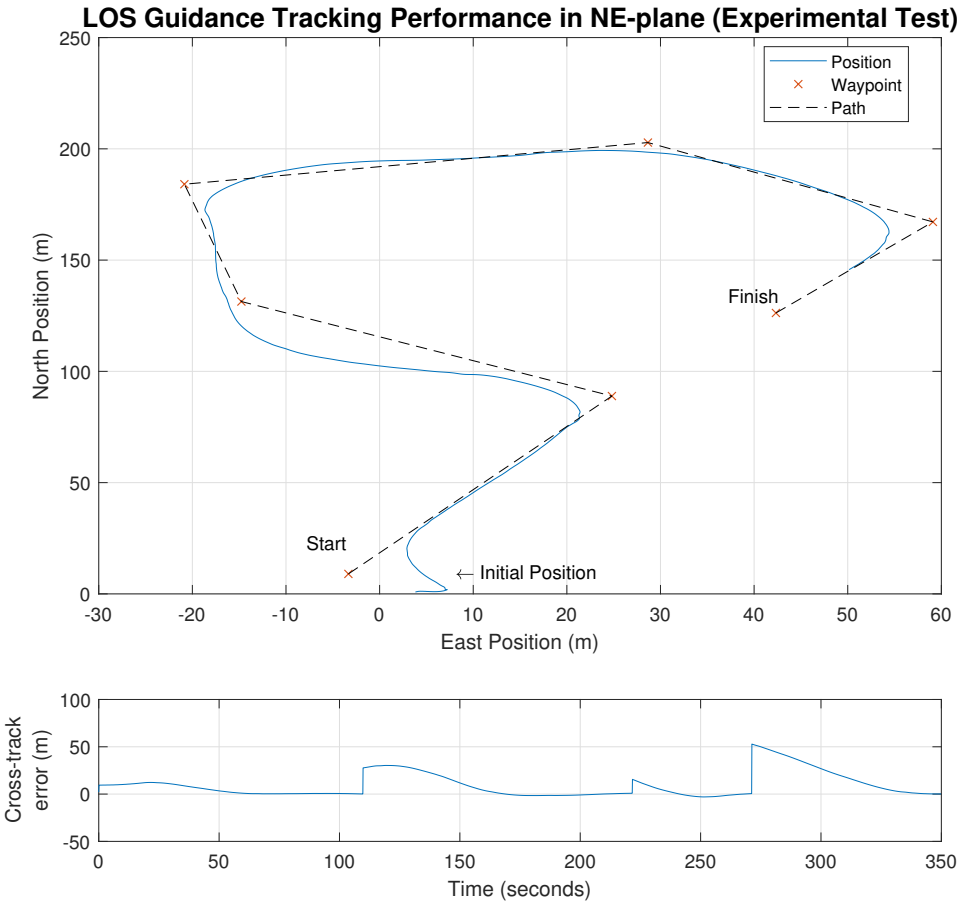


Figure 6.11: Result from the experimental test of LOS Guidance. Due to signal loss some waypoints where skipped.

In Figure 6.11, the path tracking performance is shown. In the cross-track error term there is only four convergence trajectories whereas the plot of positions infer that there should be six (number of straight-line segments). An implementation error in ReVolt's control system caused the algorithm to increment tracked waypoints to fast, if Wi-Fi signal strength is low at the time of waypoint switching. From "Start"=1, the path between waypoint 2-3 and between 5-6 are skipped. Regardless, the cross-track error converges to zero for every tracked straight-line segment

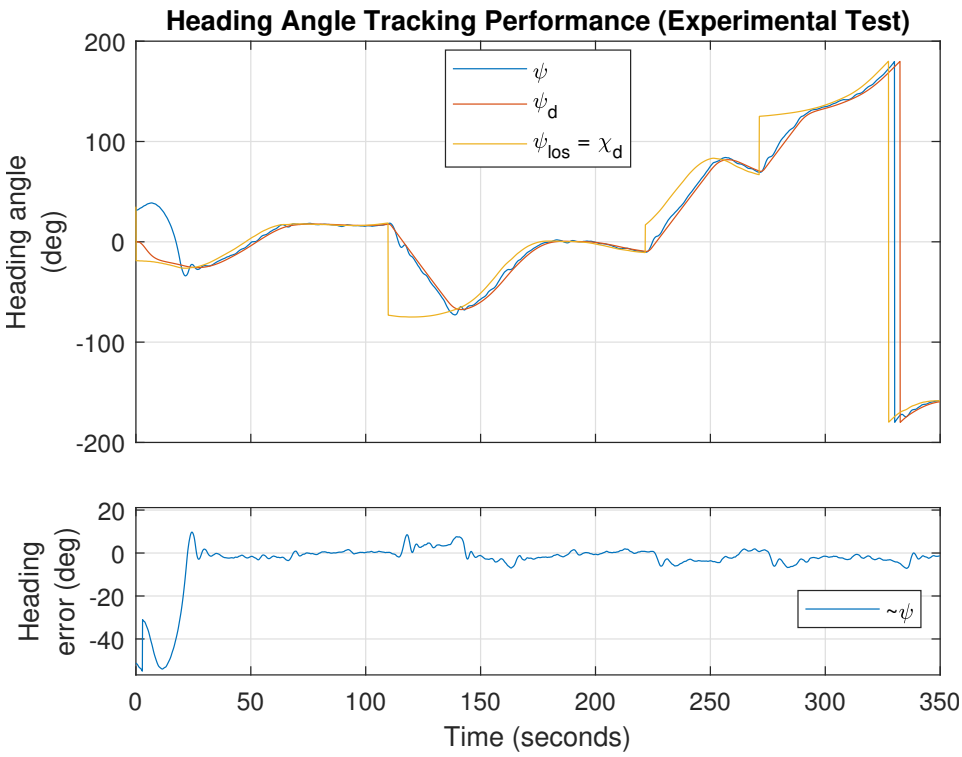


Figure 6.12: Heading angle tracking performance during LOS Guidance experimental test without sideslip compensation.

Initially in Figure 6.12, the heading reference model's initial conditions deviate from ReVolt's initial condition. This causes a large error which saturates the actuators. At $t \approx 25$ ReVolt's heading ψ converges to and tracks the desired heading ψ_d.

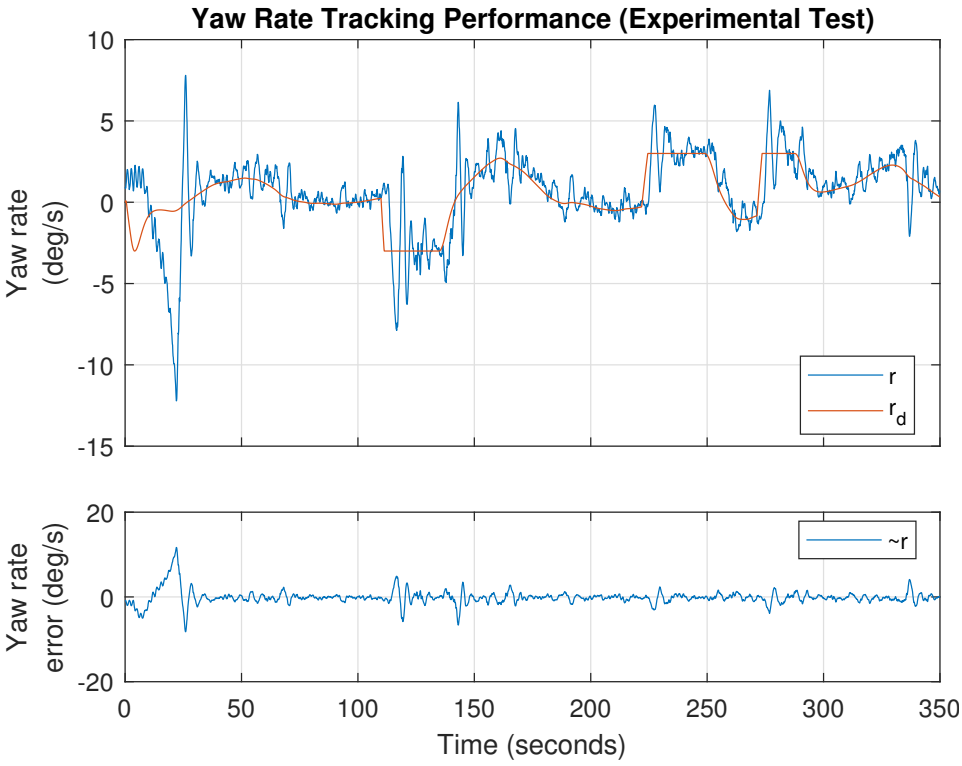


Figure 6.13: Yaw rate tracking performance during LOS Guidance experimental test.

The saturation in the actuators causes the large deviation in yaw rate in Figure 6.13. As with the heading controller experiments, the yaw rate tracks the desired yaw rate in the same manner.

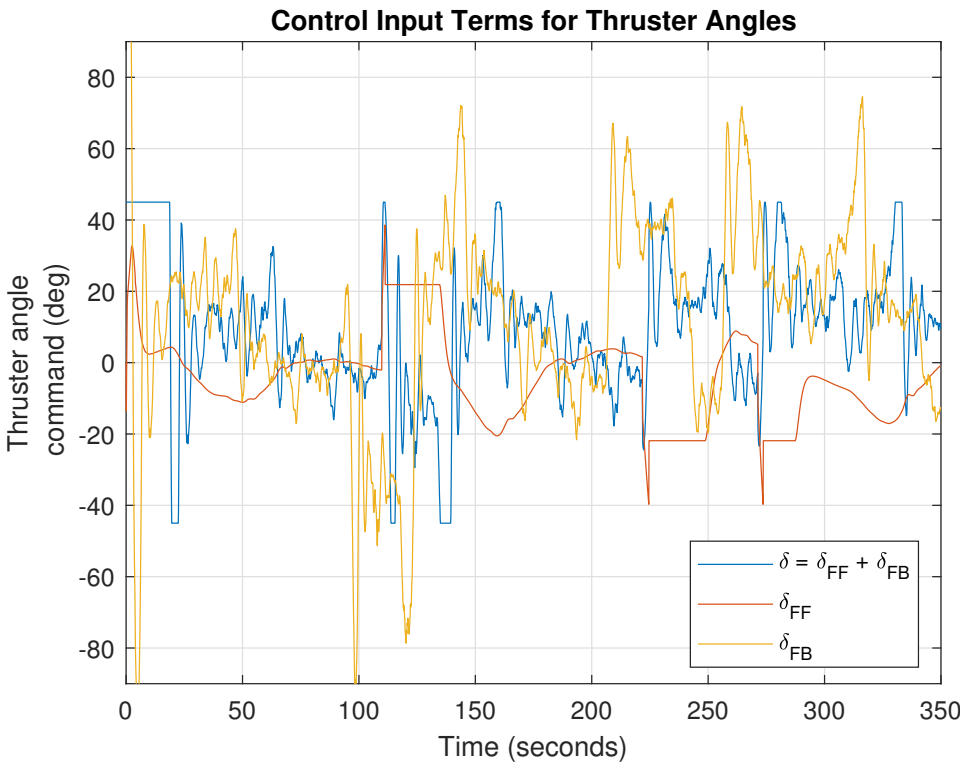


Figure 6.14: Thruster angle commands generated by model-based feedforward and PD feedback.

Due to the differential in initial conditions between ReVolt and the heading reference model, the large feedback term in Figure 6.14 accounts for the saturation at $\pm 45^o$ in δ and inherently the larger-than-desired yaw rate in Figure 6.13 as thrust builds up. Notice that the implementation error shown in Figures 6.7 and 6.10 from Section 6.3 causing the ripples in δ_{FF} is removed.

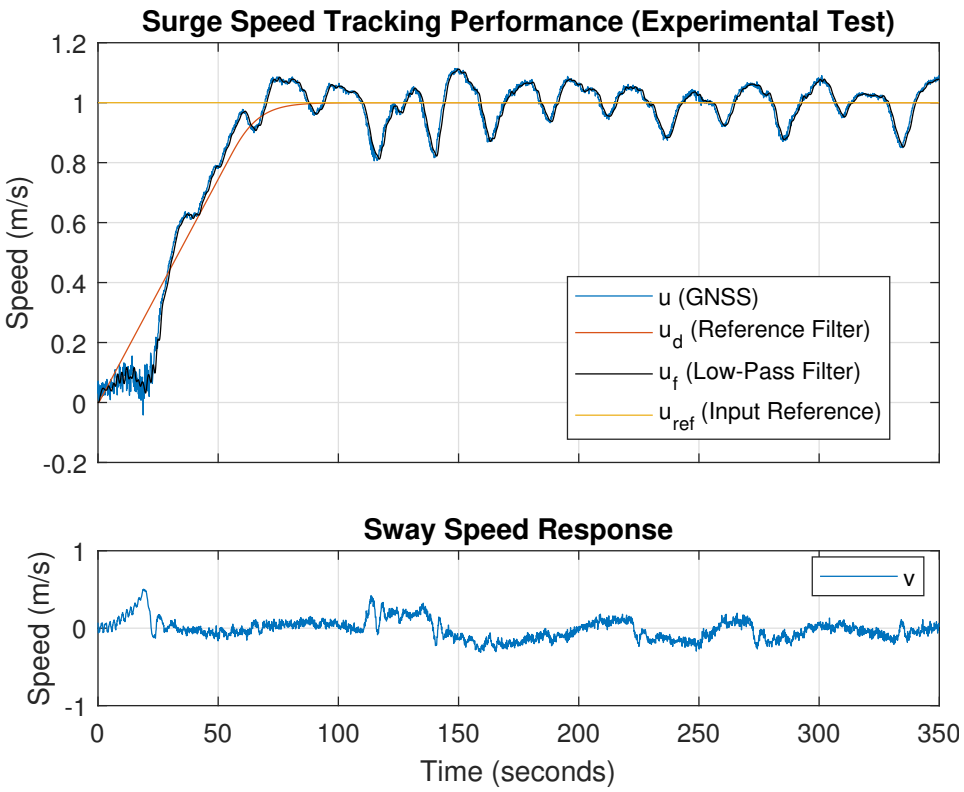


Figure 6.15: Surge Speed Controller tracking performance during LOS experimental test.

Figure 6.15 shows the speed controller tracking performance during the Guidance System for path following test. A few things to note here is that friction in the thruster bearings causes no rotation for $\tau_m < 15$ and that the initial thruster angle is saturated at $\delta = -45^o$, due to initial conditions of heading reference model. This contributes to the lower-than-desired surge speed and large sway speed (see bottom of Figure 6.15). At $t \approx 25$ the vessel stabilizes and starts to gain surge speed with a large integral term resulting overshoot at $t \approx 30$. The oscillations from $t \approx 100$ is perhaps caused by to large feedback gains as the current is weak (see Section 5.3.2).

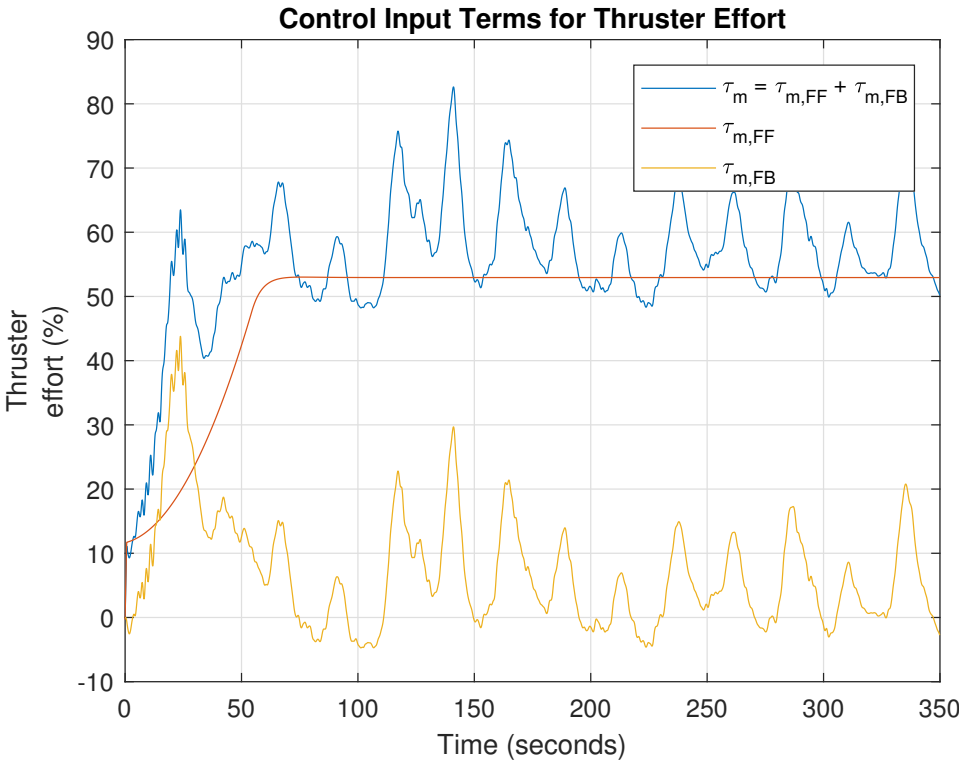


Figure 6.16: Control input terms for surge motion.

As shown in Figure 6.16, it takes approximately 10 seconds before an effort $\tau_m > 15$ is produced by combined FF and FB.

6.5 Discussion

An isolated speed controller test is not performed like in the simulations, as setting a thruster angle command $\delta = 0^o$ causes unstable behavior (see Section 7.5). The only speed controller test performed is during the LOS Guidance experiment. Friction in the bearings of the thrusters could be compensated by processing the damping term $\sigma(u_d)$ from Section 4.4.4 to exclude the area of $\sigma(u_d) < 15$. Also, the feedback gains for the speed controller using during LOS Guidance is set for much stronger currents than present during the experiment (see Section 5.3.3). A reduction might just eliminate the oscillating response in Figure 6.15.

The heading controller with a combined FF and FB term resulted in good tracking both with and without integral action, despite including the reference model implementation error shown in Figures 6.7 and 6.10. This error has been since fixed. The heading controller gains used in all experimental tests were not the ones found in Section 4.5.6. Given more time these would be used in the next experiments. Moreover, experimental tests using a FB term only would also help assess the heading controllers performance.

The Guidance System tracks the waypoints (it sees) well using the heading controller with a reference model, FF and FB term. Sideslip compensation is omitted as ocean current is weak. Another experiment with sideslip compensation active is desired to compare results. The implementation error that caused to fast iteration through list of waypoints was hard to discover and reproduce using the simulator, as Wi-Fi connection was perfect on land. Sadly, once it was discovered there was no time to implement a fix and do a re-run.

Chapter 7

Discussion

This chapter presents a discussion regarding the overall comparison between simulations and experiments as well as the development and implementation from the preceding chapters.

7.1 Experimental vs Simulation Results

Simulations were performed on ReVolt's digital twin which has been updated with the towing tank data from SINTEF Ocean, and added a virtual rudder for stability. The controllers performed well for all simulations. It is a fact that the experimental tests requires considerably more planning and effort to execute properly. Though some challenges regarding battery life, implementation errors discovered at sea and differences between the simulated and physical model, the overall results of the experimental tests are useful.

7.2 Heading Controller

The new **heading controller** was developed to increase the robustness of the low-level controllers used in LOS Guidance in the present, and with Collision Avoidance (COLAV) in the future. The previous heading controller consisted solely of a PD feedback term with low-pass error filtering. It became apparent after LOS simulations that a noisy thruster angle command was produced for time-varying setpoints (see Appendix D.1).

The new heading controller uses wrapping functions to allow a discontinuous signal from an operator or LOS algorithm like $\langle -\pi, \pi \rangle$ to be mapped to a continuous signal $\langle -\infty, \infty \rangle$ and used in the 3.order linear reference model with state saturation. The continuous

signals generate smooth trajectories for ψ_d, r_d and $\dot{r}_d$ that the vessel can follow. Furthermore, the continuous desired heading is mapped back to $\langle -\pi, \pi \rangle$ to be used in the control system.

A 1.order Nomoto model is used to determine the mass and damping during course changing maneuvers such that feedforward control can be utilized for better trajectory tracking. The Nomoto model, along with bandwidth $\omega_b > 0$, is also used the select PID feedback gains and reference model natural frequency $\omega_n > 0$ with damping factor $\zeta = 1$.

However, it lacks a few sophisticated features such as *Wave Filtering*, which avoids 1.order wave forces entering the control loop. This is not considered crucial, as the test area is protected from waves. A wind feedforward would increase response time during course-changing maneuvers subject to wind disturbance, as opposed to just integral action. A wind sensor and an accurate model of the wind force is required for this.

Lastly, gain scheduling of the tuning parameters in the linear reference model should be implemented as a current step change of 1 takes the same time to reach as a step of e.g. 10.

7.3 Speed Controller

The new **surge speed controller** was developed to enable ReVolt to maintain a desired forward speed while following a predefined path. In the future, when ReVolt has the ability to sense its environment, the speed controller should receive setpoint corrections from the COLAV system to prevent collisions if necessary.

A 2.order (velocity) reference model generates the desired states u_d and $\dot{u}_d$. Furthermore, a 2.order damping polynomial $\sigma(u_d)$ is produced from steady-state measurements with the simulator, which takes a desired surge speed as input and outputs a thruster effort as a part of the feedforward. This damping term could be improved by performing the same system identification tests at sea, as the simulator is not identical to the actual vessel. Moreover, further processing of $\sigma(u_d)$ is necessary to overcome friction in the bearings at lower thruster efforts when performing experiments at sea (see Section 4.4.4).

The last part of the feedforward is the inertia term Mu_d. The mass factor M in the inertia term is constant in the control system while dependent on states in practice. This assumption contributes to the offsets in the speed controller simulations in Section 5.3.1 (see Figure 5.13) which are corrected by a proportional feedback with low-pass filtering of the surge speed in Section 5.3.2 (see Figure 5.15). Integral action with integral anti-windup is added to compensate for ocean currents.

7.4 Guidance System

The Guidance System was developed to enable ReVolt to follow a predefined path defined by straight-line segments between user-placed waypoints in the RMC station. The

lookahead-based steering principle was chosen as it required less computing power, as opposed to enclosure-based steering. The algorithm outputs a LOS heading reference ψ_{los} to the heading controller and an operator sets the speed over ground reference U_{ref} to the speed controller. The Guidance System contains reference models, which are discussed in the sections above due to the implementation (see Section 4.1).

7.5 Digital Twin

A major advantage of using the Digital Twin is that it interacts with ReVolt's actual control system. This means the source code does not need to be translated from e.g. Matlab/Simulink, but can be written directly in C++ (or Python) and tested in real time. The downside of the real time testing was that some simulations took a lot of time to finish, and if an error occurred with the control system, debugging and restart was required. Despite the virtual rudder added to the Digital Twin, the experimental results shows promise.

7.6 RMC Station

The contributions to the RMC station in this thesis are based on the work done in the specialization project. During that period the RMC station was never tested at sea. The simulations and experimental tests performed at sea have shown that the RMC station can be used for remote monitoring and control. It was, however, only tested with Wifi. This has limitations in form of signal loss at larger distances. This is especially apparent in the live image stream. A 4G sim-card for both ReVolt and the RMC station is obtained, but needs to be set up.

The waypoint placement, path and footprint drawing worked satisfactory. Regardless, further improvements to the interactive map would improve the user-experience. For instance, editing waypoints by dragging them around the map instead of deleting and placing them again. Also, no constrains are put on placing the waypoints. Such constraints could be distance and angle between waypoints or only allowing waypoints to be placed in water.

Since ReVolt should have a Collision Avoidance System in the future, a class for creating instances of `Obstacle` and drawing the current state of the obstacles in the navigation map is implemented, but needs to be improved and tested properly. For now, testing has only been done using manually written geodetic positions transmitted from ReVolt. Obstacles needs to be generated by the simulator and transmitted to ReVolt first. From there, the obstacles states must be transmitted to the RMC station for visualization. Later, when ReVolt can sense its environment, ReVolt should track the obstacles on its own at sea.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Two low-level controllers have been developed for ReVolt. A new heading controller for turning, and a surge speed controller for forward speed. Both controllers use reference models for generating desired states for use in feedback and feedforward control. The control allocation uses the two azimuth thrusters constrained at $\pm 45^o$, while the bow thruster remains retracted during transit.

Furthermore, a LOS Guidance System for path following has been added to ReVolt's control system. This enables ReVolt to follow a predefined path independent of time by issuing reference values to the heading and speed controllers for convergence towards the path. The heading controller also improves manual control of ReVolt using the RC remote.

The RMC station from the specialization project has been further developed to support waypoint generation by selecting positions in the navigation map. The list of waypoints is transmitted to ReVolt using a TCP socket connection where the path is generated. Additional improvements to the RMC station includes eliminating the occurring image distortion. Moreover, the RMC station has been used during both simulations and experimental results at sea for the first time.

Simulations using the Digital Twin and control system has been performed to assess the tracking performance of each controller separately. For the heading controller, a FF only tracks the desired trajectories well, with a slight steady state error when no disturbances is present. The FB only obtains no steady-state error with a slight delay. A FF-FB combination showed the best performance despite begin subject to wind disturbance.

The speed controller required a FF and Proportional-FB to obtain satisfactory tracking due to some modeling errors in the damping and inertia term. Moreover, the Guidance System

for path following have been assessed with overall very good performance using combined FF-FB for both controllers.

Similar experimental tests for path following, including heading (FF-FB) and speed controller (FF-FB), and a separate heading controller test have been performed in the Dora harbor basin in Trondheim. A FB term was necessary to stabilize ReVolt and though few differences between simulator and physical model, the experimental performance was good.

8.2 Future Work

Collision Avoidance System

Expanding the GNC System with a reactive collision avoidance algorithm, e.g. *Velocity Obstacles*, is the next step towards making ReVolt autonomous. Today, ReVolt is in the process of acquiring sensing ability using camera and LIDAR fusion. Until this process is complete the DNV GL's simulator for ReVolt can provide simulated moving obstacles to avoid.

Control System Robustness

Here is a list of suggestions to improving the control system implementation:

- Merge contributions from branch *guidance-system* into *master*.
- Implement the $U_{ref} \rightarrow u_{ref}$ transformation in `GuidanceLawNode`
- Reset waypoint iterator after a LOS run in `GuidanceLawNode`*
- Select suitable control mode (e.g. DP) after finishing a LOS run.
- Create an implicit waypoint $\#0$ from initial position to waypoint $\#1$ for more predictable initial convergence.
- Add constraints to waypoint placements (distance, angle, in water etc.)
- Initialize both reference models with vessel's initial state.*
- Allow speed controller feedback term to induce negative effort to stop vessel more rapidly.
- Use parameter gain scheduling for the heading reference model to increase response time to smaller step changes.
- Allow speed controller and setting a manual thruster effort to be used interchangeably in heading control mode when controlling from RMC station (model initial condition important here).
- Add setpoint change to speed controller in RMC station.

* The ROS environment provides *services* which provides request/reply interactions between *nodes* (as opposed to *broadcast* with publisher/subscriber). This allows calls to user-defined methods based on user-defined conditions or requests in `rqt_reconfigure` during run-time.

Bibliography

[1] K. Alfheim H. & Muggerud, "Development of a dynamic positioning system for the revolt model ship," Norwegian University of Science and Technology, Department of Engineering Cybernetics, Tech. Rep., 2017.

[2] (2011). Rolls royce, ship intelligence, [Online]. Available: http://www.rolls-royce.com/~/media/Files/R/Rolls-Royce/documents/customers/marine/ship-intel/rr-ship-intel-aawa-8pg.pdf.

[3] A. Lekkas and T. Fossen, "Line-of-sight guidance for path following of marine vehicles," Tech. Rep., Jun. 2013.

[4] E. Peymani and T. I. Fossen, "2d path following for marine craft: A least-square approach," *IFAC Proceedings Volumes*, vol. 46, no. 23, pp. 98 –103, 2013, 9th IFAC Symposium on Nonlinear Control Systems, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20130904-3-FR-2041.00095. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016316421.

[5] M. Breivik, "Nonlinear maneuvering control of underactuated ships," Tech. Rep., Jun. 2003.

[6] E. Lefeber, K. Y. Pettersen, and H. Nijmeijer, "Tracking control of an underactuated ship," *IEEE Transactions on Control Systems Technology*, vol. 11, no. 1, pp. 52–61, 2003, ISSN: 1063-6536. DOI: 10.1109/TCST.2002.806465.

[7] L. Liu, D. Wang, Z. Peng, and H. Wang, "Predictor-based los guidance law for path following of underactuated marine surface vehicles with sideslip compensation," *Ocean Engineering*, vol. 124, pp. 340 –348, 2016, ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2016.07.057. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0029801816303031.

[8] A. Havnegjerde, "Remote monitoring & control of an autonomous boat," Tech. Rep., Dec. 2018.

[9] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, 2011.

[10] T. Fossen. (2016). Marine craft hydrodynamics: Kinematics, [Online]. Available: http://www.fossen.biz/wiley/Ch2.pdf (visited on 05/19/2018).

[11] O. Egeland and J. T. Gravdahl, *Modeling and Simulation for Automatic Control*. Marine Cybernetics, 2002, ISBN: 9788292356012. [Online]. Available: https://books.google.no/books?id=oK0VAAAACAAJ.

[12] M. Breivik and T. Fossen, "Guidance laws for autonomous underwater vehicles," Jan. 2009.

[13] (). Webopedia osi model, [Online]. Available: https://www.webopedia.com/quick_ref/OSI_Layers.asp (visited on 04/20/2018).

[14] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th. Pearson, 2012, ISBN: 0132856204, 9780132856201.

[15] Unversity of California, Berkeley. (). Practical c++ sockets, [Online]. Available: http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/practical/ (visited on 04/20/2018).

[16] W. Stallings, *Operating Systems: Internals and Design Principles*, 6th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008, ISBN: 0136006329, 9780136006329.

[17] EngineerJobs.com. (2013). Robot operating system (ros), [Online]. Available: https://magazine.engineerjobs.com/2013/robot-operating-system.htm (visited on 04/03/2018).

[18] IEEE Spectrum. (). The origin story of ros, the linux of robotics, [Online]. Available: https://spectrum.ieee.org/automaton/robotics/robotics-software/the-origin-story-of-ros-the-linux-of-robotics (visited on 04/23/2018).

[19] ROS - Robot Operating System. (). Node description, [Online]. Available: http://wiki.ros.org/Nodes (visited on 04/20/2018).

[20] ——, (). Topics description, [Online]. Available: http://wiki.ros.org/Topics (visited on 04/20/2018).

[21] ——, (). Message description, [Online]. Available: http://wiki.ros.org/Topics (visited on 04/23/2018).

[22] CartoType. (2017). Cartotype sdk, [Online]. Available: http://www.cartotype.com/developers/evaluation-sdks (visited on 05/23/2018).

[23] ——, (2017). Cartotype public repository. https://github.com/CartoType/CartoType-Public-4-4, (visited on 05/23/2018).

[24] (2011). Estimate flat earth position from geodetic latitude, longitude and altitude, mathworks, [Online]. Available: http://se.mathworks.com/help/aerotbx/ug/lla2flat.html?requestedDomain=www.mathworks.com.

[25] M. Breivik, V. E. Hovstein, and T. Fossen, "Straight-line target tracking for unmanned surface vehicles," pp. 131–149, Oct. 2008.

[26] B.-O. H. Eriksen and M. Breivik, "Modeling, identification and control of high-speed asvs: Theory and experiments," in *Sensing and Control for Autonomous Vehicles: Applications to Land, Water and Air Vehicles*, T. I. Fossen, K. Y. Pettersen, and H. Nijmeijer, Eds. Cham: Springer International Publishing, 2017, pp. 407–431, ISBN: 978-3-319-55372-6. DOI: 10.1007/978-3-319-55372-6_19.

[Online]. Available: https://doi.org/10.1007/978-3-319-55372-6_19.

[27] miniDPS. (2018). Fir vs iir filtering, [Online]. Available: https://www.minidsp.com/applications/dsp-basics/fir-vs-iir-filtering.

Appendices

Appendix A

Excerpt from the ReVolt Source Code

A.1 Heading Controller Constructor

```
HeadingController::HeadingController(ros::NodeHandle nh){
  /* Reference Model Init */
  const double zeta = 1;
  const double omega_n = 0.6;
  const double c = -pow(omega_n,3);
  const double b = -(2*zeta+1)*pow(omega_n,2);
  const double a = -(2*zeta+1)*omega_n;

  A_d << 0, 1, 0,
         0, 0, 1,
         c, b, a;
  B_d << 0, 0,-c;

  m_x_d << 0, 0, 0;

  r_max = 3*D2R;
  r_dot_max = 1*D2R;

  /* Wrapping function variables */
  accumulation = 0;
  state = 0;
  psi_last = 0;
```

```
  /* Nomoto Model parameters */
  K = -0.1371;
  T = 2.4457;

  /* Feedback variables and parameters*/
  double m = T/K;
  double d = 1/K;
  K_p = m*pow(omega_n,2);
  K_i = omega_n*K_p/10.0;
  K_d = (2*zeta*omega_n*m - d);
  psi_tilde = 0;
  r_tilde = 0;
  ROS_INFO("Heading␣Controller␣Gains:
␣␣␣␣Kp␣=␣%f,␣Ki␣=␣%f,␣Kd␣=␣%f", K_p, K_i, K_d);

  /* Init publishers */
  controlInputPub = nh.advertise<std_msgs::Float64>(
   "heading_controller_output",  1);

  /*Init subscribers */
  stateSub = nh.subscribe("heading_controller_input",
   1, &HeadingController::computeControlInput, this);

  // Code for logging, methods and services omitted
}
```

A.2 Heading Controller Callback Function

```
void HeadingController::computeControlInput(const
 custom_msgs::HeadingControllerInput &input){

  Vector3d x_d;
  // Calculate dt
  if(!prev_time.isZero()){
    dt = ros::Time::now() - prev_time;
    prev_time = ros::Time::now();
  }
  else{
    // init prev_time
    prev_time = ros::Time::now();
    return;
  }

  // Continuous signal for the reference filter [-Inf Inf]
  double psi_continuous = wrapToInf(input.setpoint*D2R);
  x_d = referenceFilter(psi_continuous);
  // Wrap desired heading from filter back to [-pi pi]
  x_d(0) = wrapToPi(x_d(0));

  // Compute feedforward term
  delta_ff = -((T/K)*(x_d(2) + (1/T)*x_d(1)));

  // Compute feedback term
  double psi_tilde = x_d(0) -input.state*D2R;
  double r_tilde = x_d(1) - input.r;

  if(psi_tilde > M_PI)
    psi_tilde = psi_tilde - 2*M_PI;
  else if(psi_tilde < -M_PI)
    psi_tilde = psi_tilde + 2*M_PI;

  psi_tilde_integral += psi_tilde*dt.toSec();

  // Integrator anti-windup (max 20 deg from integrator)
  if(K_i*psi_tilde_integral > 20.0*D2R)
    psi_tilde_integral = 20.0*D2R/K_i;
  else if(K_i*psi_tilde_integral < -20.0*D2R)
    psi_tilde_integral = -20.0*D2R/K_i;

  double delta_fb = -(K_p*psi_tilde +
   K_i*psi_tilde_integral + K_d*r_tilde);
```

```
  if(delta_fb != delta_fb){
    ROS_INFO("nan in fb");
    return;
  }

  if(delta_ff != delta_ff){
    ROS_INFO("nan in ff");
    return;
  }

  // Fill message : FF + FB
  controlInput.data = (delta_ff + delta_fb)*R2D;

  if(controlInput.data != controlInput.data){
    ROS_INFO("nan in control input");
    return;
  }

  // Saturate angle
  if(controlInput.data > 45)
    controlInput.data = 45;
  else if(controlInput.data < -45)
    controlInput.data = -45;

  controlInputPub.publish(controlInput);
  // Code for logging, methods and services omitted
}
```

A.3 Speed Controller Constructor

```
SpeedController::SpeedController(ros::NodeHandle nh){
  // PI parameters and variables
  Kp = 100;
  Ki = 7.5;
  u_tilde_integral = 0.0;

  // Define velocity reference model parameters
  const double zeta = 0.9994;
  const double omega_n = 0.1583;

  const double a = 2*zeta*omega_n;
  const double b = pow(omega_n,2);

  // State space matrices for filter (member fields)
  A <<  0,  1,
       -b, -a;

  B <<  0, b;

  x << 0, 0;

  // Initialize [b, a] filter coefficients

  u_f = 0.0;
  // Init subscribers, publishers and services
 stateSub = nh.subscribe("speed_controller_input",
  1, &SpeedController::computeControlInput, this);
  // Code for logging, methods and services omitted
}
```

A.4 Speed Controller Callback Function

```
void SpeedController::computeControlInput(const
 custom_msgs::SpeedControllerInput &input){
  // Calculate dt
  if(!prev_time.isZero()){
    dt = ros::Time::now() - prev_time;
    prev_time = ros::Time::now();
  }
  else{
    // init prev_time
    prev_time = ros::Time::now();
    return;
  }

  // Filter surge velocity
  u_f = measurementFilter(input.u);

  // Apply reference velocity to 'Reference Filter'
  //  to obtain u_d (desired speed) at index [0]
  std::vector<double> u_d_vector =
   referenceFilter(input.u_ref);
  double u_d = u_d_vector[0];
  double u_d_dot = u_d_vector[1];

  // Map u_ref to thruster effort n_ff and use
  //  it as feedforward
  double n_ff = mapToEffort(u_d) + 800*u_d_dot;

  // Surge error variable    = u_d - u_f
  double u_tilde = u_d - u_f;
  // Integrate error
  u_tilde_integral += u_tilde*dt.toSec();

  // Integrator anti-windup
  if(u_tilde_integral > 100.0/Ki)
    u_tilde_integral = 100.0/Ki;
  else if(u_tilde_integral < -100.0/Ki)
    u_tilde_integral = -100.0/Ki;

  // Fill message : FF + FB)
  controlInput.data = n_ff
   + (Kp*u_tilde + Ki*u_tilde_integral);

  // Limit output
```

```
if(controlInput.data > 100)
  controlInput.data = 100;
else if(controlInput.data < 0)
  controlInput.data = 0;

// Publish total control input
controlInputPub.publish(controlInput);

// Code for logging, methods and services omitted
```

A.5 Guidance Law Constructor

```
GuidanceLaw::GuidanceLaw(ros::NodeHandle nh){
  // Init BODY velocity vector
  v_b = Vector2d::Zero();
  // Init NED velocity vector
  p_n_dot = Vector2d::Zero();
  // Init Rotation Matrix
  R_psi << cos(0.0), -sin(0.0),
           sin(0.0),  cos(0.0);

  f = (SEMIMAJORAXIS - SEMIMINORAXIS) / SEMIMAJORAXIS;
  R = SEMIMAJORAXIS;


  speed_controller_input_pub = nh.advertise<
   custom_msgs::SpeedControllerInput>(
    "speed_controller_input", 1);
  heading_controller_input_pub = nh.advertise<
   custom_msgs::HeadingControllerInput>(
    "heading_controller_input", 1);
  guidance_law_input_sub = nh.subscribe(
   "guidance_law_input",
     1, &GuidanceLaw::computeOutput, this);
  waypoint_sub = nh.subscribe("waypoint_list",
   1, &GuidanceLaw::processWaypoints, this);
  ned_sub = nh.subscribe("ned_origin",
   1, &GuidanceLaw::updateNEDOrigin, this);

  // k - waypoint iterator
  k = 0;
  yk=0;
  xk=0;
  yk_1=100;
  xk_1=100;
}
```

A.6 Guidance Law Callback Function

```
void GuidanceLaw::computeOutput(const
 custom_msgs::GuidanceLawInput &input){
  // Create messages
  custom_msgs::HeadingControllerInput heading_msg;
  custom_msgs::SpeedControllerInput speed_msg;


  // Decompose Speed over ground U with angle chi and
  // update NED velocity vector
  // velocity in north direction
  p_n_dot(0) = input.U*cos(input.chi*D2R);
  // velocity in east direction
  p_n_dot(1) = input.U*sin(input.chi*D2R);

  // Update 2D Rotation matrix
  updateRotationMatrix(input.psi*D2R);

  // Transform vector from NED to BODY
  v_b = R_psi.transpose()*p_n_dot;

  // Lookahead distance (12m)
  double delta_e = 12;

  // Calculate path tangential angle chi_p/a_k
  double a_k = atan2(yk_1 - yk, xk_1 - xk);

  Vector2d p_tilde;
  p_tilde << input.N-xk, input.E-yk;

  Matrix2d R_ak;
  R_ak << cos(a_k), -sin(a_k),
          sin(a_k), cos(a_k);


  // Calculate cross-track error
  Vector2d se = R_ak.transpose()*p_tilde;
  // Calculate LOS steering law
  double chi_r = atan2(-se(1), delta_e);

  // Calculate desired course chi_d
  double chi_d = a_k + chi_r;

  // Calculate sideslip beta
```

```
  double beta;
  if(input.U < 0.2)
    beta = 0;
  else
    beta = asin(v_b(1)/input.U);

  // GuidanceLaw Node outputs desired heading
  double psi_d = chi_d - beta;
  if(psi_d > M_PI)
    psi_d = psi_d - 2*M_PI;
  else if(psi_d < -M_PI)
    psi_d = psi_d + 2*M_PI;

  heading_msg.state = input.psi;
  heading_msg.setpoint = psi_d*R2D;
  heading_msg.error = psi_d*R2D - input.psi;
  heading_msg.r = input.r;
  heading_msg.source = 1;
  speed_msg.u = v_b(0);
  speed_msg.v = v_b(1);
  speed_msg.u_ref = 1;


  // Publish to topics
  heading_controller_input_pub.publish(heading_msg);
  speed_controller_input_pub.publish(speed_msg);

  // Check to see if increment in setpoints is necessary
  double sk_1 = sqrt(pow(xk_1-xk, 2) + pow(yk_1 - yk, 2));

  if(sk_1 - se(0) < 16.0/*Rk_1*/){
    k+=2;
  }
  // Code for logging, methods and services omitted
}
```

Appendix B

Excerpt from the RMC Station Source Code

Figure B.1: Class diagram from [8] with new contributions in boldface text.

B.1 Add/Remove Waypoints

```
void MapForm::addWaypointToList(int32_t aY, int32_t aX){
  CartoType::TPoint p(aX, aY);
  error = iFramework->ConvertPoint(p,
   CartoType::TCoordType::Display,
    CartoType::TCoordType::Degree);
  // Convert integer coordinates to regular decimal
  //  coordinates as we add them to the list
  m_waypoint_list.push_back(pair<double,
   double>(p.iY/CONVERSION, p.iX/CONVERSION));
  emit waypointListChanged();
  navMapUpdate();
}

bool MapForm::removeLastWaypoint(){
  if(!m_waypoint_list.empty()){
    m_waypoint_list.pop_back();
    emit waypointListChanged();
    navMapUpdate();
    return true;
  }
else
  return false;
}
```

B.2 Draw Lines Between Waypoints

```
void MapForm::drawLinesBetweenWaypoints(QImage &chart){
  if(m_waypoint_list.empty())
    return;

  std::list<pair<double,double>>::iterator it;
  QPainter painter(&chart);
  QPen linePen(Qt::black, 2, Qt::DashLine);
  QPen waypointPen(Qt::red, 7);
  painter.setPen(linePen);
  int waypointNumber = 1;
  it = m_waypoint_list.begin();
  // Draws lines from FIRST to LAST position in the list
  while(it!=m_waypoint_list.end()){
    // Extract first pair of coordinates from list
    int32_t integer_latitude1 = (*it).first*CONVERSION;
    int32_t integer_longitude1 = (*it).second*CONVERSION;
    CartoType::TPoint wp1(integer_longitude1,
     integer_latitude1);
    iFramework->ConvertPoint(wp1,
    CartoType::TCoordType::Degree,
     CartoType::TCoordType::Screen);

    // Draws the last waypoint in the list and returns
    if(std::next(it,1) == m_waypoint_list.end()){
      painter.setPen(waypointPen);
      painter.drawPoint(wp1.iX,wp1.iY);
      painter.setPen(Qt::black);
      painter.drawText(wp1.iX,wp1.iY,
      QString::number(waypointNumber));
      return;
    }

    // Otherwise
    it++;

    // Extract next pair from list
    int32_t integer_latitude2 = (*it).first*CONVERSION;
    int32_t integer_longitude2 = (*it).second*CONVERSION;
    CartoType::TPoint wp2(integer_longitude2,
     integer_latitude2);
    iFramework->ConvertPoint(wp2,
     CartoType::TCoordType::Degree,
      CartoType::TCoordType::Screen);
```

```
    // Draw waypoins, waypoint numbers and lines between
    // the current pair
    painter.setPen(waypointPen);
    painter.drawPoint(wp1.iX,wp1.iY);
    painter.drawPoint(wp2.iX,wp2.iY);
    painter.setPen(Qt::black);
    painter.drawText(wp1.iX,wp1.iY,
     QString::number(waypointNumber));
    painter.drawText(wp2.iX,wp2.iY,
     QString::number(waypointNumber+1));
    painter.setPen(linePen);
    painter.drawLine(wp1.iX, wp1.iY, wp2.iX, wp2.iY);
    waypointNumber++;
  }
}
```

B.3 Draw ReVolt's Footprint

```
void MapForm::drawRevoltFootprint(QImage &chart){
  // For the case of a single position (No
  //  line can be drawn)
  if(m_position_list.size() < 2 )
    return;

  QPainter footprintPainter(&chart);
  QPen footprintPen(QColor(0, 170 , 0), 2, Qt::SolidLine);
  footprintPainter.setPen(footprintPen);

  std::list<CartoType::TPoint>::iterator it;
  it=std::prev(m_position_list.end(),1);

  // Draw lines from LAST to FIRST position in the list
  while(it!=std::prev(m_position_list.begin(), 1)){
    CartoType::TPoint current((*it).iX, (*it).iY);
    iFramework->ConvertPoint(current,
     CartoType::TCoordType::Degree,
      CartoType::TCoordType::Screen);

    // At final iteration
    if(it == m_position_list.begin())
      return;

    // Otherwise get previous point
    it--;
    CartoType::TPoint previous((*it).iX, (*it).iY);
    iFramework->ConvertPoint(previous,
     CartoType::TCoordType::Degree,
      CartoType::TCoordType::Screen);
    footprintPainter.drawLine(current.iX,
     current.iY, previous.iX, previous.iY);
  }
}
```

B.4 Draw Obstacles

```
void MapForm::drawObstacles(QImage &chart){
  if(m_obstacle_list.empty())
    return;

  QPainter obstaclePainter(&chart);
  QPen radiusPen(QColor(255, 170, 0), 2);
  QPen obstaclePen(QColor(255, 170, 0),
   iFramework->MetersToPixels(3));
  obstaclePainter.setPen(radiusPen);

  std::list<Obstacle>::iterator it;
  const int REVOLT_RADIUS = 5;
  for(it=m_obstacle_list.begin();
   it!=m_obstacle_list.end(); it++){
    std::vector<int32_t> temp =
     (*it).getObstaclePositionINT();
    const int OBS_RADIUS = (*it).getRadius();
    CartoType::TPoint obs(temp[1], temp[0]);
    iFramework->ConvertPoint(obs,
     CartoType::TCoordType::Degree,
      CartoType::TCoordType::Screen);

    obstaclePainter.drawEllipse(QPoint(obs.iX,obs.iY),
     iFramework->MetersToPixels(OBS_RADIUS),
      iFramework->MetersToPixels(OBS_RADIUS));
    obstaclePainter.drawEllipse(QPoint(obs.iX,obs.iY),
     iFramework->MetersToPixels(OBS_RADIUS+REVOLT_RADIUS),
      iFramework->MetersToPixels(OBS_RADIUS+REVOLT_RADIUS));
    obstaclePainter.setPen(obstaclePen);
    obstaclePainter.drawPoint(obs.iX, obs.iY);
  }
}
```

Appendix C

Images From Experimental Tests

Figure C.1: Transport stage with drogue connected to ReVolt's aft.

Figure C.2: Gunnerus Workboat(left), ReVolt and Nidelv 690 Sport. Courtesy of Tom Arne Pedersen.

Figure C.3: Transport stage without drogue.

Figure C.4: ReVolt on the trailer at the unloading area.

Appendix D

Miscellaneous

D.1 Previous Heading Controller for LOS Guidance Simulation

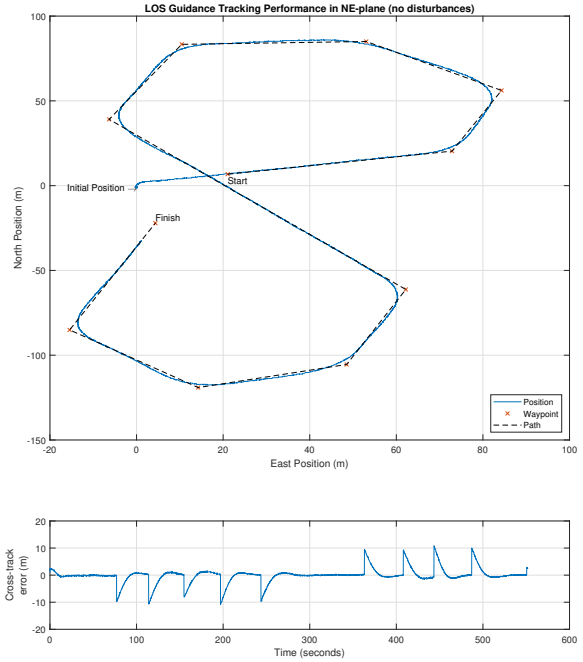


Figure D.1: LOS simulation using the old heading controller

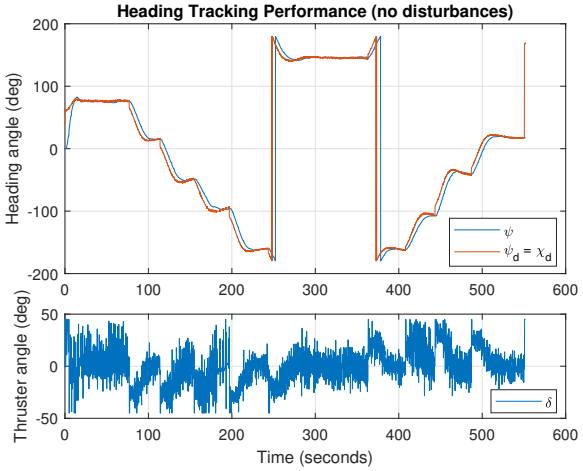


Figure D.2: LOS simulation using the old heading controller causing noisy control output (no reference filter or FF)

D.2 Excerpt from the Towing Tank at SINTEF Ocean

Tests and processing were performed during the DNV GL summer internship in 2017.

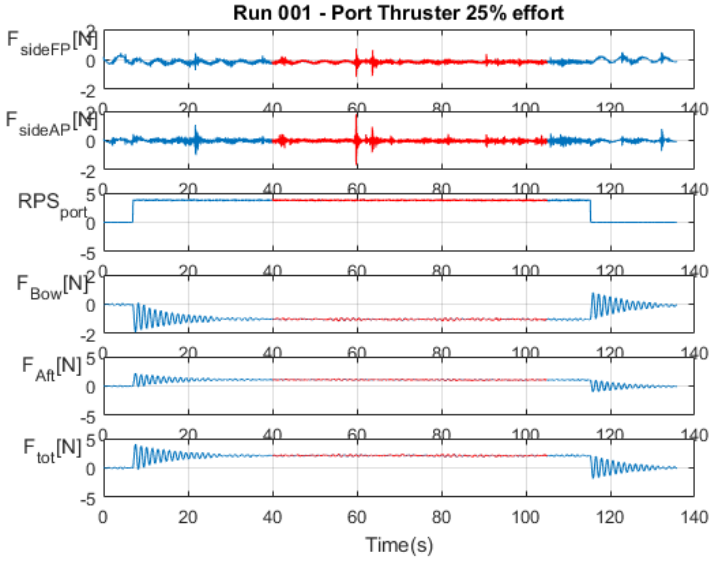


Figure D.3: Towing tank results at SINTEF Ocean with port thruster at 25% effort.

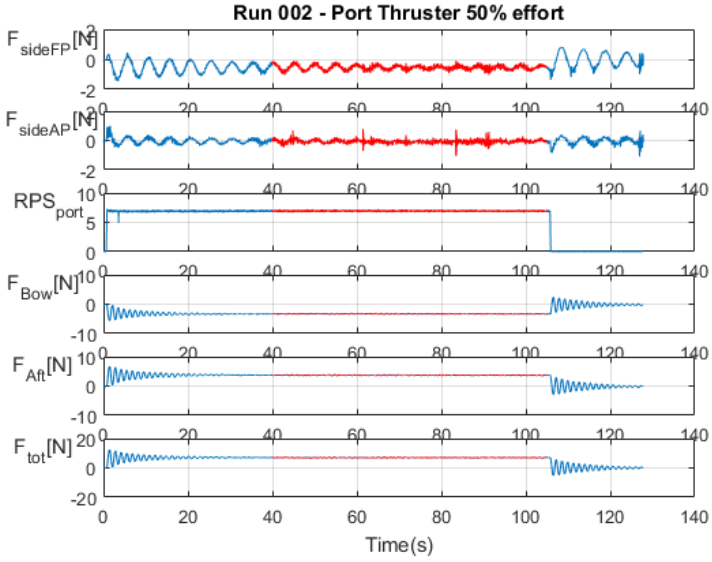


Figure D.4: Towing tank results at SINTEF Ocean with port thruster at 50% effort.

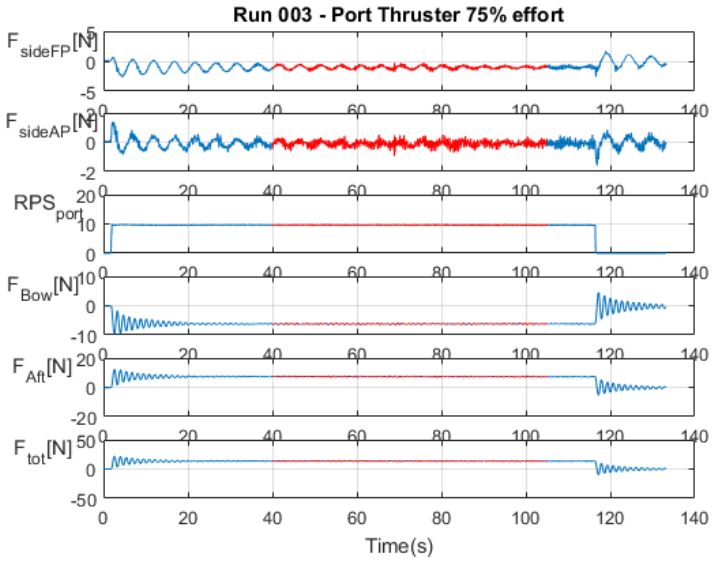


Figure D.5: Towing tank results at SINTEF Ocean with port thruster at 75% effort.

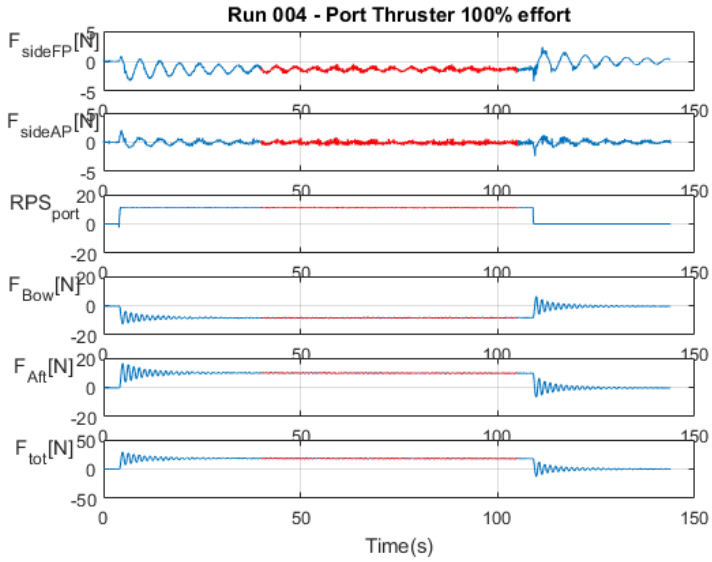


Figure D.6: Towing tank results at SINTEF Ocean with port thruster at 100% effort.

In Figures D.3, D.4, D.5 and D.6, the F_{tot} is the total force produced by the thruster, combining both thrusters yields $2 \times F_{tot}$. Red markings is steady state area.

D.3 Velocity Low-pass Filter Coefficients

i	b_i	a_i
0	0.000109	1
1	0.000546	-3.840964
2	0.001092	6.007554
3	0.001092	-4.766603
4	0.000546	1.914322
5	0.000109	-0.310815

Table D.1: Velocity low pass filter coefficients