Master's thesis

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

Stian André Elvenes

Development of C/S Jonny with optimal constrained thrust allocation

Master's thesis in MTMART
Supervisor: Roger Skjetne
Co-supervisor: Emir Cem Gezer
June 2023

NTNU
Norwegian University of
Science and Technology

Stian André Elvenes

Development of C/S Jonny with optimal constrained thrust allocation

Master's thesis in MTMART
Supervisor: Roger Skjetne
Co-supervisor: Emir Cem Gezer
June 2023

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

NTNU
Norwegian University of
Science and Technology

Stian Andre Elvenes

Development of C/S Jonny with optimal constrained thrust allocation

Master's thesis in marine technology
Supervisor: Roger Skjetne
Co-supervisor: Emir Cem Gezer
June 2023

Norwegian University of Science and Technology
Department of Marine Technology

NTNU

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER OF TECHNOLOGY THESIS DEFINITION (30 SP)

Name of the candidate:	Stian Andre Elvenes
Field of study:	Marine cybernetics
Thesis title (Norwegian):	Utvikling av C/S Jonny med optimal beskranket propellkraftsallokering
Thesis title (English):	Development of C/S Jonny with optimal constrained thrust allocation

Background

Our cybership model C/S Enterprise I (CSE1) has served us well for many years after it was built in 2011. However, CSE1 is starting to experience wear and tear, and it has had some weaknesses with respect to its Voith-Schneider thrusters and challenging control-to-thrust mapping. Hence, this project aims to develop a new cybership ASV for MC-Lab, temporarily called C/S Jonny (CSJ), built on the same uBuntu/ROS platform that now exist for all our cybership models in MC-Lab.

Our cybership models in MC-Lab need an improved robust default thrust allocation algorithm and improved low-level actuator controls – going beyond pseudoinverse-based methods. An additional goal of this project is therefore to study background literature on local thruster control and constrained numerical optimization algorithms for thrust allocation.

The objective of the thesis is to develop and finalize the CSJ model and implement thrust allocation algorithms for the CSJ and the cybership fleet.

Scope of Work

1. Perform a background and literature review to provide information and relevant references on:
   - Cybership developments for MC-Lab, such as CS2, CS3, CSS, CSE1, and CSAD.
   - ROS architecture and ROS-based control on uBuntu.
   - Thruster dynamics and thrust mappings for different types of thrusters, typical constraints, and local thruster control.
   - Relevant optimal constrained thrust allocation methods (beyond pseudoinverse).

   Write a list with abbreviations and definitions of terms and symbols, relevant to the literature study and project report.
2. Develop the CSJ to be ready for deployment in the MC-Lab. Document the building process. Make and perform a "basin trials" test program for completion, incl. joystick control.
3. Control system design:
   - Make drawings that describe:
     - Control system hardware architecture.
     - Power system single line diagram, power flow, voltage levels, converters, etc.
     - Communication signal/network information flow.
     - ROS software topology, i.e., ROS nodes covering the basic functions.
   - Adapt and fit the ROS control system of CSE1 and/or CSAD for CSJ.
4. Perform thrust-mapping of C/S Jonny, and implement algorithm to map force to control signal.
5. Implement thrust allocation algorithms for the CSJ and tentatively the rest of the cyber fleet, e.g.:
   - Optimal thrust allocation for marine vessels (Sørdalen 1997).
   - Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming (Johansen et al 2004).
   - Constrained control allocation for vessels with azimuth thrusters (Scibilia & Skjetne 2012).
   - "*Optimal thrust allocation is a maneuvering problem*" by Skjetne (2022).
6. Define test scenarios to evaluate the thrust allocation algorithms performance. Conduct both simulation testing and physical model-scale testing of the algorithms.

Specifications

Every weekend throughout the project period, the candidate shall send a status email to the supervisor and co-advisors, providing two brief bulleted lists: 1) work done recent week, and 2) work planned to be done next week.

The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, problem/research statement, design/method, analysis, and results. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 70 A4-pages, 100 B5-pages, from introduction to conclusion, unless otherwise agreed. It shall be written in English (preferably US) and contain the elements: Title page, project definition, preface (incl. description of help, resources, and internal and external factors that have affected the project process), acknowledgement, abstract, list of symbols and acronyms, table of contents, introduction (project background/motivation, objectives, scope and delimitations, and contributions), technical background and literature review, problem formulation or research question(s), method/design/development, results and analysis, conclusions with recommendations for further work, references, and optional appendices. Figures, tables, and equations shall be numerated. The contribution of the candidate shall be clearly and explicitly described, and material taken from other sources shall be clearly identified. Chapters/sections written together with other students shall be explicitly stated at the start of the chapter/section. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. natbib Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct, which is taken very seriously by the university and will result in consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed.

The thesis shall be submitted with an electronic copy to the main supervisor and department according to NTNU administrative procedures. The final revised version of this thesis definition shall be included after the title page. Computer code, pictures, videos, data, etc., shall be included electronically with the report.

Start date: 15 January, 2023 **Due date:** As specified by the administration.

Supervisor: Roger Skjetne
Co-advisor(s): Emir Cem Gezer

Signatures:

Roger Skjetne

Digitally signed by rskjetne
Date: 2023.06.08 09:35:42 +02'00'

ABSTRACT

This thesis presents the development and building process of the new marine cybernetics laboratory vessel C/S Jonny.

C/S Jonny is built as a 1:32 scale model of a tug boat. It has two azimuth thrusters and a bow thruster powered by two 4-cell LiPo batteries. A Raspberry Pi running ROS is used to control the system. The model is made to be modular with plug-and-play capabilities by creating a control box for housing most of the electronic components and LEMO plugs for the connections of parts. A watertight deck has been installed with two access hatches. For tracking, the model is equipped with four IR spheres to be used with the Qualisys tracking system. A basin trial has been performed to verify that the model is functional. It showed the model is working well, however the control signal for the azimuth thrusters is relatively coarse, which needs to be addressed for optimal operation.

The control signals of the thrusters have been mapped to produced forces. The results have been used to create an algorithm to map force to control signals. This has been showed to work adequately.

Four constrained thrust allocation algorithms have been reviewed, implemented, and tested. They have been shown to work as desired and are ready for use on the new model. Furthermore, two have been selected for implementation for the rest of the C/S fleet.

SAMMENDRAG

Denne oppgaven presenterer utviklingen og byggeprosessen gjort for det nye laboratoriefartøyet til MC-lab kalt C/S Jonny.

C/S Jonny er bygget som en 1:32 skala modell av en slepebåt. Den har to azimut thrustere og en baugpropell som får strøm av to 4-cellers LiPo-batterier. En Raspberry Pi som kjører ROS brukes til å kontrollere systemet. Modellen er laget for å være modulær med plug-and-play funksjoner. Dette er oppnådd ved å lage en kontroll boks for å huse de fleste elektroniske komponentene og ved bruk av LEMO tilkoblinger. Det er også montert ett vanntett dekk med to tilgangsluker. For å måle posisjonen av modellen har den blitt utstyrt med fire IR-kuler som skal brukes sammen med Qualisys sporingssystemet. Det er utført ett bassengforsøk for å verifisere at modellen er funksjonell. Denne testen viste at modellen fungerer bra, men det kom også frem av resultatene at styresignalet for asimut-thrusterne er relativt grov, noe som må løses for at modellen skal fungere optimalt.

Styresignalene til thrusterne er kartlagt til produserte krefter. Resultatene har deretter blitt brukt til å lage en algoritme for å omgjøre kraft til kontroll signaler. Dette har vist seg å fungere bra.

Fire optimale beskrankete propellkraftsallokerings algoritmer har blitt presentert, implementert og testet. De har vist seg å fungere som ønsket og er nå klare til bruk på den nye modellen. Videre er to av disse valgt for å bli brukt videre på resten av «Cybeship» flåten.

PREFACE

This thesis was written during the spring of 2023 as part of a master of science degree in marine cybernetics at NTNU. The work has been done over two semesters and has proven to be a challenging and rewarding project. Through the project, I have acquired new skills, such as 3D modeling, 3D printing, soldering, and knowledge about thrusters and thrust allocation. Through the completion and building of C/S Jonny, there have been a few hiccups, such as parts stuck in customs and a short-circuited Raspberry Pi. The biggest hiccup, however, was when, at the start of May, right at the start of the physical testing of the vessel, one of the thrusters was destroyed by a bug in the program. The replacement parts arrived on the 26 of May, leaving only two short weeks for the rebuild of the thruster and all of the physical testing. However, through several late nights at the MC lab and copious amounts of coffee, the thrusters were rebuilt and tests performed.

I want to thank:

- Professor Roger Skjetne for his guidance and for creating and showing me a new method for thrust allocation.
- Ph.D. candidate Emir Cem Gezer for his help with answering questions on quadratic programming
- Senior Engineer Robert Oppland for all the help in the development and building process of C/S Jonny.

CONTENTS

Abstract i

Sammendrag ii

Preface iii

Contents vi

List of Figures vi

List of Tables viii

Abbreviations x

1 Introduction 1
1.1 Motivation and objective . . . 1
1.2 Contributions . . . 2

I Building C/S Jonny 3

2 Background 4
2.1 Marine Cybernetics Laboratory . . . 4
2.1.1 Equipment . . . 4
2.1.2 Qualisys motion-capture . . . 5
2.1.3 Cybership fleet . . . 5
2.2 Cyber-physical testing . . . 7
2.3 Robot Operating System . . . 8
2.4 Pulse width modulation . . . 8
2.5 Kinematics . . . 9

3 Requirement specification 11
3.1 Requirements Specifications . . . 11
3.1.1 Intended use . . . 11
3.1.2 Main requirements . . . 11
3.1.3 Equipment and instruments . . . 12

4 Development of C/S Jonny 13
4.1 Parts . . . 13
4.2 Assembly of bowthruster . . . 13
4.2.1 Installation of the thruster . . . 13

4.2.2 Installation of motor . . . 14
4.3 Assembly of azimuth thrusters . . . 14
4.3.1 Creation of mounting plate . . . 14
4.3.2 Installation of mounting plate . . . 15
4.3.3 Converting thruster to 360° rotation . . . 15
4.4 IMU . . . 17
4.5 Electronic box . . . 17
4.6 Battery holder . . . 18
4.7 Painting . . . 19
4.8 Deck . . . 19
4.9 IR spheres . . . 20
4.10 Ballasting . . . 20

5 System architecture 22
5.1 Vessel systems . . . 22
5.1.1 thruster configuration . . . 22
5.1.2 IMU . . . 22
5.1.3 IR spheres . . . 23
5.2 Power system single-line diagram . . . 23
5.3 Communication signal . . . 23
5.4 Software topology . . . 24

6 Basin trial 27
6.1 Expermiental setup . . . 27
6.2 Results . . . 27
6.2.1 Body-fixed frame . . . 28
6.2.2 Basin-fixed frame . . . 28

7 Discussion & further work 31
7.1 Discussion . . . 31

II Thrust allocation for C/S jonny 33

8 Background 34
8.1 Thrusters and thruster dynamics . . . 34
8.2 Mathematical notation and thruster configuration matrix . . . 35
8.2.1 Polar configuration matrix . . . 35
8.2.2 Rectangular configuration matrix . . . 36
8.3 Thrust allocation problem . . . 37
8.4 Maneuvering problem . . . 37

9 Thrustmapping 39
9.1 experimental setup . . . 39
9.2 mapping . . . 40
9.2.1 Azimuth thruster . . . 40
9.2.2 Bow thruster . . . 42
9.3 Force to PWM . . . 43

10 Thrust allocation **45**
10.1 Pseudoinverse with filtering . . . 45
10.1.1 controller . . . 45
10.2 Quadratic programming . . . 46
10.2.1 Quadratic programming . . . 47
10.2.2 Sequential quadratic programming . . . 47
10.3 Maneuvering . . . 48
10.3.1 controller . . . 48
10.4 Results and discussion . . . 50
10.4.1 Constraints . . . 50
10.4.2 Computation times . . . 57
10.5 Realtime implementation . . . 58

11 Conclusions & further work **61**
11.1 Conclusion . . . 61
11.1.1 Development of C/S Jonny . . . 61
11.1.2 Thrust mapping . . . 61
11.1.3 Thrust allocation . . . 61
11.2 Further work . . . 62

References **63**

Appendices: **67**

A - Pseudoinverse with filtering **68**

A - QP Johansen **71**

A - QP Scibilla **74**

A - Manuvering **78**

B - Parts **83**

C - Attachments **90**

LIST OF FIGURES

2.1.1 Towing carriage. Courtesy: NTNU (2023). 5
2.1.2 CSEI. 6
2.1.3 CSAD. 7
2.1.4 CSS. 7
2.3.1 ROS architecture concept. 8
2.4.1 PWM signal courtesy: EEPOWER (2023) 9
2.5.1 6DOF courtesy: Fossen (2011) 10

4.2.1 Installation process for bow thruster 14
4.2.2 Mounting bracket for bow thruster 14
4.3.1 Mounting plate, top view, and underside 15
4.3.2 Installation of mounting plate 15
4.3.3 Conversion to 360° rotation 16
4.3.4 Printed azimuth mount with all components 17
4.4.1 IMU box 17
4.5.1 Electronic box inlay 18
4.5.2 Electronic box 18
4.6.1 Battery holder 18
4.7.1 Painting. 19
4.8.1 Deck 19
4.8.2 Deck installation 19
4.9.1 IR installation 20
4.10.1 Water line guide courtesy Aeronaut (2022) 20
4.10.2 Waterline 20
4.10.3 Ballasting 21

5.1.1 Thruster configuration of CSJ. 22
5.2.1 Single line diagram of CSJ's power system. 23
5.3.1 Communication signals of the CSJ. 24
5.4.1 Software topology models MC-lab courtesy: Roger Skjetne 25
5.4.2 The implemented software topology 25

6.1.1 Basin trial 27
6.2.1 x and y position for basin trial with body-fixed joystick controller 28
6.2.2 Heading and Comanded tau for basin trial with body-fixed joystick controller 28
6.2.3 x and y position for basin trial with basin-fixed joystick controller 29
6.2.4 Heading and Comanded tau for basin trial with basin-fixed joystick controller 29

7.1.1 Measurement from port force sensor 31

9.1.1 Bollard pull setup for C/S Jonny 39
9.2.1 0° starboard measurements 40
9.2.2 Mapping of starboard thruster 41
9.2.3 Comparison between starboard and port azimuth thruster 42
9.2.4 Bow thruster map 42

10.1.1 Slow varying angles 45
10.4.1 Saturation constraint comparison tau 51
10.4.2 Saturation constraint comparison force of thrusters 51
10.4.3 Saturation constraint comparison angles 52
10.4.4 Rate constraints comparison tau 53
10.4.5 Rate constraints comparison force of thruster 54
10.4.6 Rate constraints comparison angles 54
10.4.7 Angle constraint comparison tau 55
10.4.8 Angle constraint comparison force of thrusters 56
10.4.9 Angle constraint comparison angle of thrusters 56
10.4.10 Computational time for each iteration 57
10.5.1 Realtime implementation pseudoinverse 58
10.5.2 Realtime implementation QP Johansen 59
10.5.3 Realtime implementation QP Scibilla 59
10.5.4 Realtime implementation maneuvering 60

LIST OF TABLES

2.1.1 Dimensions CSEI. Courtesy: Bjørnø (2015). 5
2.1.2 Dimensions CSAD. Courtesy: Bjørnø (2016). 6
2.5.1 SNAME-notation for 6DOF courtesy Fossen (2011) 10

3.1.1 Hull specifics of CSJ. 11

5.1.1 Thruster configuration. 22
5.1.2 Position of IMU 23
5.1.3 Position of IR spheres 23

10.4.1Constraints for the different algorithms. 50

NOMENCLATURE

Abbreviation

- **DOF:** Degerees of freedom
- **QTM:** Qualisys track manager
- **ROS:** Robotic operating system
- **C/S:** Cybership
- **CSEI:** Cybership enterprise
- **CSAD:** Cybership Arctic drill-ship
- **PWM:** Pulse width modulation
- **CSJ:** Cybership Jonny
- **LOA:** Length over all
- **ESC:** Electronic speed controler
- **IR:** Infrared
- **Mc-lab** Marine cybernetic laboratory

Symbols

n Rotational speed

ρ Density of water

K_t, K_q Thrust loss coefficients

H wave height

T_a produced thrust

Q_a produced torque

P_a power consumption

τ thrust load vector

f the twodimensional vector of the force

m moment

α thruster angle

$B(\alpha)$ polar thruster configuration matrix

B rectangular thruster configuration matrix

ξ Thruster force rectangular (or extended)

τ_{cmd} commanded thrust load vector

$B^{\dagger}$ pseudoinverse of B

$B_w^{\dagger}$ weighted pseudoinverse

θ parametrization variable

σ singular values of matrix

$W_i(F_i)$ power consumption function

W_i weighting matrix for power consumption

Q weighting matrix for the slack variable

s slack variable

Ω weighting matrix for angular constraint

ϵ small value to avoid division by 0

ϱ tuning parameter for singularity term

α_0 previous thruster angle

F^+ predicted force

τ predicted thrust load vector

τ predicted angular rate

s^+ predicted slack variable

P^+ weighting matrix for future power prediction

Ω^+ weighting matrix for future angular rate

Q^+ weighting matrix for future slack variable

F_{max} maximum thruster force

F_{min} minimum thruster force

α_{max} maximum thruster angle

α_{min} minimum thruster angle

$\Delta\alpha_{max}$ maximum thruster angle rate

$\Delta\alpha_{min}$ minimum thruster angle rate

ξ_p particular solution to ξ

W Weight matrix

γ steepest decent gain

μ gradient update law gain

ρ saftey gain to avoid saturation limits

$\bar{U}$ rate limit

CHAPTER
ONE

INTRODUCTION

1.1 Motivation and objective

As with any research field, the use of models and lab-scale experimentation serve a purpose. Be it risk mitigation, cost reduction, practicality, or all three. In this particular case, C/S Jonny's predecessor, C/S Enterprise, has experienced enough wear to no longer serve this purpose. Thus, CSJ was built to fulfill the same specifications, providing the ability to perform maneuvering-based tests in the narrow basin at the Marine cybernetic laboratory and to keep up with the demands of state-of-the-art research. The Cybership fleet also needs an improved thrust allocation algorithm that can handle physical constraints. The thesis aims to develop and finalize the CSJ model and implement thrust allocation algorithms for the CSJ and the cyber ship fleet.

1.2 Contributions

The main contribution of this thesis is to develop and build a new model for the Marine Cybernetics laboratory. The new vessel is made small to perform maneuvering tests in the basin at MC-lab. It is meant to replace C/S enterprise as it is experiencing wear and tear and some problems with the control to thrust mapping. The new model will be used for future research and student projects. In addition to the new model, this thesis aims to implement robust constrained thrust allocation algorithms to be used by all the models of the MC-lab. contributions are also given through

- A complete building guide of the new vessel
- A working vessel
- System drawings of the vessel
- Core functionality is implemented for the vessel
- Provide mapping from control signal to force for the vessel
- A function for mapping force to control signal is implemented
- Theory of different thrust allocation algorithms is presented
- Different thrust allocation algorithms are implemented and tested.

Part I

Building C/S Jonny

CHAPTER

TWO

BACKGROUND

The building of CSJ was performed in collaboration with Magnus Løvold Kvæbek .In this section, section 2.1-2.3 is from Elvenes et al. (2023), performed in the fall of 2022 as part of the pre-project.

2.1 Marine Cybernetics Laboratory

MC-Lab is a small wave basin. It was created from the old storage basin for the towing tank when models were made of paraffin wax and needed to be stored wet. Today the models are no longer made of paraffin wax, so the old storage basin was repurposed into the MC-lab in the 1990s. The basin has a length of 40m, a breadth of 6.45m, and a depth of 1.5 m (NTNU (2023)). Today the lab is mainly used by master's students and Ph.D. candidates to perform model tests.

2.1.1 Equipment

The laboratory has a suite of sensors and equipment to perform several types of tests. This includes tools to generate environmental forces, towing carriage, and a motion-capture system.

Wave maker

The MC-Lab is equipped with a 6m width single paddle wave maker located at the basin's short end. It can produce both regular and irregular waves with height and period (NTNU (2023)):

- Regular waves H $<$ 0.25, T = 0.3 – 3 s.
- Irregular waves Hs $<$ 0.15 m, T = 0.6 – 1.5 s.

It can create different wave spectra, such as JONSWAP and Pierson-Moskowitz.

Towing carriage

The lab is equipped with a precision towing carriage. This carriage enables hydrodynamic testing of small-scale models in 6 Degrees of Freedom (DOF). It has a maximum speed of $2[\frac{m}{s}]$ in the x direction, $1[\frac{m}{s}]$ in the y direction and $0.5[\frac{m}{s}]$ in the z direction. Figure 2.1.1 illustrates the orientation of the axes.

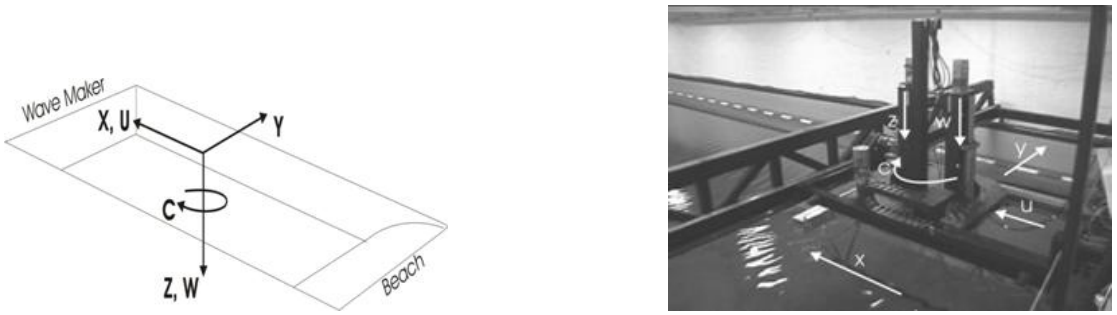


Figure 2.1.1: Towing carriage. Courtesy: NTNU (2023).

2.1.2 Qualisys motion-capture

To capture movement in 6DOF, the lab is equipped with the Qualisys motion capture system. This system consists of two parts. Firstly, the Oqus infrared cameras. They track infrared reflective spheres positioned on the models. Today the lab has three cameras, but it is currently being fitted with additional cameras for a larger tracking area. Secondly, the Qualisys Track Manager (QTM) is on a dedicated computer. It performs triangulation and broadcasts the position on the local network.

2.1.3 Cybership fleet

The MC-Lab today consists of several models to perform tests with. Initially, the models were created using NI CompactRIO modules, but later the fleet converted to Raspberry Pi's using ROS. The fleet has a tradition of being named with the prefix C/S, meaning Cybership.

C/S Enterprise In 2009 a model boat named Azis was purchased and built for the MC-Lab. The model was refitted with two Voith-Schneider propellers, new bow thrusters, and actuators (Skåtun (2011)). The upgraded model was renamed CS Enterprise 1 after the Star Trek series. The model has the dimensions shown in 2.1.1:

Table 2.1.1: Dimensions CSEI. Courtesy: Bjørnø (2015).

Description	Data
Displacement	$0.01479[m^3]$
Mass	$14.79[kg]$
Length	$1.10[m]$
Breadth	$0.24[m]$
Depth	$0.07[m]$

Today the model is primarily used for DP-testing, maneuvering operations, and for the course "TMR4243-Marine Control Systems II".

Figure 2.1.2: CSEI.

CS Arctic Drillship In 2015 Jon Bjørnø created the C/S Arctic Drillship (CSAD) Bjørnø (2016). CSAD is a 1:90 scale model of the Arctic drillship created by Inocean for Statoil. It was developed to perform tests on thruster-assisted position mooring systems. The dimensions are displayed in table 2.1.2.

Table 2.1.2: Dimensions CSAD. Courtesy: Bjørnø (2016).

Description	**Data**
Length	2.578$[m]$
Breadth	0.440$[m]$
Depth (moulded)	0.211$[m]$
Draft (design)	0.133$[m]$

The model is equipped with six azimuth thrusters and a rotatable circular turret. This turret enables it to connect four mooring lines and a riser. The model is primarily used for station-keeping purposes, as its size makes it impractical for maneuvering operations in the MC-Lab.

Figure 2.1.3: CSAD.

C/S Saucer In 2015 the C/S Saucer (CSS) was created by Tor Kvestad Idland (Idland (2015)). The CSS's hull is spherical, providing an unorthodox testing platform and rapid response in surge and sway. CSS is fully actuated with three azimuth thrusters. Furthermore, the lid works as a carrying platform where different sensors can be equipped.

Figure 2.1.4: CSS.

2.2 Cyber-physical testing

Cyber-physical testing provides the ability to develop and test integrated systems. One way of performing cyber-physical testing is through hardware-in-the-loop (HIL) testing. Johansen et al. (2007) defines HIL testing as verifying the required functions of a hardware/software system or component by interfacing it with a HIL simulator and executing

the functions for the integrated system. The ability for HIL testing is important for the MC-Lab, as lab time is limited. It provides a way to develop an test before arriving in the lab. HIL testing also provides ways to test software or control algorithms in ways that would be impractical or infeasible with the real model in the lab. Therefore, a good HIL simulator is important. In Bjørnø (2015), they describe the MC-Lab as having a HIL-Lab. Today all of the models have HIL-simulator models.

2.3 Robot Operating System

The system's architecture is based on ROS. ROS is an open-source framework with tools and libraries that help developers and researchers build and reuse code to create robotic applications. This includes drivers, advanced algorithms, and developing applications to simplify the process of making a robotic system. In addition, it has a global community of engineers, computer scientists, and hobbyists who make robotics accessible and available for everyone.

ROS is a middleware. It handles services such as hardware abstraction, package management, and message-passing between processes. Furthermore, it provides a publish-subscribe messaging infrastructure supporting construction of distributed computing system. It also provides supporting functionality to build and maintain the application.

ROS-based processes are represented as nodes in graph architecture. The nodes connect through topics, which they use to interact through publish-subscribe messages-passing. The nodes communicate through service calls and parameter services connected to the ROS master. The ROS master is a parameter server that keeps track of all active nodes and the topic to which each node is connected. It establishes a peer-to-peer communication network between the nodes. Figure 2.3.1 illustrates a system containing two nodes communicating through a topic. In this case, NODE1 publishes messages to the TOPIC, and NODE2 subscribes to the same topic. The publish-subscribe communication follows the structure of event-driven programming, where the subscribing node will act when a message is published on the topic.

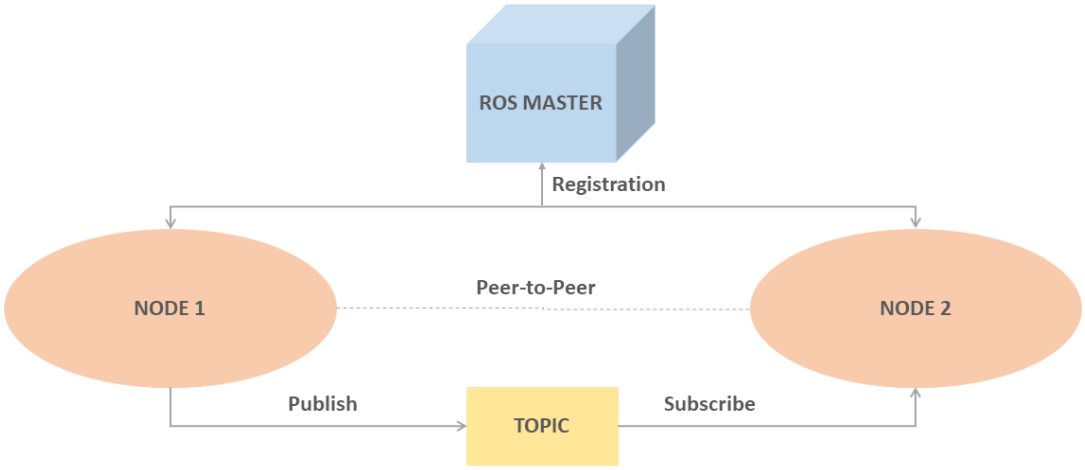


Figure 2.3.1: ROS architecture concept.

2.4 Pulse width modulation

Pulse width modulation(PWM) is a way to digitally create an analog signal. PWM is often used to control motors, servos, and LEDs. A PWM signal is created using a

continuous signal that is set to either on or off.

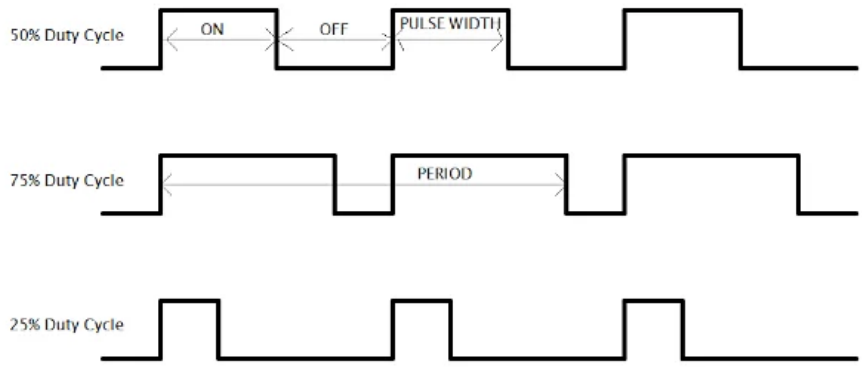


Figure 2.4.1: PWM signal courtesy: EEPOWER (2023)

The amount the signal is set to high over the period is called the Duty cycle from EEPOWER (2023).

$$DutyCycle = \frac{T_{on}}{T_{of} + T_{on}} \tag{2.1}$$

This is represented as a percentage, so at 50% duty cycle, a motor would spin at 50% capacity, and at 100% duty cycle, a 100% capacity.

2.5 Kinematics

This section shortly presents the kinematic notation and reference frames used in the MC-lab and this report. The two main reference frames used in the MC-lab are

- **Basin frame:** Normally, for ships, an earth-fixed reference frame is used. When tests are performed at the MC-lab the basin-fixed reference frame is used instead. It has a coordinate system as illustrated in figure 2.1.1

- **Body frame:** For the body frame, the coordinate system is fixed at the center of the vessel. With x direction positive in the forward direction, y direction positive in the starboard direction, and z positive in the downward direction.

For the models, the 6 degrees of motion(6DOF) in the body-fixed frame is as illustrated in figure 2.5.1.

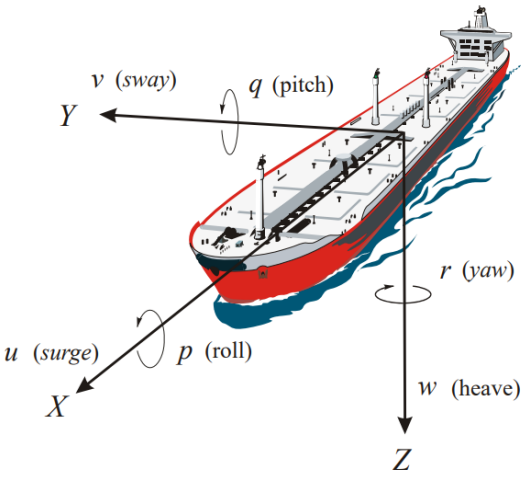


Figure 2.5.1: 6DOF courtesy: Fossen (2011)

Following SNAME-notation for 6DOF of a vessel. The notation is as follows

	Forces and moments	position and Euler angles	linear and angular velocities
Surge motion	X	x	u
Sway motion	Y	y	v
Heave motion	Z	z	w
Roll motion	K	ϕ	p
Pitch motion	M	θ	q
Yaw motion	N	ψ	r

Table 2.5.1: SNAME-notation for 6DOF courtesy Fossen (2011)

Often only 3 degrees of motion(3DOF) is used, meaning only surge, sway, and yaw motions. To translate the body-fixed frame motion to the basin-fixed frame, we need the principal rotation described in Fossen (2011).

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s & c\phi \end{bmatrix}, \quad \mathbf{R}_{y,\theta} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & s0 & c\theta \end{bmatrix}, \quad \mathbf{R}_{z,\psi} = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

CHAPTER

THREE

REQUIREMENT SPECIFICATION

3.1 Requirements Specifications

This section is a reformulation of Elvenes et al. (2023) and outlines the intended use and requirement specification for the new vessel.

3.1.1 Intended use

C/S Jonny is intended to be used for research and testing at the MC-Lab at the Department of Marine Technology at NTNU. The MC-Lab fleet needs a smaller, more maneuverable ASV that can test higher-speed maneuvering-based guidance and control algorithms. The purpose of CSJ is to satisfy this need. CSJ will mainly operate in calm water, i.e., no waves or currents.

3.1.2 Main requirements

CSJ should use the Jonny Harbor Tugboat hull manufactured by Aeronaut (Aeronaut (2022)). It is on a scale of 1:32. Tabel 3.1.1 illustrates the specifications of the Jonny Harbor Tugboat hull.

Table 3.1.1: Hull specifics of CSJ.

Description	Data
Length overall (LOA)	0.99 [m]
Breadth	0.308 [m]
Draft	0.110 [m]

Furthermore, CSJ should be fitted with two azimuth thrusters astern and a tunnel thruster in the bow. After the construction, CSJ should have the following functional requirements:

- Inertial measurement unit to measure angular rates and acceleration.
- Wireless communication and control.
- IR spheres for Qualisys motion capture system measurements.
- Joystick control with two modes:
    1. Body-fixed motion control
    2. Basin-fixed motion control
- Modular components with plug-and-play capabilities
- Watertight deck

3.1.3 Equipment and instruments

Equipment and instruments that CSJ needs to perform the functions listed in table 3.1.2 are listed below. The specific items are described in detail in the appendix.

- Two azimuth thrusters and a bow thruster.
- Motor and Electronic Speed Controller (ESC) for each thruster.
- Servo motor for angle control of azimuth thrusters.
- LiPo batteries as a power source.
- Inertial measurement units (IMU).
- Dualshock 4 controller for joystick-control.
- Raspberry Pi as an embedded computer.
- Control box.
- Watertight deck

CHAPTER

FOUR

DEVELOPMENT OF C/S JONNY

This section outlines the development process of C/S Jonny. The development was performed over two semesters, and this section contains some reformulations and figures from Elvenes et al. (2023) as well as further developments. This is done to provide a complete and cohesive outline of the development process.

4.1 Parts

The parts needed for the model were chosen with the help of Robert Oppland. A desire for standardization in the MC-lab led to many of the components being chosen based on CSAD. The entire part list with specifications is in the appendix.

4.2 Assembly of bowthruster

After the parts were selected, the bow thruster was installed. Firstly by installing the thruster and tunnel, Then installing the motor. This section is a reformulation from Elvenes et al. (2023)

4.2.1 Installation of the thruster

The first step in installing the bow thruster was to mark the hull for drilling. , The instruction manual and guide markings on the hull from Aeronaut (2022) were used to ensure proper placement. Once the holes were drilled, a mounting rig was created to ensure proper alignment. The mounting rig was created by modeling and 3D printing mounting plugs for the ends of the tunnels. The plugs were then threaded over two long threaded rods, with a nut on either side. The rods were then attached to a long bar that could be placed over the model. The setup can be seen in figure 4.2.1. The mounting bracket enabled us to ensure that the thruster was leveled around the model's z, x, and y-axis while also providing a way to hold the thruster in place while casting. When the thruster was correctly positioned, it was then cast in place using two-part epoxy. Then when the epoxy had cured, the excess tubing was cut off and sanded flush with the hull

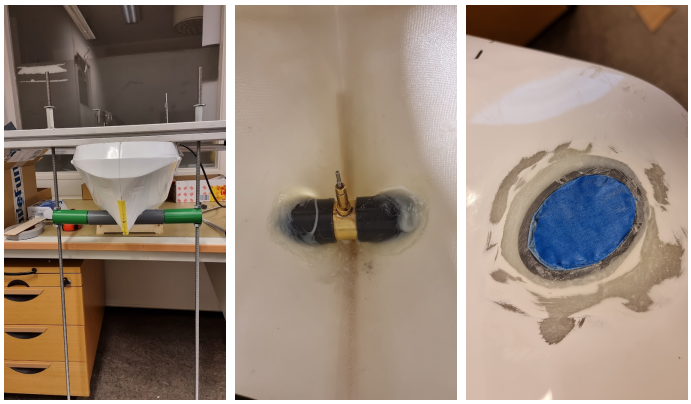


Figure 4.2.1: Installation process for bow thruster

4.2.2 Installation of motor

The chosen bow thruster had no mounting for a motor, so one had to be created. The mount needed to hold the motor securely and align the two driveshafts. To achieve this, the bow thruster and motor were modeled using the 3D modeling software FreeCad. After an iterative design process around these models, a two-part design was created, as seen in 4.3.3. Using four bolts and nuts, the mounting bracket is designed to clamp around the bow thruster and the driveshaft casing. The clamping around the driveshaft casing ensures the alignment of the two driveshafts. Space for the driveshaft coupling between the two driveshafts is also provided. After the model was done, it was 3D printed using 100% infill with a PLA+ filament for extra durability. The final installation can be seen in figure 4.3.3

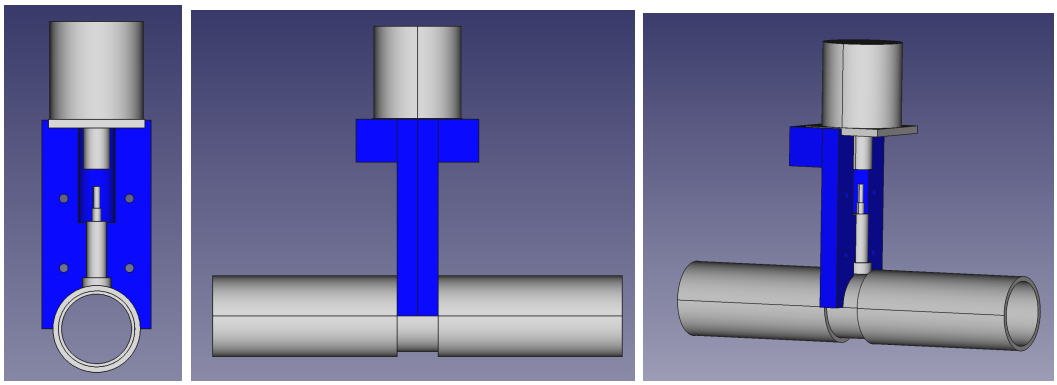


Figure 4.2.2: Mounting bracket for bow thruster

4.3 Assembly of azimuth thrusters

The assembly of the rear azimuth thruster was a multi-step process. Firstly a mounting plate for level installation was needed. Secondly, the provided thrusters only had 180° turning rate and were rebuilt to be able to provide 360° rotation.

4.3.1 Creation of mounting plate

The Jonny tug boat model Aeronaut (2022) is intended to be used with driveshaft propellers and rudders. Though it also comes with a conversion kit for azimuth thruster, this kit is not compatible with our selection of thrusters. Even though we could not use the included conversion kit, it provided an excellent reference to build our mounting

plate. A lower flat surface could be made using the same width, length, and angles of the included kit. The part was first modeled in Sketchup and then produced in steel by Sintef. This mounting plate provides both a flat level surface and the position for the azimuths. The model and finished mounting plate can be seen in figure 4.3.1

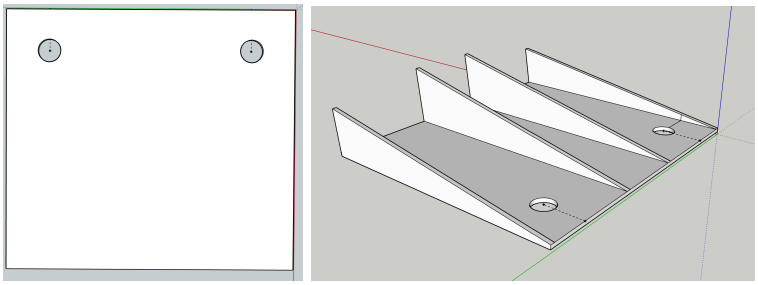


Figure 4.3.1: Mounting plate, top view, and underside

4.3.2 Installation of mounting plate

After the mounting plate was manufactured, the plate could be installed. Firstly the correct position was marked, ensuring that the thrusters would be placed symmetrically on the hull. Then the hull fittings that came with the thrusters were mounted to the plate using screws and epoxy. Then the holes for the hull fittings were drilled in the marked positions. Lastly, the plate and hull fittings are installed using epoxy by applying epoxy along the marked lines and then clamping the plate to the hull. Custom caps were modeled, printed, and used during the installation to prevent the epoxy from clogging up the hull fittings.

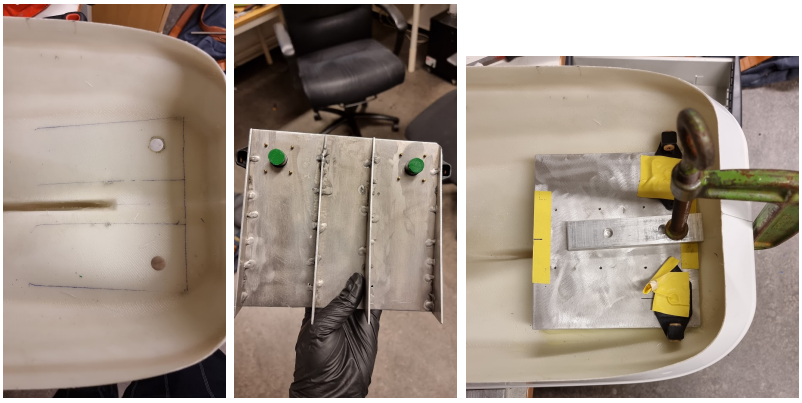


Figure 4.3.2: Installation of mounting plate

4.3.3 Converting thruster to 360° rotation

The thrusters provided only had 180° of rotation; to achieve full 360° capabilities, the thrusters were modified. The modification started by modeling all the necessary parts, such as the motor, ESC, Servo, driveshaft coupling, and the driveshaft of the thruster. We wanted a single unit to house all the necessary components of the thruster. To decrease the number of loose components in the final model and for better protection of the components. It was also a desire to have as direct drive as possible, meaning the motor should be directly connected to the driveshaft.

Designing the conversion was an iterative process, and many prototypes were made. To model the part FreCad was used. The process started with creating two sets of gears in a gear ratio of 1:1. The 1:1 was chosen to simplify the operation of the model by

removing the need for conversion of the angles between the thruster and servo. The Mx-28 servo has a torque capacity of 3.1[Nm]Robotis (2022a), which should be plenty to handle the 1:1 ratio. The gear's height was modeled after the included thruster gear. Since the gears will be 3D printed, a gear module of 2 was selected with 20 teeth, which means relatively large teeth.

$$module = \frac{diameter}{NO.teeth} \tag{4.1}$$

from stock gears (2023). This leads to a gear with a diameter of 4[cm]. The gears were also created as double helix gears with an angle of 20°. The double helix gear removes a lot of the axial forces along the drive shaft that a regular gear usually creates. After this, the holes for mounting to the thruster and for attaching to the servo horn were added.

Once the gears were modeled, the positioning of the servo relative to the driveshaft could be determined. The motor's position was made to be directly above the driveshaft, providing the possibility of connecting the two driveshafts directly with a driveshaft coupling. The height was determined using the height of the gear and the length of the driveshaft coupling. Lastly, the placement of the ESC was made to be above the servo and behind the motor.

With the placement of each component, the mount could be created. It is created as two parts that are mirrored around the center. It is joined together by four bolts going horizontally and screws attaching it to the mounting plate. An open back behind the motor and an opening below provide airflow for cooling the motor. Lastly, channels were created to thread the wires through. The mount and gears were 3D printed using 100% infill and PLA+ filament.

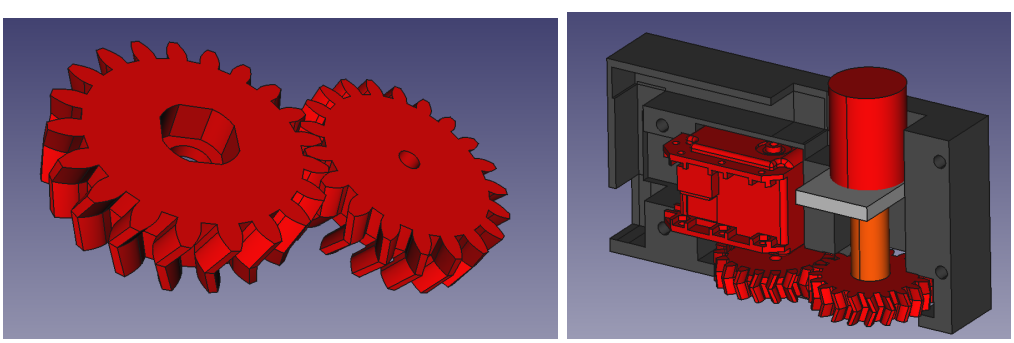


Figure 4.3.3: Conversion to 360° rotation

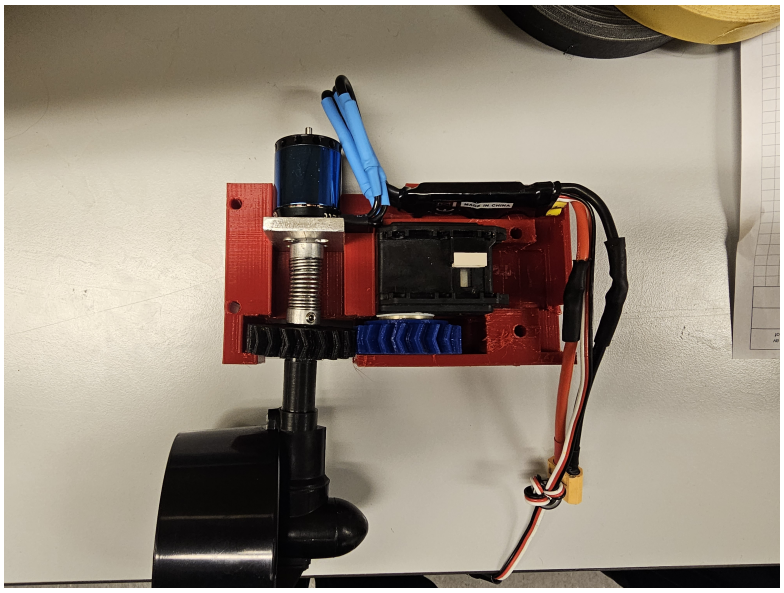


Figure 4.3.4: Printed azimuth mount with all components

4.4 IMU

The IMU arrives as a stand-alone chip. To mount and protect the sensor, an IMU box was created. The objective of the box is to provide protection, mounting, and the ability to daisy chain the sensors together with plug-and-play. Since the IMU uses the I2C protocol for communication, we can, through the use of an I2C breakout board, daisy chain the sensors. This means that for each IMU, there is a breakout board. For this purpose, a 3-piece box was modeled as seen in figure 4.4.1. The IMU sensor is mounted in the center at the bottom, with the I2C breakout board over it. The holes are for LEMO connections, which provide the IMU with plug-and-play capabilities.

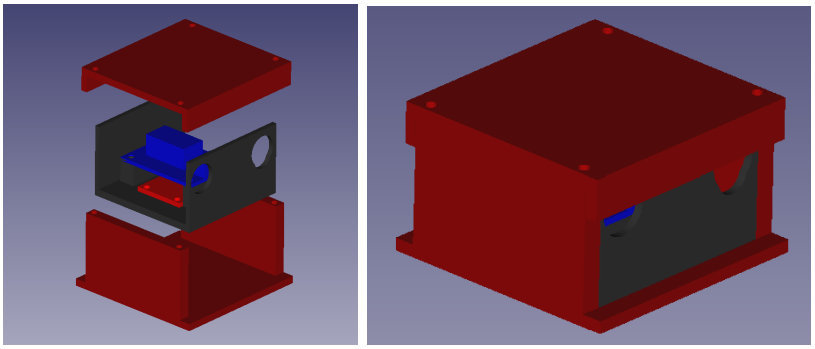


Figure 4.4.1: IMU box

4.5 Electronic box

The purpose of the electronic box is to house most of the electronic components, such as the PWM-Breakout board, DC-DC converter, U2D2 module, and the Raspberry PI. While holding all the components, the box should provide protection, plug-and-play capabilities, and cable management. To create the box, a relatively small box was selected. An inlay was created to fit all the components and securely attach them. The inlay provides mounting points for all components, as shown in red in figure 4.5.1. The Raspberry Pi can easily be removed by simply lifting it out of the box.

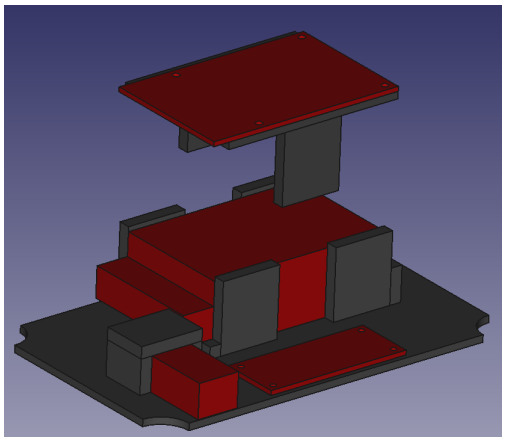


Figure 4.5.1: Electronic box inlay

After the inlay was printed and all components were attached. Holes for the lemo connections of the three motors, servo, and IMU, were drilled, and also for the XT-60 power plug. These connections provided the ability to easily disconnect the different units without opening the control box. After drilling holes, the electronic box was soldered, and cable managed. The final box can be seen in figure 4.5.2.

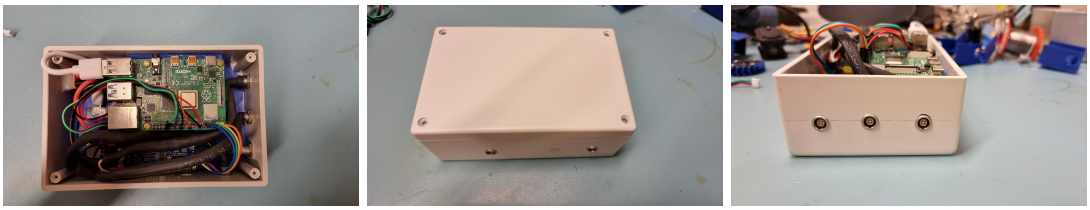


Figure 4.5.2: Electronic box

A holder for the electronic box was also created using 3D modeling and 3D printing. It was attached by screwing it into the frame of the model.

4.6 Battery holder

For securing the batteries, a battery holder was created. The objective of this holder is to keep the batteries from moving while also providing ease of access and connection. To achieve this, a plate was modeled by modeling the batteries and connection plug, as seen in figure 4.6.1. The two-part at the front are for the connection points of the battery; here, the plugs can be inserted and fastened. The batteries are easy to insert and remove while also staying secure during operations.

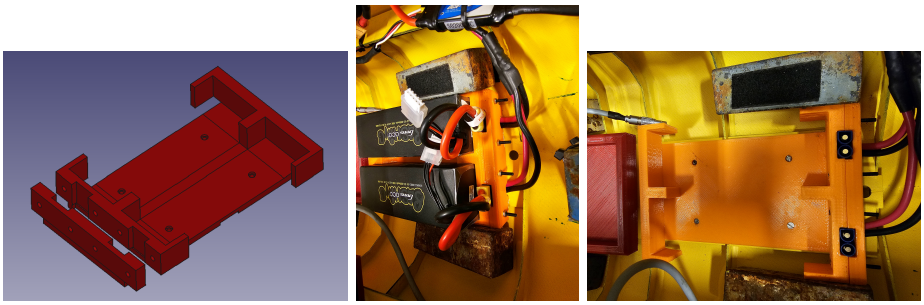


Figure 4.6.1: Battery holder

4.7 Painting

The paint provides some resistance to wear but, more critically, better tracking capabilities with the camera. This is why yellow is selected to create an excellent contrast to the water, making it easier to track with a camera. Before painting, the entire outside was sanded to increase the adhesion of the paint. Then the bow thruster and holes for the azimuth thrusters were masked with masking tape. Finally, the model was painted yellow using spray paint as illustrated in figure 4.7.1.

Figure 4.7.1: Painting.

4.8 Deck

A deck is needed to prevent water from entering the hull during operation and provide a mounting surface for the IR spheres. The deck should be reasonably watertight to prevent short-circuiting of the components inside while providing easy access when needed. First, a stencil was made using a piece of cardboard to create the deck. When the cardboard had been cut to fit over the hull, the next step was to decide where the access hatches should be. Two access hatches were chosen, one for the rear providing access to the electronic box and azimuth thrusters and one for the bow for access to the bow thruster and batteries. When the hatches were placed, the deck's outline was drawn on a honeycomb panel. A raised edge of about 2[cm] was planned around the hatches to provide waterproofing. When the dimensions were selected, the honeycomb panel was given to Sintef, which made the deck.

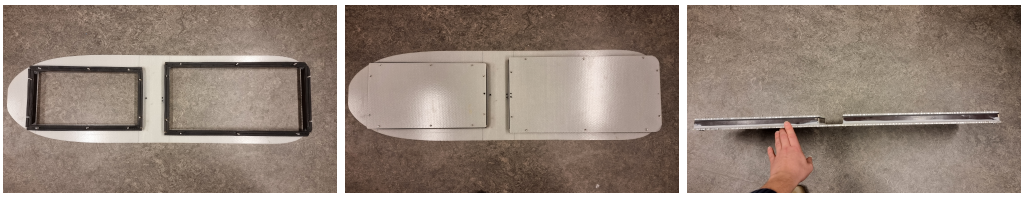


Figure 4.8.1: Deck

The next step was then to attach the deck to the hull. Firstly the deck was clamped in place. Then the masking tape was used along the edges of the hull to hold the epoxy. The epoxy was then applied and left to cure before removing the clamps and tape.

Figure 4.8.2: Deck installation

4.9 IR spheres

The Qualisys motion capture system uses infrared to track the motion of the vessel. Therefore, IR reflective spheres are needed. For the best tracking capabilities, there should be different spheres, and they should not be on the same plane. For C/S Jonny, 4 IR spheres arranged in an L shape with different heights were chosen. To achieve this, a mounting bracket was first modeled and 3d printed, as seen in the figure. After printing, the brackets had two nuts inserted by melting them into the plastic. Then threaded rods were cut to different lengths, and the IR spheres were threaded on the top. Lastly, the mounting brackets were epoxyed to the deck in an L shape.

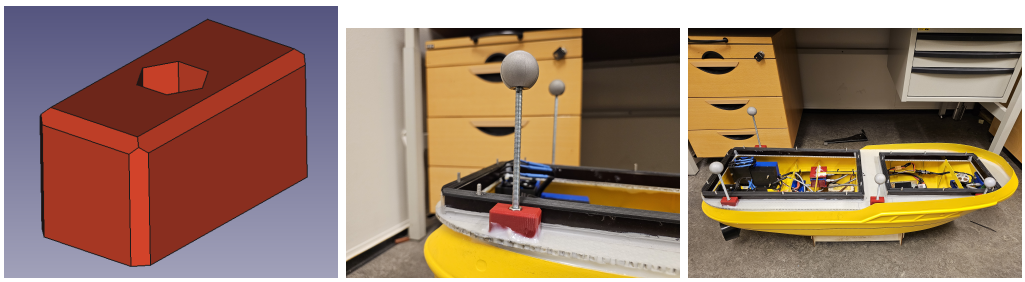


Figure 4.9.1: IR installation

4.10 Ballasting

After all components were assembled, the final step was to ballast the model. To perform this, the guide from the Jonny tugboat model was used Aeronaut (2022). This provided instruction on where the waterline should be drawn, as seen in figure 4.10.1

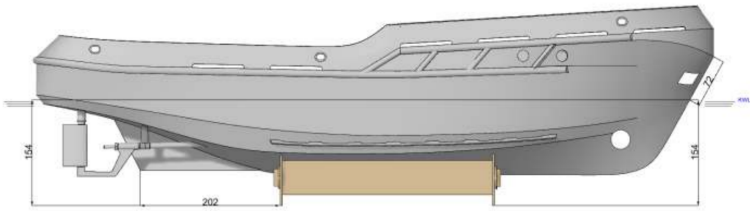


Figure 4.10.1: Water line guide courtesy Aeronaut (2022)

The model was set up as instructed on a flatbed to mark the waterline. Then using a pen holder that holds the pen at a fixed height, the waterline was drawn by dragging the penholder across the surface. The waterline can be seen in figure 4.10.2

Figure 4.10.2: Waterline

When the waterline had been marked, it could be ballasted. By adding weights until the model floated at the waterline as seen in 4.10.3

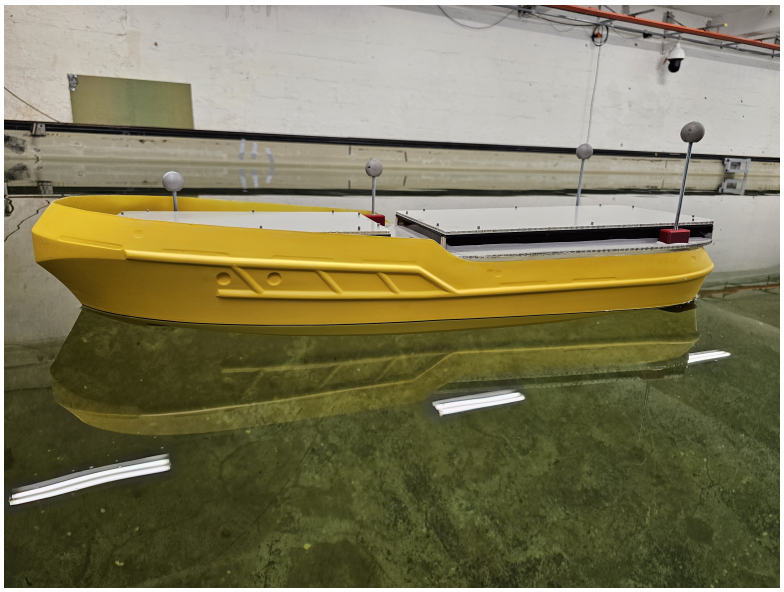


Figure 4.10.3: Ballasting

CHAPTER

FIVE

SYSTEM ARCHITECTURE

5.1 Vessel systems

This section presents the physical placement of all relevant components, such as thrusters, IMUs, and IR spheres, and is a reformulation from Elvenes et al. (2023).

5.1.1 thruster configuration

The final dimensions and position of the thrusters are shown in table 5.1.1, with the position indicated as in figure 5.1.1.

Table 5.1.1: Thruster configuration.

Parameter	Value[cm]
l_{x1}	41.5
l_{x2}	41.5
l_{x3}	37.0
l_{y1}	7.0
l_{y2}	7.0

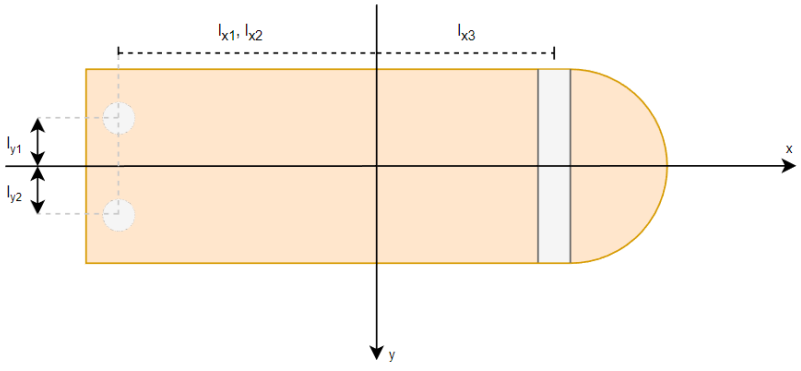


Figure 5.1.1: Thruster configuration of CSJ.

5.1.2 IMU

the position of the IMUs is presented in table 5.1.2

IMU no.	x[cm]	y[cm]	z[cm]
1	-24	-5	3
2	-16	-8	6.4
3	7.5	11.5	9.9
4	43	0	11.5

Table 5.1.2: Position of IMU

5.1.3 IR spheres

The position of the IR spheres is presented in table 5.1.3

IR sphere no.	x[cm]	y[cm]	z[cm]
1	-37.5	13	33.2
2	-38.2	12.5	30.2
3	2.9	12.5	26.5
4	34.2	11.5	23.6

Table 5.1.3: Position of IR spheres

5.2 Power system single-line diagram

A representation of the power system is illustrated in figure 5.2.1. The batteries have a 14.8V voltage which is connected directly to the motors. There is a potential transformer to adjust the voltage to 5V for the Raspberry Pi. The port azimuth thruster motor is denoted M_{AP} and its corresponding servo S_{AP}. The same goes for starboard azimuth with the subscript AS. The bow thruster motor is denoted as M_B.

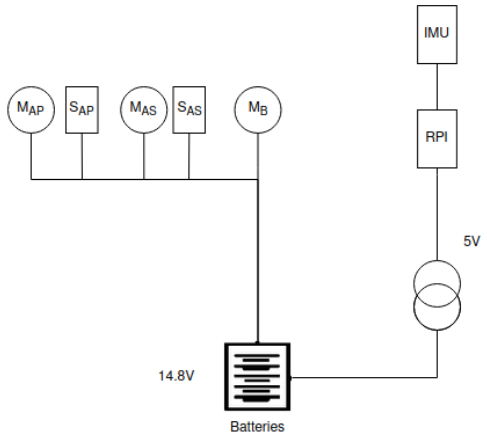


Figure 5.2.1: Single line diagram of CSJ's power system.

5.3 Communication signal

figure 5.3.1 shows the different communication protocols used. I2C is used for the IMUs and to connect the PWM breakout board. The breakout board provides the ESC with the PWM signal. The ESC then amplifies this signal with the 14.8V power connection to drive the motor. The servos are driven using the UART Rx protocol provided by the U2D2. Which, in return, is connected to the Raspberry Pi through USB and powered by the 14.8v connection.

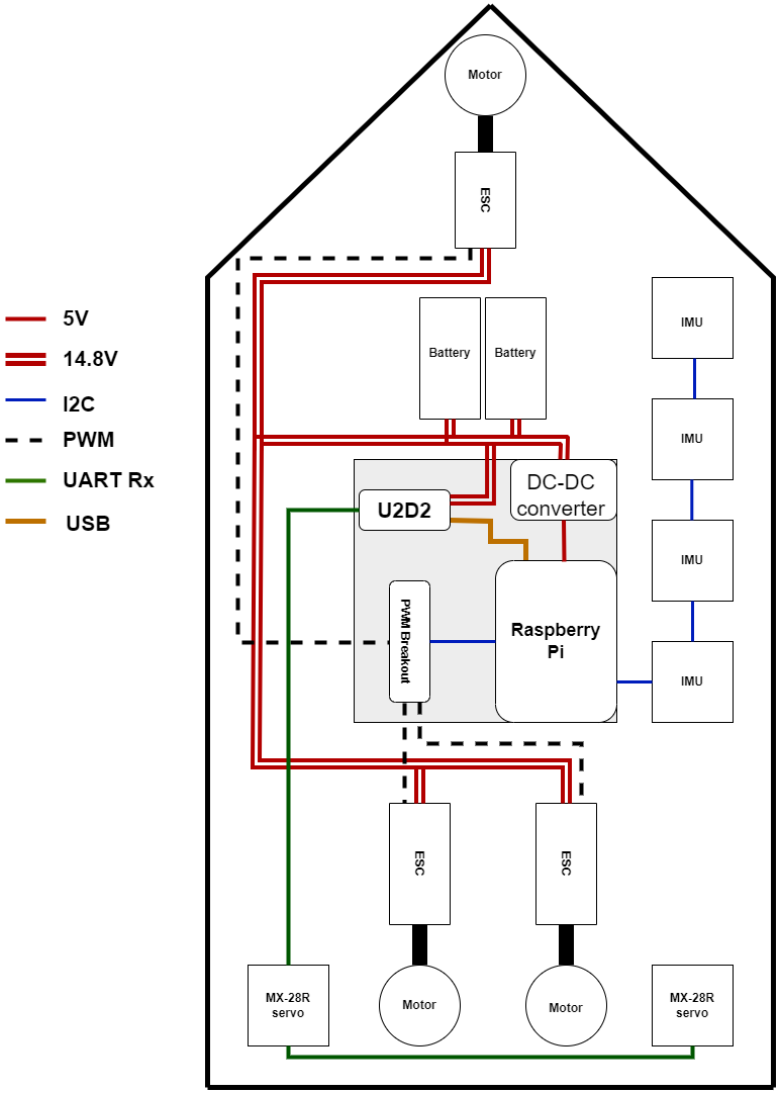


Figure 5.3.1: Communication signals of the CSJ.

5.4 Software topology

This section is about the implemented software topology and ROS nodes. In an attempt for standardization in the MC-lab, Roger Skjetne proposed a software topology for all the models. The proposed solution is illustrated in 5.4.1.

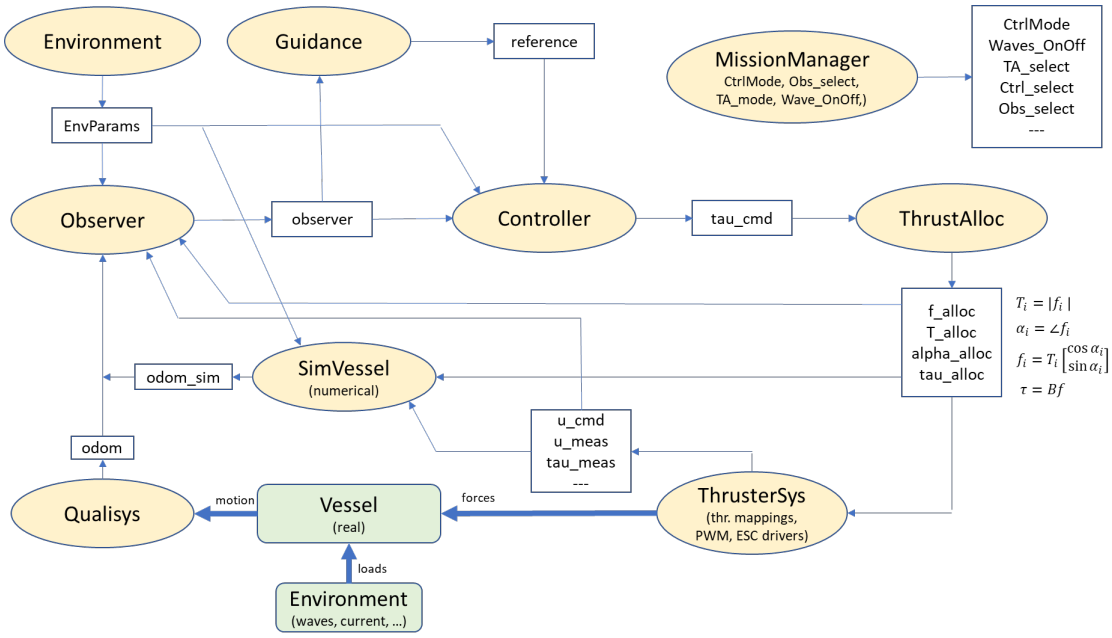


Figure 5.4.1: Software topology models MC-lab courtesy: Roger Skjetne

Since a lot of these nodes are out of the scope of this project, only some have been implemented, and the rest is further work. The implemented nodes are illustrated in figure 5.4.2 and provides the basic functionality for the model.

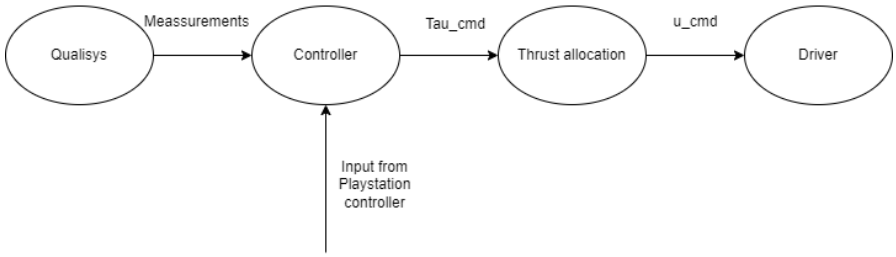


Figure 5.4.2: The implemented software topology

Qualisys

The Qualisys system uses physical cameras placed at the MC-Lab and uses triangulation to measure the position of the vessel. The cameras measure the pose and attitude of the vessel by tracking markers attached to it. It outputs the position of the vessel. The Qualisys node is run on the computer on land and not on the vessel

Controller

The controller node is currently only implemented as a joystick controller. Here the input signals from the Playstation controller are mapped to a tau command. Two types of joystick control are implemented one in the body frame and one in the basin frame. To translate the forces from the basin frame to the body frame, the heading of the ship is needed from the Qualisys node.

Thrust allocation

The thrust allocation node takes the commanded forces and uses one of the algorithms described in section 10. Then the calculated forces are mapped to PWM using the algorithm outlined in 9. This provides the angles and PWM signal needed by the driver.

Driver

The driver node is adapted from CSAD Bjørnø (2015) to work with three thrusters and two servos. It takes the signal u_cmd and sends it to the physical components.

CHAPTER
SIX

BASIN TRIAL

When the development of C/S Jonny was completed, a basin trial was performed to validate and check the model was working correctly.

6.1 Expermiental setup

The basin trial will verify the building process and show that the components work correctly. A basin trial was created with inspiration from the four-corner test. The test is shown in figure 6.1.1. It consists of motion in all 3DOF, meaning surge, sway, and yaw. The thrust mapping algorithm from section 9 and a simple pseudo inverse thrust allocation were implemented to perform the test. Then the test was performed with a joystick controller, using a body-fixed frame and then a basin-fixed frame.

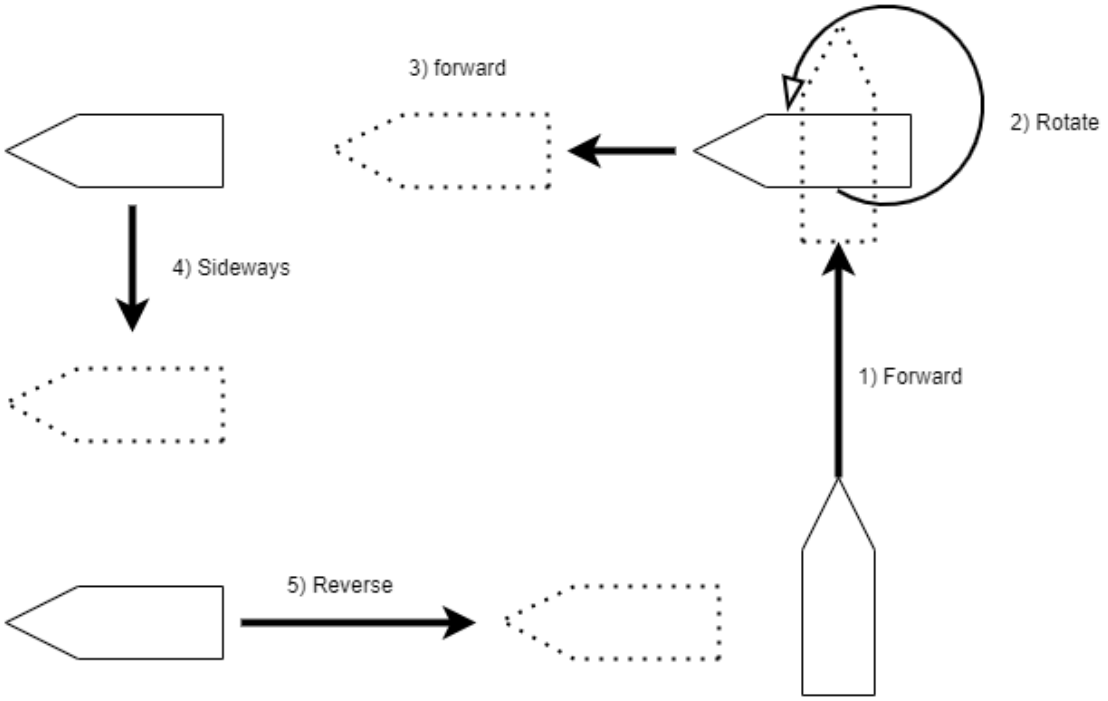


Figure 6.1.1: Basin trial

6.2 Results

In this section, the results from the basin trial are presented.

6.2.1 Body-fixed frame

In figure 6.2.1, we can see the position of CSJ during the test using the body-fixed controller. The heading and commanded tau can be seen in 6.2.2. A video of the test is found in the appendix.

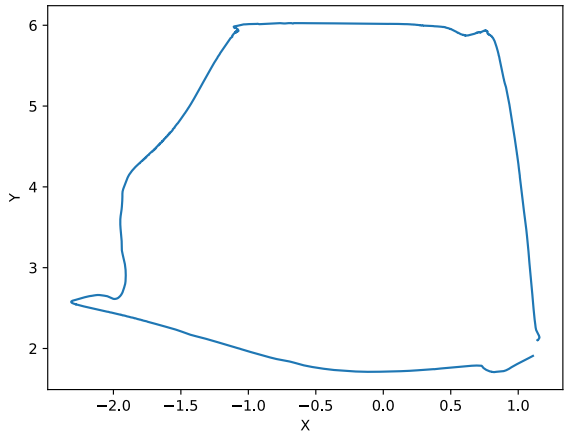


Figure 6.2.1: x and y position for basin trial with body-fixed joystick controller

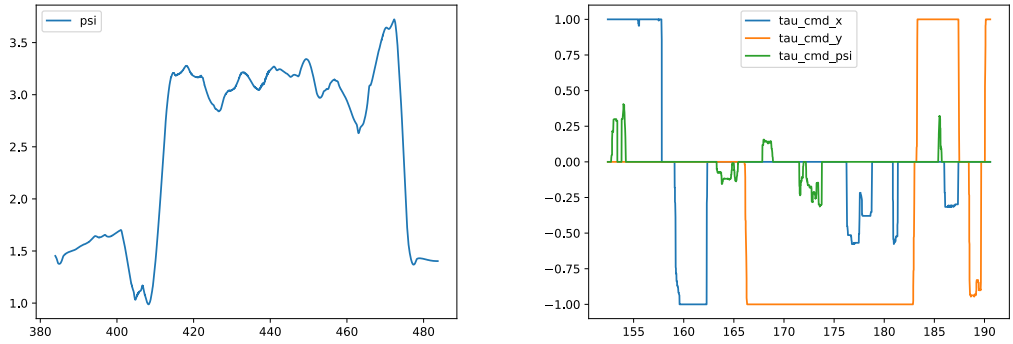


Figure 6.2.2: Heading and Comanded tau for basin trial with body-fixed joystick controller

6.2.2 Basin-fixed frame

In figure 6.2.3, we can see the position of CSJ during the test using a basin-fixed controller. The heading and commanded tau can be seen in 6.2.4. A video of the test is found in the appendix.

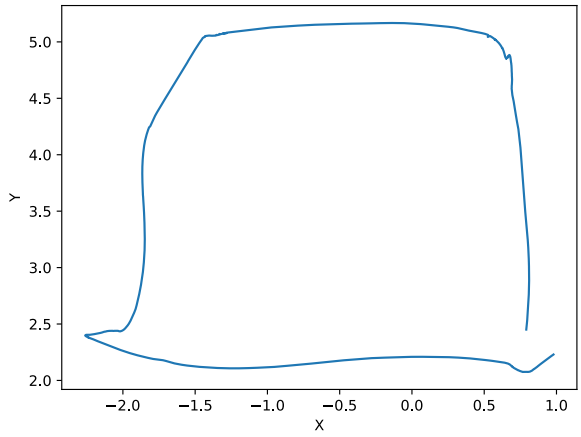


Figure 6.2.3: x and y position for basin trial with basin-fixed joystick controller

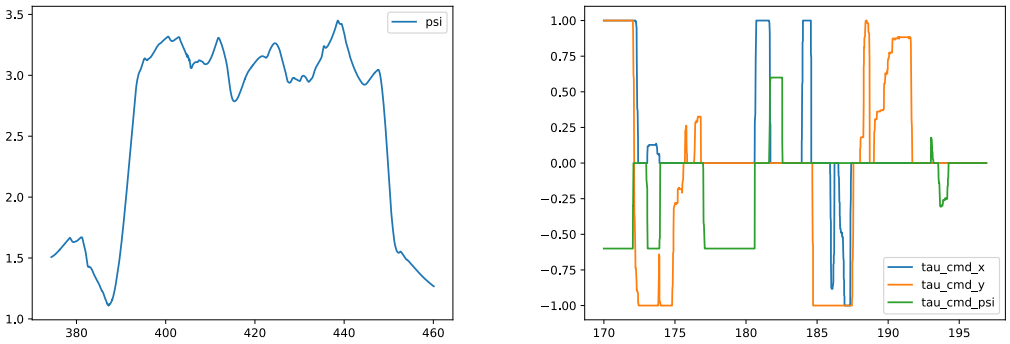


Figure 6.2.4: Heading and Comanded tau for basin trial with basin-fixed joystick controller

CHAPTER
SEVEN

DISCUSSION & FURTHER WORK

7.1 Discussion

After CSJ was completely built, thrust mapping was planned. When the thrust mapping commenced, a bug in the driver code was discovered. This bug would sometimes set one or more PWM signals to the max at the closing of either the driver node or thrust allocation node. This also occurs on CSAD, but CSAD has much smaller motors and thrusters, so the problem is manageable. However, when this happened on CSJ, the peak measured force was around 113[N]—causing the port side thruster to be destroyed.

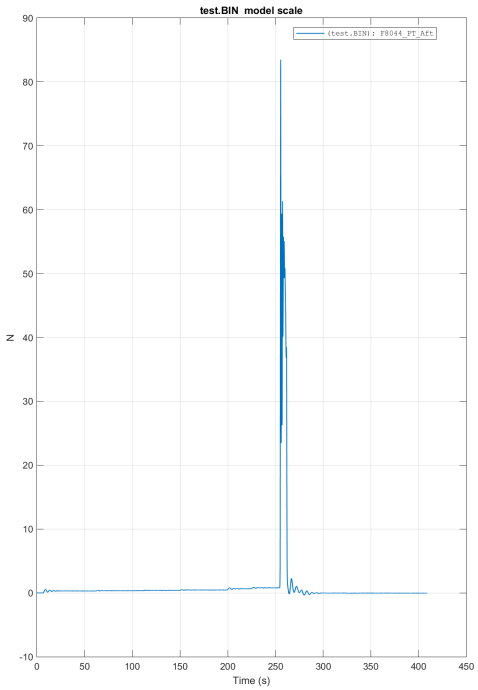


Figure 7.1.1: Measurement from port force sensor

The bug was fixed, and new thrusters were ordered. New motors were selected that were much smaller than the original. The rebuild of the azimuth mount could then commence, including design, printing, and implementation. The same thrusters were ordered as the original due to the mounting holes in the hull being epoxyed in place.

The rebuilt azimuth mounts work well, but the thrusters are over-dimensioned. This causes the maximum PWM signal to be set relatively low. To limit the model to a reasonable speed, the maximum PWM signal is set to 0.22. The motors do not start spinning before a PWM signal of 0.11. As Frederich (2016) noted when implementing thrust allocation algorithms for CSAD, the minimum PWM interval for the ESC is 0.01. This causes CSJ to only have 12 points of regulation for the speed of the motors.

Even though the command signal for the thrusters is relatively coarse, the basin trial went quite well. As seen in figure 6.2.3 and 6.2.1, CSJ drives well in the surge direction, with minimal change in heading. When given a commanded yaw moment, CSJ turns well around its axis without too much movement. For sway movements, some yaw moments arise. This is probably due to the coarse control of the azimuth thrusters and hydrodynamics since the hull is not symmetric in this direction. The rest of the components work pretty well. The bow thruster operates smoothly with minimal vibration. The control box manages the wires and components, so CSJ has fewer loose components and wires than the other models. The Batteries also provide a very long run-time. For the thrust mapping, CSJ was run almost continually for about 4 hours.

Part II

Thrust allocation for C/S jonny

CHAPTER

EIGHT

BACKGROUND

8.1 Thrusters and thruster dynamics

According to Fossen (2011), the most common actuators for marine vessels are:

- **Main propellers :** often mounted with rudders at the aft to produce thrust force in the surge direction in the body-fixed frame of the ship.
- **Tunnel thrusters:** For ships usually fixed in the transverse direction. Installed using a tunnel through the hull.
- **Azimuth thrusters:** Thrusters that can produce forces in the X and Y directions by rotating the thruster around the z-axis. Often the preferred type of thruster for station keeping, as it can produce thrust in several directions.
- **water jets:** often used for high-speed vessels as an alternative to main propellers.

In this project, the relevant thrusters are the azimuth and tunnel thrusters since CSJ is equipped with two azimuth thrusters and one bow thruster. To relate the rotational speed of the propeller to produced thrust(T_a) or torque(Q_a) Sørensen (2012) defines the relation as.

$$T_a = sign(n)K_T\rho D^4 n^2 \tag{8.1}$$

$$Q_a = sign(n)K_Q\rho D^5 n^2 \tag{8.2}$$

Where n is the rotational speed of the propeller in $[Revelutions/s]$, D is the diameter of the propeller, ρ the density of water, and K_T and K_Q is thrust loss coefficients. According to Sørensen (2012), the most common types of thrust loss come from:

- **Inline velocity fluctuation:** The produced thrust is correlated to the water inflow over the propeller
- **Cross-coupling drag:** arises when perpendicular inflow to the thruster direction occurs. This can occur by the ship moving through the water, currents, or jets from other propellers.
- **Ventilation:** When the propeller creates a low pressure that sucks in air. It often arises when the submergence of the propeller is slight, often due to wave motions.
- **In and out of water effects:** For significant motion of the vessel, the thrusters can be lifted out of the water.

- **Coanda effect:** is losses due to the interactions between the thrusters and the hull.
- **Thruster-thruster interactions:** occurs when the outflow of one thruster interferes with neighboring thrusters' ability to produce thrust.

and lastly, he defines the relationship between power consumption and torque as

$$P_a = 2\pi n Q_a \tag{8.3}$$

8.2 Mathematical notation and thruster configuration matrix

Using the mathematical notation presented in Skjetne (2023). All equations in this section are from Skjetne (2023). Starting with the thrust load vector τ for 3DOF.

$$\tau = \begin{bmatrix} f \\ m \end{bmatrix} \qquad f = \begin{bmatrix} X \\ Y \end{bmatrix} [N] \qquad m = \begin{bmatrix} n \end{bmatrix} [Nm] \tag{8.4}$$

The forces and moments are the combined result in the body frame of the m individual thrusters. The forces in the body frame for each thruster is given as follows:

$$f_i = \begin{bmatrix} X_i \\ Y_i \end{bmatrix} [N], \qquad m_i = \begin{bmatrix} N_i \end{bmatrix} [Nm], \qquad l_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} [m] \tag{8.5}$$

The moments m_i is a function of the forces f_i, if $S := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$, then the thrust load vector can be written as

$$\tau_i = \begin{bmatrix} f_i \\ m_i \end{bmatrix} = \begin{bmatrix} X_i \\ Y_i \\ N_i \end{bmatrix} = \begin{bmatrix} f_i \\ l_i^T S^T f_i \end{bmatrix} = \begin{bmatrix} X_i \\ Y_i \\ x_i Y_i - y_i X_i \end{bmatrix} \tag{8.6}$$

$$\tau = \sum_{i=1}^{m} \tau_i = \sum_{i=1}^{m} \begin{bmatrix} f_i \\ l_i^T S^t f_i \end{bmatrix} \tag{8.7}$$

8.2.1 Polar configuration matrix

In polar form, each thruster force is represented by an angle α relative to the body-fixed frame and a force magnitude F. If we define

$$a_i := \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \end{bmatrix} \tag{8.8}$$

then we can write f_i as $f_i = F_i a_i$. Here $a_i := (a_1 ...., a_m)$ and $F := col(F_1, F_2 ..., F_m)$ is the force magnitude. Equation 8.7 can now be re-formulated as

$$\tau = \sum_{i=1}^{m} \tau_i = \sum_{i=1}^{m} \begin{bmatrix} a_i \\ l_i^T S^T a_i \end{bmatrix} F = \begin{bmatrix} a_1 & a_2 & ... & a_m \\ l_1^T S^T a_1 & l_2^T S^T a_2 & ... & l_m^T S^T a_m \end{bmatrix} F =: B(\alpha) F \tag{8.9}$$

where $B(\alpha)$ is the configuration matrix, giving.

$$\tau = B(\alpha) F \tag{8.10}$$

where

$$B(\alpha) = \begin{bmatrix} \cos(\alpha_1) & \cos(\alpha_2) & ... & \cos(\alpha_m) \\ \sin(\alpha_1) & \sin(\alpha_2) & ... & \sin(\alpha_m) \\ x_1 \sin(\alpha_1) - y_1 \cos(\alpha_1) & x_2 \sin(\alpha_2) - y_2 \cos(\alpha_2) & ... & x_m \sin(\alpha_m) - y_m \cos(\alpha_m) \end{bmatrix} \tag{8.11}$$

8.2.2 Rectangular configuration matrix

Another way to represent the configuration matrix is in rectangular form, often called the extended form. Starting with a reformulation of the thrust load vector.

$$\tau_i = \begin{bmatrix} f_i \\ l_i^T S^T f_i \end{bmatrix} = \begin{bmatrix} X_i \\ Y_i \\ x_i Y_i - y_i X_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -y_i & x_i \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \tag{8.12}$$

This gives $F_i := |f_i| = \sqrt{X_i^2 + Y_i^2}$ for the force, and $\alpha_i := \angle f_i = atan2(Y_i, X_i)$ for the angles. If we use m_1 for the number of thrusters with varying angles and m_2 for the number of thrusters with a fixed angle, such as bow thrusters. Then we can define

$$f := \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{m_1} \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ \vdots \\ X_{m_1} \\ Y_{m_1} \end{bmatrix} \in \mathbb{R}^{2m_1} \tag{8.13}$$

for the varying thrusters, and $F := col(F_{m_1+1}, \ldots, F_m)$ for the fixed. The thrust load vector then becomes

$$\tau = \sum_{i=1}^{m_1} \tau_i + \sum_{i=m_1+1}^{m} \tau_i = \sum_{i=1}^{m_1} \begin{bmatrix} X_i \\ Y_i \\ x_i Y_i - y_i X_i \end{bmatrix} + \sum_{i=m_1+1}^{m} \begin{bmatrix} \cos(\alpha_i) \\ \sin(\alpha_i \\ x_i \sin(\alpha_i) - y_i \cos(\alpha_i) \end{bmatrix} F_i = B_1 f + B_2 F \tag{8.14}$$

where $B_1 \in \mathbb{R}^{3x2m_1}$. Here B_1 and B_2.

$$B_1 := \begin{bmatrix} I & I & \ldots & I \\ l_1^T S^T & l_2^T S^T & \ldots & l_m^T S^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & \ldots & 1 & 0 \\ 0 & 1 & 0 & 1 & \ldots & 0 & 1 \\ -y_1 & x_1 & -y_2 & x_2 & \ldots & -y_m & x_m \end{bmatrix} \tag{8.15}$$

$$B_2 := \begin{bmatrix} \cos(\alpha_{m_1+1}) & \ldots & \cos(\alpha_m) \\ \sin(\alpha_{m_1+1}) & \ldots & \sin(\alpha_m) \\ x_{m_1+1} \sin(\alpha_{m_1+1}) - y_{m_1+1} \cos(\alpha_{m_1+1}) & \ldots & x_m \sin(\alpha_m) - y_m \cos(\alpha_m) \end{bmatrix} \tag{8.16}$$

If we now combine the two vectors f and F, and the two configuration matrices B_1 and B_2 as

$$\xi := \begin{bmatrix} f \\ F \end{bmatrix}, \qquad B := \begin{bmatrix} B_1 & B_2 \end{bmatrix} \in \mathbb{R}^{3xp} \tag{8.17}$$

where

$$p = 2m_1 + m_2 \tag{8.18}$$

. Then thrust load vector can be written as

$$\tau = B\xi \tag{8.19}$$

8.3 Thrust allocation problem

The primary goal for the thrust allocation problem is to control the different actuators through some control signal u so they combined to produce a desired thrust load vector τ_{cmd}. from Johansen and Fossen (2013)

$$\tau_{cmd} = h(u, x, t) \tag{8.20}$$

An important aspect of thrust allocation is the actuation of the system. If we want to control a n DOF system, then the actuation of the system is given as Fossen (2011)

- **Underactuated if** $p < n$: the system cannot produce desired forces for all n degrees of freedom.
- **Fullyactuated if** $p = n$: The system can produce reasonable desired forces for all n degrees of freedom.
- **Overactuated if** $p > n$: Capable of producing forces in all n degrees of freedom while also having fault tolerance.

For CSJ, we have two azimuth thrusters and one bow thruster. Using equation 8.18, we get p = 5. Using the 3DOF system, we get that 5>3, meaning we have an overactuated system.

When we have an overactutaed system, many, if not infinite, solutions exist to the thrust allocation problem, and the problem becomes an optimization problem. The secondary goal is to minimize power consumption. A solution to the unconstrained problem in 8.20 is to use the simple pseudoinverse with the configuration matrix.

$$F = B(\alpha)^{\dagger}\tau_{cmd}, \quad \textbf{or} \quad ,\xi = B^{\dagger}\tau_{cmd} \tag{8.21}$$

Here α needs to be fixed angles. This solution, however, does not consider the thrusters' physical constraints. Some of the most common constraints are

- **Saturation**: The thrusters have a minimum and maximum thrust they can produce.
- **Angular constraints**: Azimuth thrusters can have angles they are not allowed to operate in. This can be due to the physical limitations in maximum and minimum angles or forbidden zones. Forbidden zones are often implemented to avoid thruster-thruster interaction or Thruster-sensor interactions.
- **Rate constraints**: The physical system does not work instantaneously and takes time both to turn and to ramp up propellers. This causes angular and force rate constraints. It is also important to limit the wear of the components.

When angular rate constraints are applied, the thrust configuration matrix can become singular, meaning its determinant becomes 0. When this happens, the inverse of the matrix becomes infinitely large. This means that if the system approaches a singular configuration, the thrust force for each thruster grows very large. Steps to limit this growth or avoid singular configurations are often added.

8.4 Maneuvering problem

Skjetne et al. (2011) defines a generic maneuvering problem as follows: If we have a system output $y = h(x)$ with $h : \mathbb{R}^n \to \mathbb{R}^m$, then the desired manifold is all points represented by the set

$$Q := \{x \in \mathbb{R}^n : \exists \theta \in \mathbb{R}^q \quad \text{s.t} \quad h(x) = h_d(\theta)\} \tag{8.22}$$

where $q \leq m$ and the map $\theta \mapsto h_d(\theta)$ is sufficiently smooth. Given the parametrization $h_d(\theta)$ and a dynamics assignment of the manifold, the maneuvering problem is compromised of the two tasks.

1. **Geometric task:** For some absolutely continuous function $\theta(t)$ force the output of y to converge to the manifold $h_d(\theta)$

$$\lim_{t \to \text{inf}} |y(t) - h_d(\theta(t)| = 0 \tag{8.23}$$

   Dynamic task: Force $\dot{\theta}$ to converge to a desired dynamic assignment $f_d(\theta, y, t)$

$$\lim_{t \to \text{inf}} \left| \dot{\theta}(t) - f_d(\theta(t), y(t), t) \right| = 0 \tag{8.24}$$

CHAPTER
NINE

THRUSTMAPPING

This section is about the thrust mapping of C/S Jonny and how to relate force to a PWM signal.

9.1 experimental setup

The motors for the thrusters are controlled through a PWM signal, as described in section 2.4. This signal does not directly correlate to the produced force, so a way to correlate the control signal and forces is needed. To achieve this, a bollard pull test was performed. A bollard pull test is required for ships to be used for towing operations Authority (2023), which is usually performed by fastening the ship using cables that connect the aft to a bollard on land. The test performed on C/S Jonny is a modified version, with more fastening, as seen in figure 9.1.1.

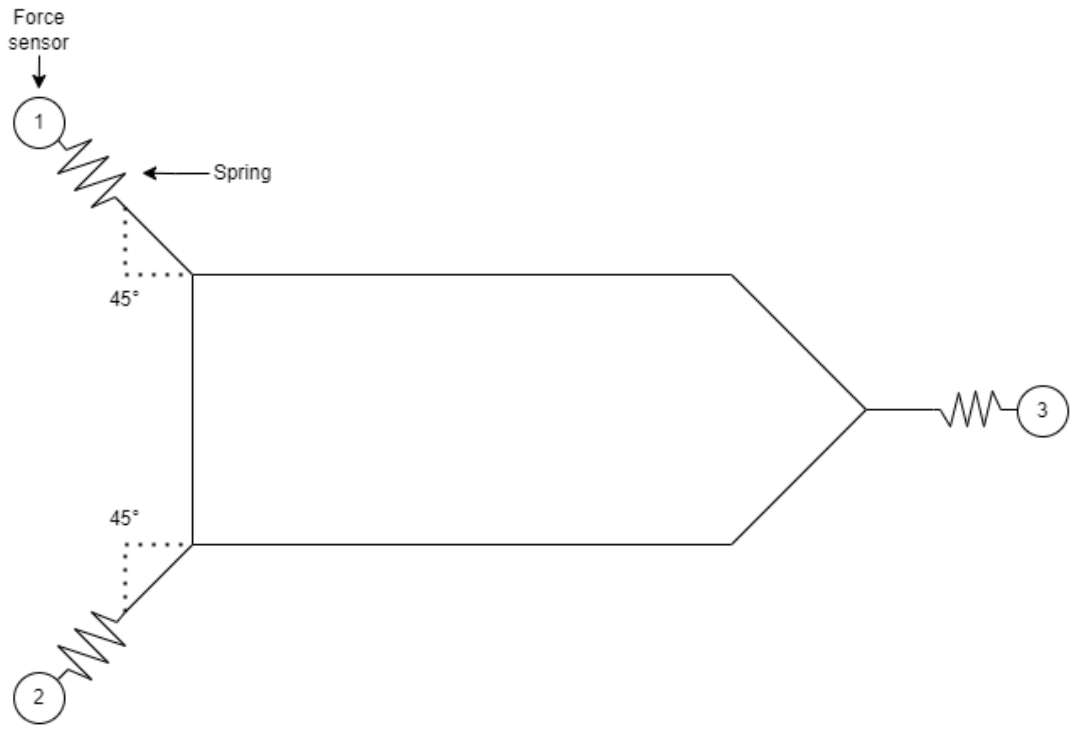


Figure 9.1.1: Bollard pull setup for C/S Jonny

Each cable is connected to a spring and a force sensor that provides the force mea-

surements. The angled fastening at the aft can be decomposed to forces in X and Y directions in the body frame. Since the model will move during the testing, Qualisys was also used to provide the heading of the model.

9.2 mapping

The three measurements were decomposed and translated to the body frame to get the force for the thrusters. Using

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos(\alpha_1 + \psi) & \cos(\alpha_2 - \psi) & \cos(-\psi) \\ \sin(\alpha_i + \psi) & \sin(\alpha_2 - \psi) & \sin(-\psi) \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \tag{9.1}$$

where $\alpha = [-\frac{3\pi}{4}, \frac{3\pi}{4}, 0]$, $F = [F_1, F_2, F_3]$ from the force sensors and ψ is the heading of the ship. Then the force the thruster produced was calculated as

$$F_{thruster} = \sqrt{X^2 + Y^2} \tag{9.2}$$

Before mapping, the maximum and minimum PWM signals were determined. The azimuth thrusters' maximum was set to 0.22 by driving the boat and setting the maximum at a reasonable speed. The minimum was set to -.16 since a lower PWM resulted in aeration, meaning the thruster sucking in air. The maximum and minimum for the bow thrusters were also selected based on aeration and determined to be [-0.44,0.44].

9.2.1 Azimuth thruster

Since the force produced by the azimuth thrusters can vary greatly depending on the angle, they were mapped in 30° increments from $[-180°, 180°]$. The motors for the azimuth thrusters start rotating at a PWM signal of between 0.11 and 0.12, depending on battery voltage. With the range of PWM signals set to [-0.16,0.22] and the smallest PWM increment at 0.01, the thruster was mapped in two intervals for each angle. For positive [0.1,0.22] with a PWM increment of 0.01, and [-0.1,-0.16] with a PWM increment of -0.01. For the most accurate results, the thruster was run for 50$[s]$ for each measurement point. The signals were then used to calculate X, Y, and F as described in section 9.1. The calculated values for 0° can be seen in figure 9.2.1.

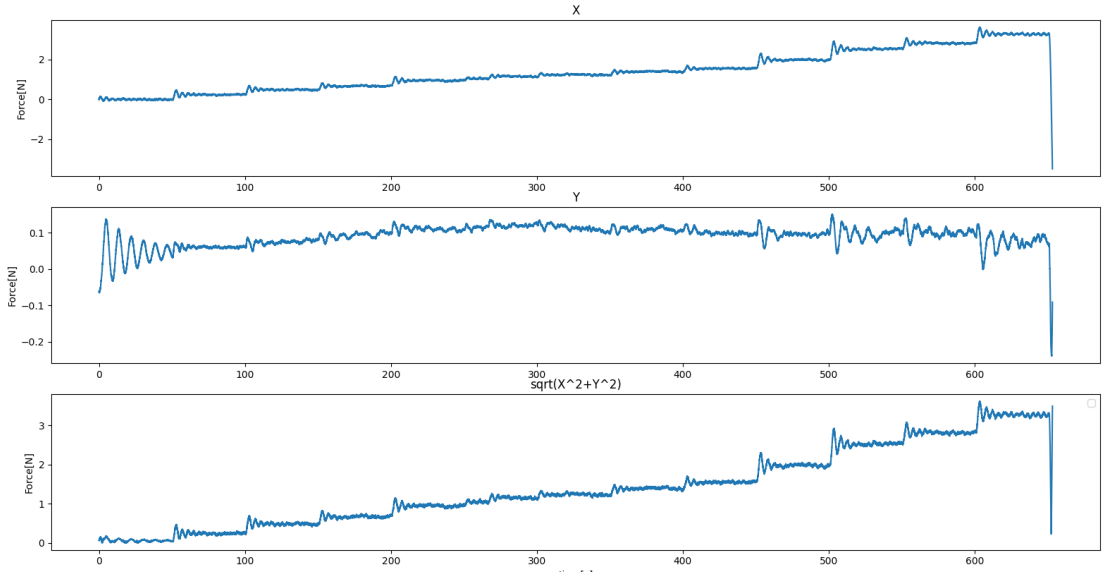


Figure 9.2.1: 0° starboard measurements

The measurements for all the plots is found in the appendix. When the forces had been measured for each angle, they were used to create a thrust map as seen in figure 9.2.2. In this polar plot, the angle is the angle of the thruster, the radius is the produced force, and the different lines are the different PWM signals.

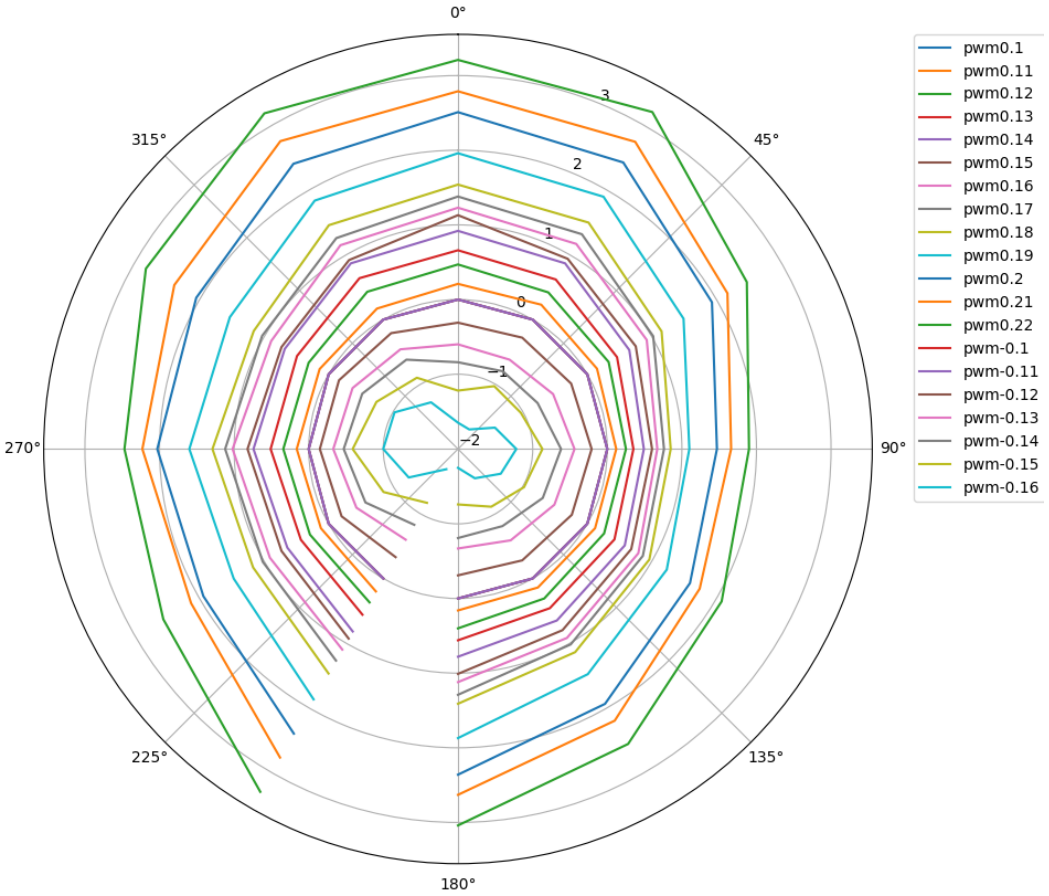


Figure 9.2.2: Mapping of starboard thruster

Here we can see the resulting loss of power when the thruster is pointed directly at the opposing thruster. The maximum then goes from around 3.2$[N]$ to about 1.9$[N]$, meaning around 40% decrease in performance. Since the thrusters are identical, the measurements from the starboard can be used on the port side thruster. By flipping the measurements around the X-axis in the body frame. The comparison between the two thrusters at 0° can be seen in figure 9.2.3

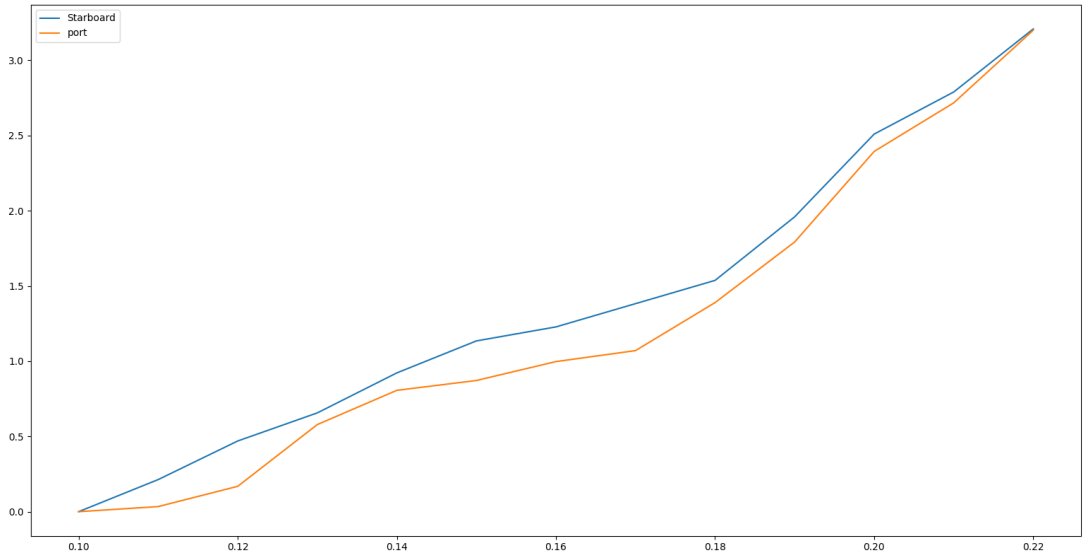


Figure 9.2.3: Comparison between starboard and port azimuth thruster

Here we can see that there are some deviations between the thrusters. Mainly for smaller forces, as here, the difference in resistance in each thruster has a more significant effect. However, the thrusters are similar enough that the thrust map for the starboard thruster can be used for both.

9.2.2 Bow thruster

For the mapping of the bow thruster, a larger PWM interval was selected as 0.02. The bow thruster motor starts spinning at a PWM signal of 0.11 or -0.11. So two intervals were mapped [-0.1,-0.44] and [0.1,0.44]

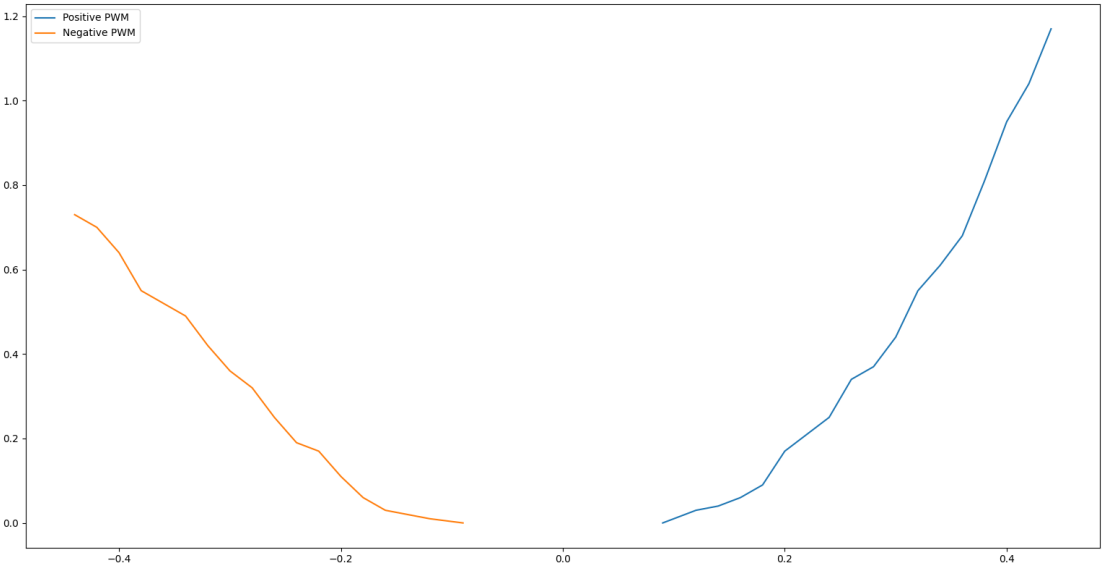


Figure 9.2.4: Bow thruster map

The bow thruster is more efficient and can produce a higher thrust in the positive Y direction than in the negative.

9.3 Force to PWM

When the forces had been measured and a map created, a function for transforming force to PWM signal was created. The force of a thruster with respect to the speed of the motor is quadratic, as can be observed in figure 9.2.4. Because the smallest increment for the PWM signal is 0.01, and the measurements were performed at this interval or 0.02, linear interpolation was chosen. For the bow thruster, standard linear interpolation was implemented.

$$pwm(F) = pwm(F_1) + \frac{pwm(F_2) - pwm(F_1)}{F_2 - F_1}(F - F_1) \tag{9.3}$$

for the azimuth thrusters, bilinear interpolation was implemented between angle and force as .

$$\begin{aligned} pwm(\alpha, F_1) &= \frac{\alpha - \alpha_1}{\alpha_2 - \alpha_1} pwm(\alpha_1, F_1) + \frac{\alpha - \alpha_1}{\alpha_2 - \alpha_1} pwm(\alpha_2, F_1) \\ pwm(\alpha, F_2) &= \frac{\alpha - \alpha_1}{\alpha_2 - \alpha_1} pwm(\alpha_1, F_2) + \frac{\alpha - \alpha_1}{\alpha_2 - \alpha_1} pwm(\alpha_2, F_2) \\ pwm(\alpha, F) &= \frac{F_2 - F}{F_2 - F_1} pwm(\alpha, F_1) + \frac{F - F_1}{F_2 - F_1} pwm(\alpha, F_2) \end{aligned} \tag{9.4}$$

CHAPTER
TEN

THRUST ALLOCATION

10.1 Pseudoinverse with filtering

Sørdalen (1997) proposed a solution for the thrust allocation problem by using a filtered pseudo inverse to derive slowly varying angles and singular value decomposition to allocate forces. The equations in this section are taken from Sørdalen (1997)

10.1.1 controller

Starting with the solution of the pseudoinverse gives a solution for ξ

$$\tau = B\xi \tag{10.1}$$

$$\xi = B^{\dagger}\tau \tag{10.2}$$

where B^{+} is the pseudoinverse of B. From this, the two force components for each thruster in ξ are filtered as shown in figure 10.1.1 to produce slow varying angles. The structure of the figure is from Sørdalen (1997), but reformulated to fit mathematical notations.

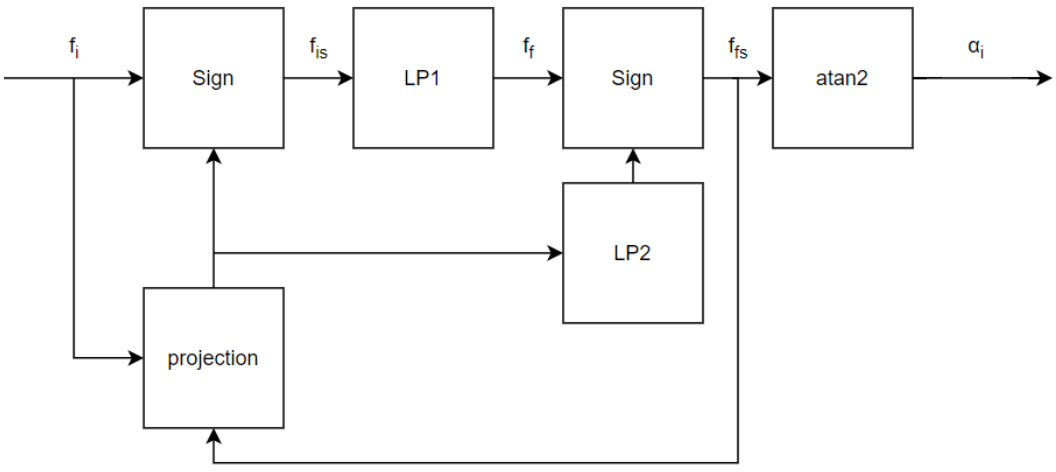


Figure 10.1.1: Slow varying angles

In this structure, firstly, the force vector is rotated 180 according to the sign of the projection of f_i on f_s. The resulting vector f_{is} is then run through a lowpass filter to

produce the slow varying force vector. The second sign and lowpass filter rotate the thruster 180 degrees depending on how often the vector f_i is changed. Meaning if the force vector is changed more than not meaning, the thruster produces negative force, operating in reverse. The thruster is rotated 180 degrees.

The next step is to use the slow varying angles to calculate the forces. This is done using singular value decomposition. singular value decomposition states that any matrix $A \in \mathbb{R}^{mxn}$ can be factored into

$$A = USV^T \tag{10.3}$$

Where $U \in \mathbb{R}^{mxm}$ and $V \in \mathbb{R}^{nxn}$ is orthogonal matrices, and S is

$$S = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \tag{10.4}$$

. σ is the singular values of A, they are positive and strictly decreasing, r is the rank of A. These values can be used for the pseudo inverse

$$A^+ = VS^+U^T \tag{10.5}$$

where

$$S_{ii}^+ = \frac{1}{\sigma_i} \tag{10.6}$$

Sørdalen (1997) proposes to use this pseudo inverse, with $A = B(\alpha_f)$

$$F = VS_\delta^+U^T\tau \tag{10.7}$$

$$S_{\delta,ii}^+ = s_i \tag{10.8}$$

The singular values are used as a tuning parameter to limit the forces when singularities occur.

$$s_i = \begin{cases} \frac{1}{\sigma_i} & \text{if } \frac{\sigma_i}{\sigma_1} > \delta_{hi} \\ \frac{1}{\delta_{hi}\sigma_1} & \text{if } \delta_{hi} \geq \frac{\sigma_i}{\sigma_1} \geq \delta_{lo} > 0 \\ 0 & \text{if } \frac{\sigma_i}{\sigma_1} < \delta_{lo} \end{cases} \tag{10.9}$$

with the tuning parameters $\delta_{hi} \geq \delta_{lo} > 0$

10.2 Quadratic programming

Quadratic programming is a way to solve quadratic optimization problems with linear constraints. The standard notation for these problems are

$$\min_x \frac{1}{2}x^THx + h^Tx \tag{10.10}$$

$$\text{s.t.} C(x) = c \tag{10.11}$$

$$Ax \leq b \tag{10.12}$$

. Here x is the vector of optimization variables, H, and h represents the quadratic function. The function is optimized with equality constraints C, and c, also inequality constraints A and b. This section outlines the proposed QP algorithms outlined in Johansen et al. (2004) and Scibilia and Skjetne (2012); all equations are from these.

10.2.1 Quadratic programming

In Johansen et al. (2004), a formulation of the thrust allocation problem was proposed.

$$\min_{F,\alpha,s}\{J = \sum_{i=1}^{m} W_i(F_i) + s^T Q s + (\alpha - \alpha_0)^T \Omega (\alpha - \alpha_0) \ + \frac{\varrho}{\epsilon + det(B(\alpha)B(\alpha)^T)}\} \quad (10.13)$$

$$\text{s.t.} \quad (10.14)$$

$$s = \tau - B(\alpha)F \quad (10.15)$$

$$F_{min} \leq F \leq F_{max} \quad (10.16)$$

$$\alpha_{min} \leq \alpha \leq \alpha_{max} \quad (10.17)$$

$$\Delta\alpha_{min} \leq \alpha - \alpha_0 \leq \Delta\alpha_{max} \quad (10.18)$$

. Where $W_i(F_i)$ represents the total power consumption for each thruster. s is the slack variable representing the error between wanted τ and calculated, with the weighting matrix Q. Ω is the cost matrix for the turning rate. The term $\frac{\varrho}{\epsilon+det(B(\alpha)B(\alpha)^T)}$ is the singularity avoidance term, with ϱ and ϵ as tuning parameters. For the constraints, the equality constraint $s = \tau - B(\alpha)F$ represents the thrust allocation problem. We can also implement saturation constraints, limit angles, and angular rates.

In Scibilia and Skjetne (2012), a new method for dealing with the singularity avoidance was introduced. Building upon Johansen et al. (2004) using future prediction to avoid singularities by using an estimated τ^+. This estimation can be obtained in several ways, i.e., mean environmental load or worst-case scenarios. These forces can then be used in the optimization with the parameters F^+,α^+, and s^+, as

$$\min_{\Delta F,\Delta\alpha,s,\Delta F^+,\Delta\alpha^+,s^+}\{J = F^T P F + \Delta\alpha^T \Omega \Delta\alpha + s^T Q s + \Delta F^{+T} P^+ F^+ + \Delta\alpha^{+T} \Omega^+ \Delta\alpha^+ + s^{+T} Q^+ s^+\} \quad (10.19)$$

$$\begin{aligned}
&\textbf{s.t.} \\
B(\alpha)F &= \tau + s \\
B(\alpha^+)F^+ &= \tau^+ s^+ \\
F = \Delta F + F_0, &F^+ = \Delta F^+ + F \\
\alpha = \Delta\alpha + \alpha_0, &\alpha^+ = \Delta\alpha^+ + \alpha \\
F_{min} \leq F &\leq F_{max} \\
F_{min} \leq F^+ &\leq F_{max} \\
\alpha_{min} \leq \alpha &\leq \alpha_{max} \\
\alpha_{min} \leq \alpha^+ &\leq \alpha_{max} \\
\Delta\alpha_{min} \leq \Delta\alpha &\leq \Delta\alpha_{max} \\
\Delta\alpha_{min} \leq \Delta\alpha^+ &\leq \Delta\alpha_{max}
\end{aligned} \quad (10.20)$$

10.2.2 Sequential quadratic programming

The two proposed solutions are nonconvex and due to the nonlinear term $B(\alpha)F = \tau - s$ and the term $\frac{\varrho}{\epsilon+det(B(\alpha)B(\alpha)^T)}$. These terms can be locally estimated around a point

using Taylor series expansion for small angle changes. This can be achieved by solving the optimization for each timestep, with $\alpha = \alpha_0 + \Delta\alpha$ and $F = F_0 + \Delta F$ and then using.

$$\begin{aligned} g(f,\alpha) &\approx a_0 + a_1 f + a_2 \alpha = \\ g(f_0,\alpha_0) + \left.\frac{\partial g}{\partial f}\right|_{f_0,g_0} (f - f_0) &+ \left.\frac{\partial g}{\partial \alpha}\right|_{g_0,g_0} (\alpha - \alpha_0) \end{aligned} \tag{10.21}$$

to estimate the nonlinear constraints. The solution proposed by Johansen et al. (2004)then becomes

$$\begin{aligned} \min_{\Delta\alpha,\Delta F,\Delta s} \{ J = \sum_{i=1}^{m} (\frac{dW_i}{dF_i}(F_{0,i})\Delta F_i + \frac{d^2 W_i}{dF_i^2}(F_{0,i})\Delta F_i^2) + s^T Q s + \Delta\alpha^T \Omega \Delta\alpha \\ + \frac{d}{d\alpha} \left(\frac{\varrho}{\epsilon + det(B(\alpha)B(\alpha)^T)} \right)_{\alpha=\alpha_0} \Delta\alpha \} \end{aligned} \tag{10.22}$$

$$\begin{aligned} s + B(\alpha_0)\Delta F + \frac{\partial}{\partial \alpha} (B(\alpha)F)_{\substack{\alpha=\alpha_0 \\ F=F_0}} \Delta\alpha = \tau - B(\alpha_0)F_0 \\ F_{min} - F_0 \leq \Delta F \leq F_{max} - F_0 \\ \alpha_{min} - \alpha_0 \leq \Delta\alpha \leq \alpha_{max} - \alpha_0 \\ \Delta\alpha_{min} \leq \Delta\alpha \leq \Delta\alpha_{max} \end{aligned} \tag{10.23}$$

The same can be done for the equality constraints in the solution proposed by Scibilia and Skjetne (2012). Since most QP-solvers require the problem to be formulated as stated in 10.12 when implemented, the equation is reformulated to standard notation.

10.3 Maneuvering

In Skjetne (2023), it is proposed that the thrust allocation problem can be solved as a maneuvering problem. All equations in this part are taken from Skjetne (2023)

10.3.1 controller

Starting with creating the manifold using the weighted pseudoinverse

$$F(t) := \{ \xi \in \mathbb{R}^p : \theta \quad \text{s.t} \quad \xi = \xi_d(t,\theta) \} \tag{10.24}$$

with

$$\xi_d(t,\theta) = B_W^{\dagger} \tau_d + Q\theta \tag{10.25}$$

Here θ is the parametrization variable, Q is the nullspace of B, and ξ_p is the particular solution to ξ. The geometric task then becomes

$$\lim_{t \to \inf} |\xi - \xi_d(t,\theta(t))| = 0 \tag{10.26}$$

For the desired dynamic assignment, a convex cost function $J(t,\theta)$ is used, and the dynamic task is then set to the dynamics of a continuous steepest descent.

$$\dot{\theta} \to \nu(t,\theta) := -\gamma \bigtriangledown_\theta J(t,\theta)^T \tag{10.27}$$

The saturation constraints are then implemented as control barrier functions. Limiting the maximum force each thruster can produce.

By setting

$$J(t,\theta(t)) = \frac{1}{2} \xi_d(t,\theta)^T W \xi_d(t,\theta) \tag{10.28}$$

the solution becomes a reference-filter, with inputs τ_d and $\dot{\tau_d}$. The outputs are $F_{d,i} = |\xi_i|$ and $\alpha_{d,i} = atan2(\xi_{i,y}, \xi_{i,x})$, and dynamics

$$
\begin{aligned}
\xi_p &:= B_W^\dagger \tau & \dot{\xi}_p &:= B_W^\dagger \dot{\tau} \\
\xi_{d,i} &:= \xi_{p,i} + Q\theta & \tilde{\xi} &:= \xi_i - \xi_{d,i} \\
J &:= \sum_{i=1}^{m_1} \frac{w_i}{2} \xi_{d,i}^T + \xi_{d,i} + \sum_{m_1+1}^{m_2} \frac{w_i}{2} \xi_{d,i}^2 & J^\theta &= \theta^T \sum_{i=1}^{m} w_i Q_i^T Q_i \\
V &:= \sum_{i=1}^{m_1} \frac{w_i}{2} \tilde{\xi}_i^T \tilde{\xi}_i + \sum_{m_1+1}^{m} \frac{w_i}{2} \tilde{\xi}_i^2 & V^\theta &:= -\sum_{i=1}^{m} w_i \tilde{\xi}_i^T Q_i \\
\upsilon &:= -\gamma (J^\theta)^T \\
a_i &:= \rho(\xi_i^T \xi_i - F_{i,max}^2) & b_i &:= 2\xi_i \\
k_i &:= -\bar{U}_i \frac{\tilde{\xi}_i}{|\tilde{\xi}_i| + \epsilon} + \dot{\xi}_{p,i} + Q_i \upsilon \\
\dot{\theta} &= \upsilon - \mu (V^\theta)^T \\
\dot{\xi}_i &:= k_{i,safe} := \begin{cases} ki & a_i + b_i^T k_i \leq 0 \\ k_i - \frac{a_i + b_i^T k_i}{b_i^T b_i} b_i & a_i + b_i^T k_i > 0 \end{cases}
\end{aligned}
\tag{10.29}
$$

where

$B, B_W^\dagger$	Configuration matrix and weighted pseudo inverse	(10.30)
$F_{i,max} > 0$	Saturation limit	
$W > 0$	Weight matrix	
$\gamma > 0$	steepest decent gain for $J(t, \theta)$	
$\mu > 0$	Gradient update law gain $\mu > \gamma$	
$\rho > 0$	Safety gain to avoid saturation limits	
$\bar{U}_i > 0$	rate limit	
$\epsilon > 0$	small parameter to avoid dividing by 0	

Using another cost function, the angular rates of the azimuth thrusters can also be constrained. A cost function can be created based on the dot product rule for a two-dimensional vector.

$$J_i(\xi_i) = w_i |\xi_i| - a_o^T \xi_i, \qquad w_i \geq 1 \tag{10.31}$$

Here $a_0 = [cos(\alpha_0), sin(\alpha_0)]$ where α_0 is the previous thruster angle. Note that the gradient for this cost function does not exist for $|\xi| = 0$. The resulting reference filter is the same, but with

$$J := \sum_{i=1}^{m_1} \left[w_i |\xi_{d,i}| - a_{0,i}^T \xi_{d,i} \right] + \sum_{m_1+1}^{m} w_i |\xi_{d,i}| \tag{10.32}$$

$$J^\theta := \sum_{i=1}^{m_1} \left(w_i \frac{\xi_{d,i}^T}{|\xi_{d,i}| + \epsilon} - a_{0,i}^T \right) Q_i + \sum_{m_1+1}^{m} \left(w_i \frac{\xi_{d,i}^T}{|\xi_{d,i}| + \epsilon} \right) Q_i \tag{10.33}$$

and requiring $[w_0, ..., w_{m1}] \geq 1$

10.4 Results and discussion

10.4.1 Constraints

The implemented thrust allocation algorithms have constraints, as seen in table 10.4.1.

	Pseudoinverse w filtering	QP	maneuvering
Saturation constraint		X	X
Force rate constraint			X
Angle constraint		X	
Angle rate constraint	X	X	X

Table 10.4.1: Constraints for the different algorithms.

Different tests have been simulated to compare how each algorithm handles the various constraints.

Saturation

Starting with testing the saturation constraints. To test this, a τ_d with increasing force in the yaw direction was chosen. The signal sent is $\tau_d = [0, t/5, 0]$. This causes the saturation of the bow thruster at around 12 seconds. In figure 10.4.2, we see that all but the pseudoinverse algorithm limits its signals. The QP algorithms and the maneuvering algorithm handles this constraint differently. The Qp algorithms minimize the error by applying more force to azimuth thrusters to account for the loss from the bow thruster. The maneuvering algorithm follows the solution given by the pseudoinverse and constrains the bow thruster. In figure 10.4.3, we can see that the QP algorithms choose to run one thruster in reverse. Since the azimuth thruster can produce less force in the reverse direction, the QP algorithm ends with all thrusters saturated. Also note the change in angle at t = 0, as this is discussed later.

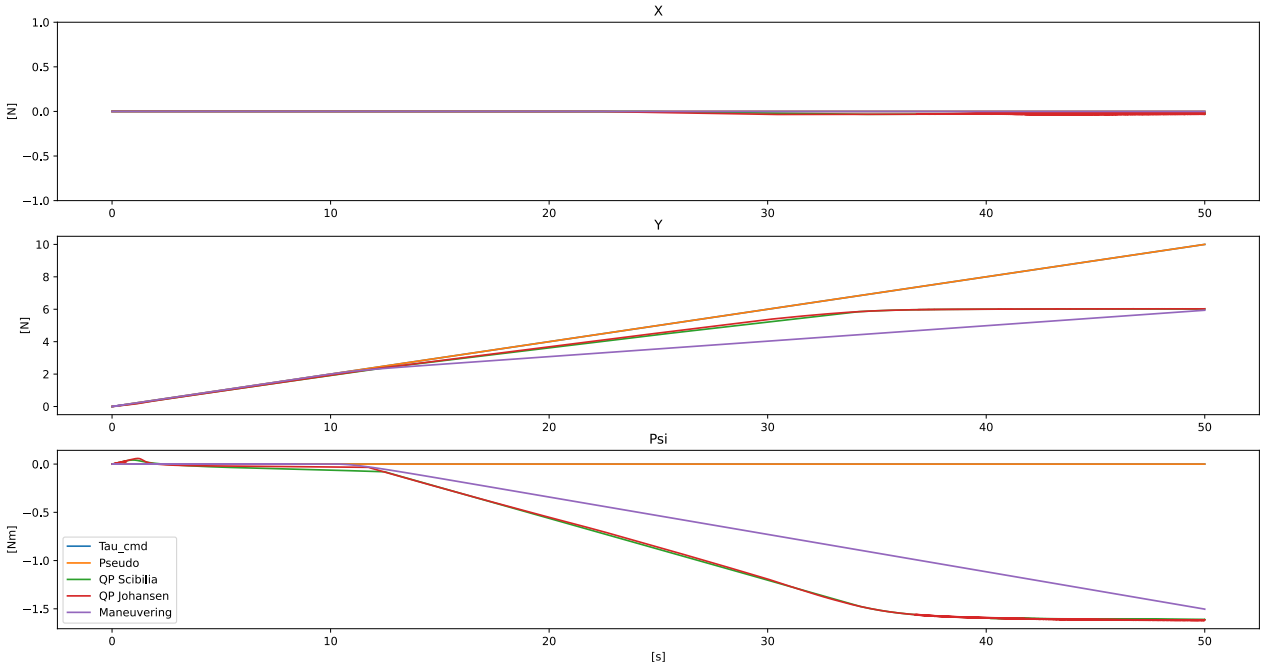


Figure 10.4.1: Saturation constraint comparison tau

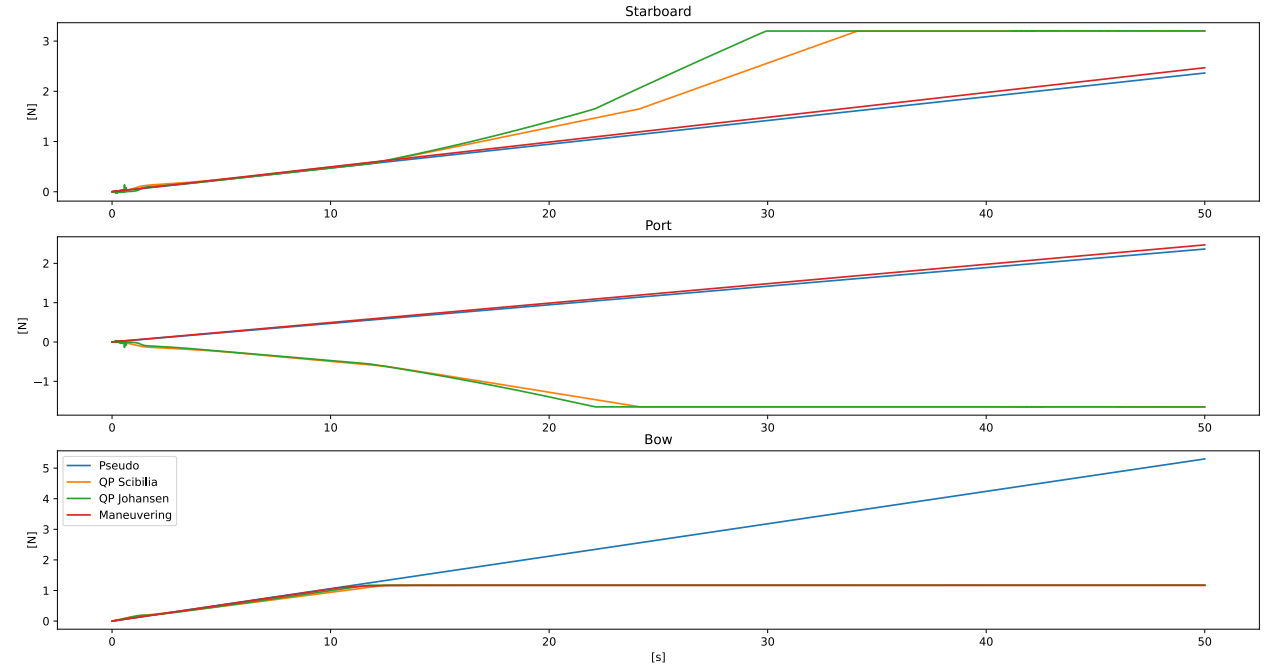


Figure 10.4.2: Saturation constraint comparison force of thrusters

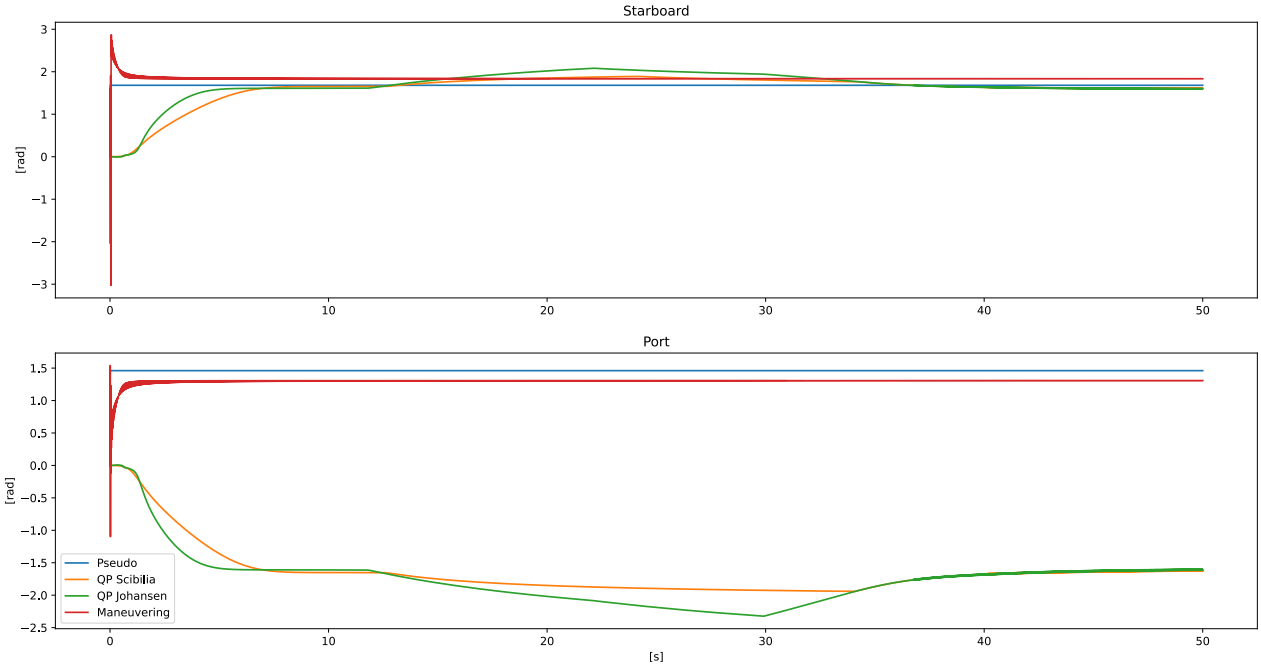


Figure 10.4.3: Saturation constraint comparison angles

Rate constraints

To test the rate constraint capabilities of the algorithms, the signal

$$\tau_d = \begin{cases} (2,0,0) & \text{if} \quad t < 5 \\ (0,2,0) & \text{if} \quad t >= 5 \end{cases} \tag{10.34}$$

was used. The commanded thrust vector causes the thrusters to rotate 90 degrees to meet the desired force. The signal starts with $2[N]$ in surge due to the pseudoinverse and maneuvering algorithms. In figure 10.4.2, we can see that if $\xi = 0$, the rate constraint for the angle does not work. Due to the algorithms limiting the change in the x and y directions of the decomposed force vectors. For $\xi > 0$, however, the angle rate constraint works. In figure 10.4.5, we see that only the maneuvering algorithm limits the change in force. The other algorithms make jumps that can cause wear on the components. We can also see that the maneuvering algorithm performs better with the sudden change, as the other algorithms get spikes in the commanded forces. In figure 10.4.6, we can see that all algorithms limit the angle rate. Here we can also see that the rate constraint is linear for the QP algorithms.

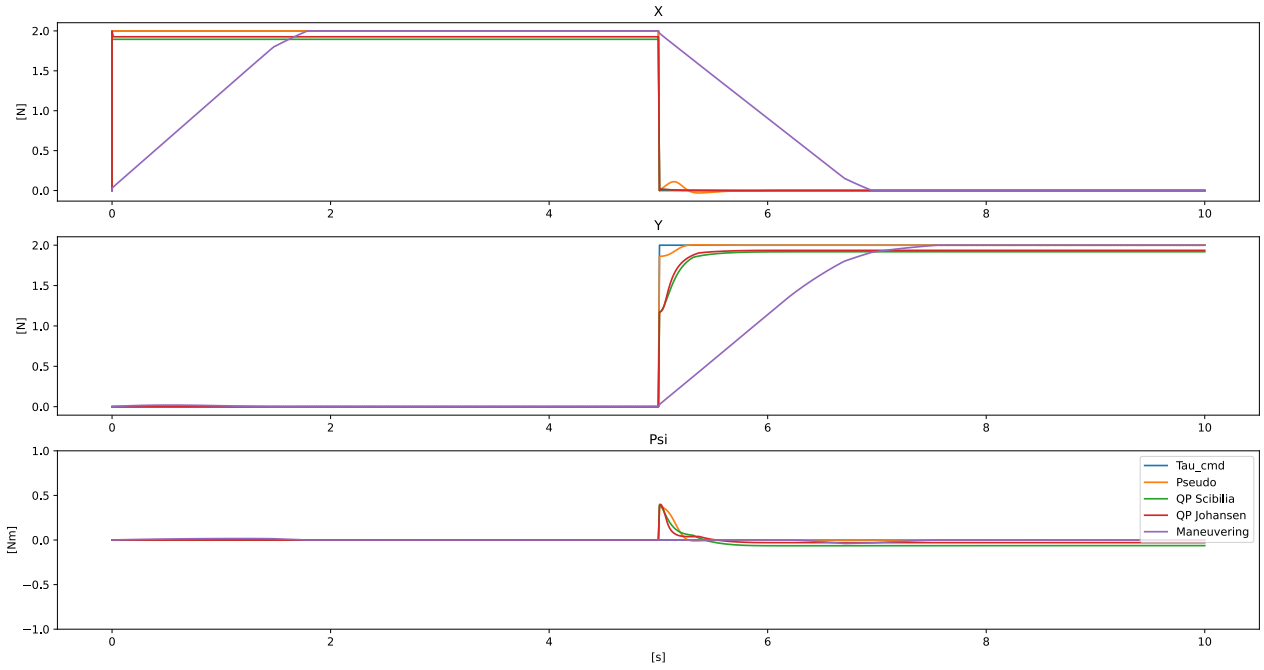


Figure 10.4.4: Rate constraints comparison tau

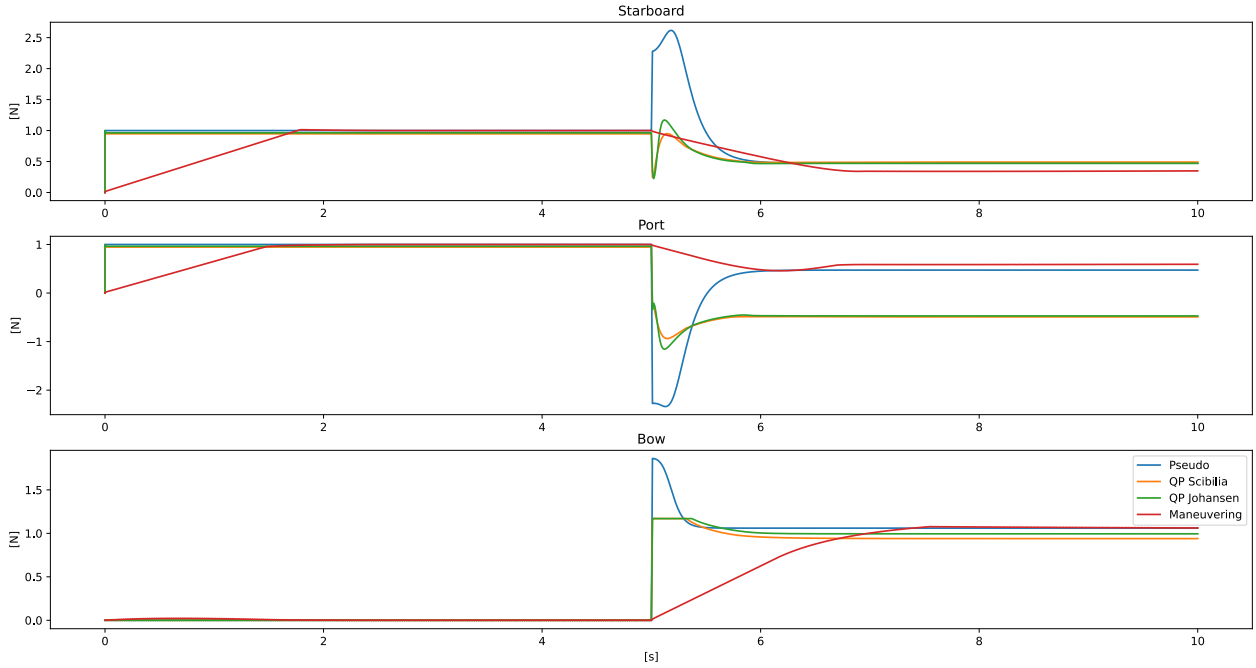


Figure 10.4.5: Rate constraints comparison force of thruster

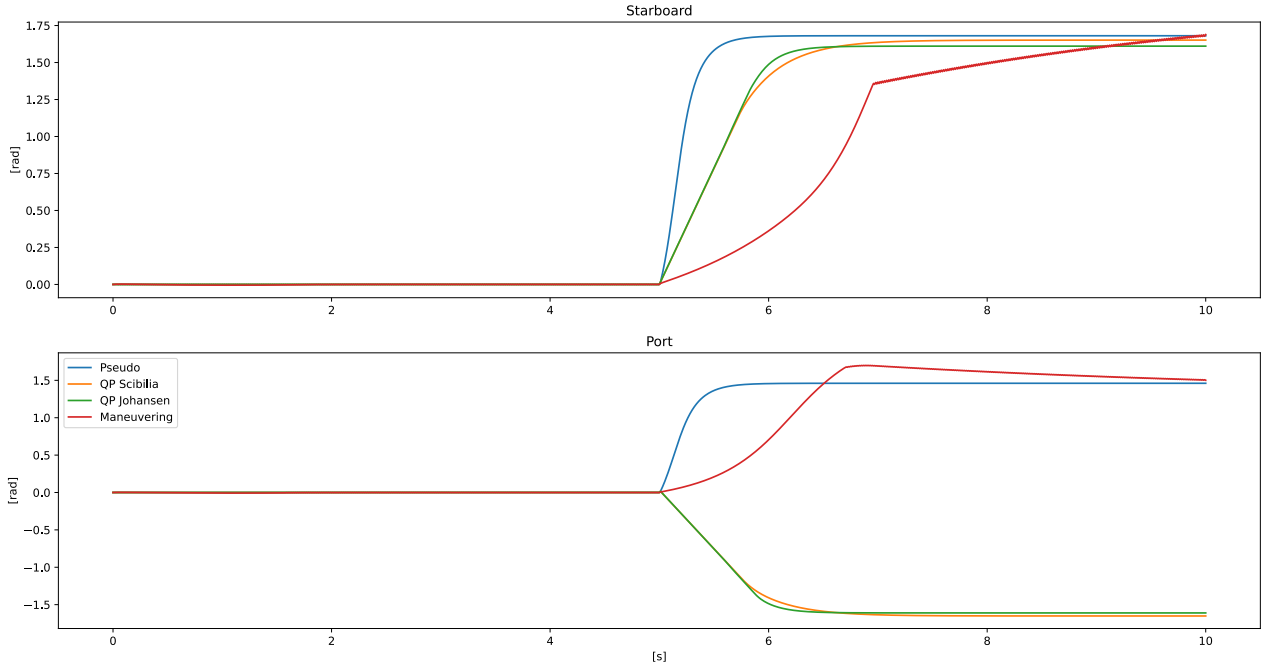


Figure 10.4.6: Rate constraints comparison angles

Angle constraint

To test the angle constraints of the algorithms a $\tau_d = [2\cos(2\pi\frac{t}{180}), 2\sin(2\pi\frac{t}{180}), 0]$ was used. Creating a force with a length of $2[N]$ that slowly sweeps around the 360°. The thrusters have a maximum and minimum angle of ±180°. If the commanded angle is higher or lower than this, the thruster does a sweep to arrive at the opposite value. The pseudoinverse and maneuvering do this, as seen in figure 10.4.9. It is worth noting that the algorithms work continuously over this area, and the atan2 function at the end causes this jump. It also explains why the rate constraints seem to not be working since it is a small change in angle for the algorithms. The QP algorithms, however, do have angular constraints. Causing the thrusters to be stuck at 180° which causes the errors seen in figure 10.4.7. The algorithms use some time before changing the thrusters to reverse, as seen in figure 10.4.8. This test will be in the two upcoming sections.

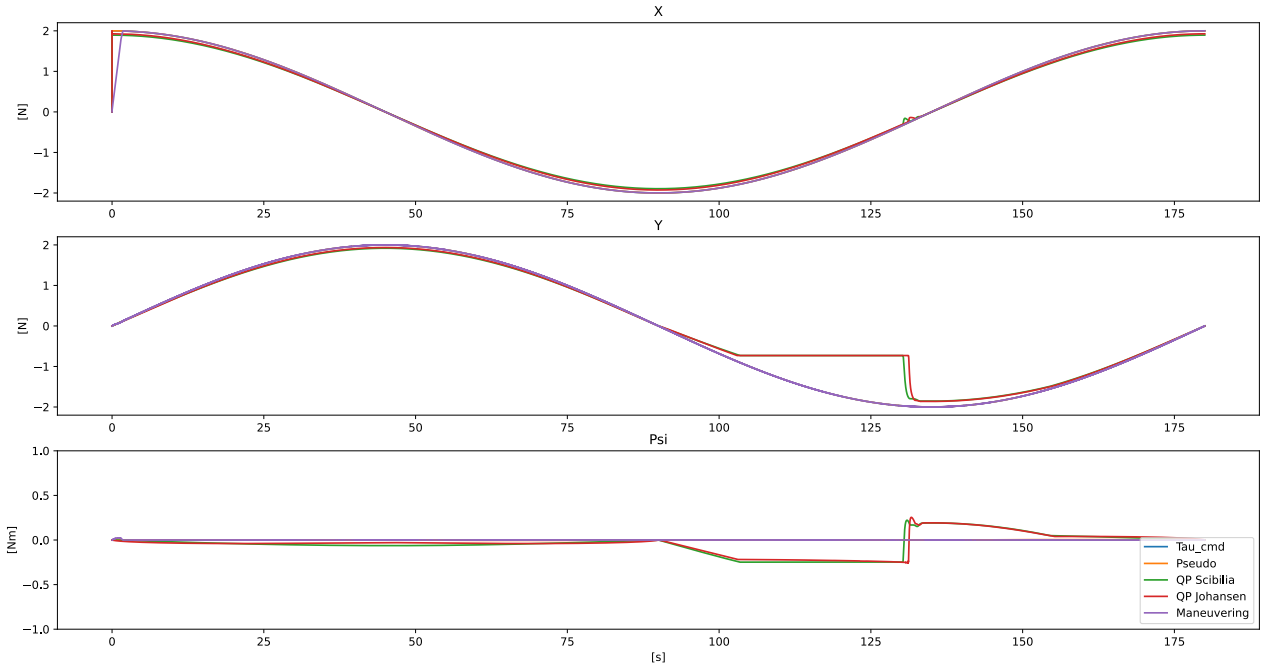


Figure 10.4.7: Angle constraint comparison tau

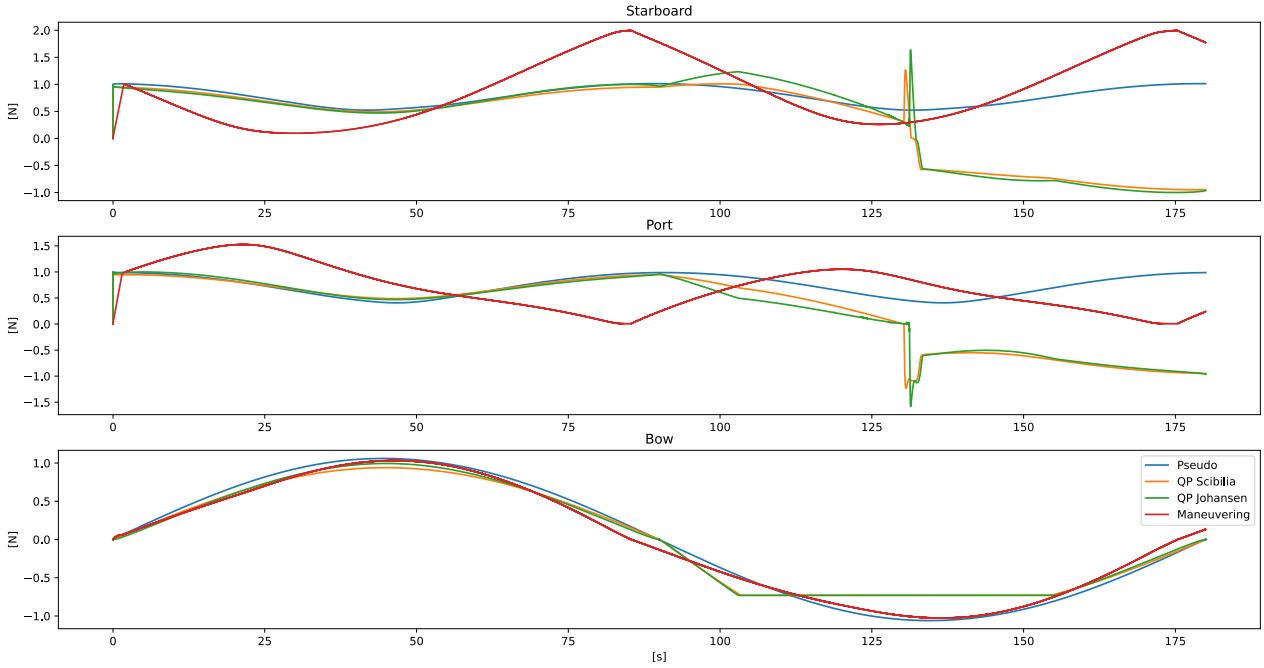


Figure 10.4.8: Angle constraint comparison force of thrusters

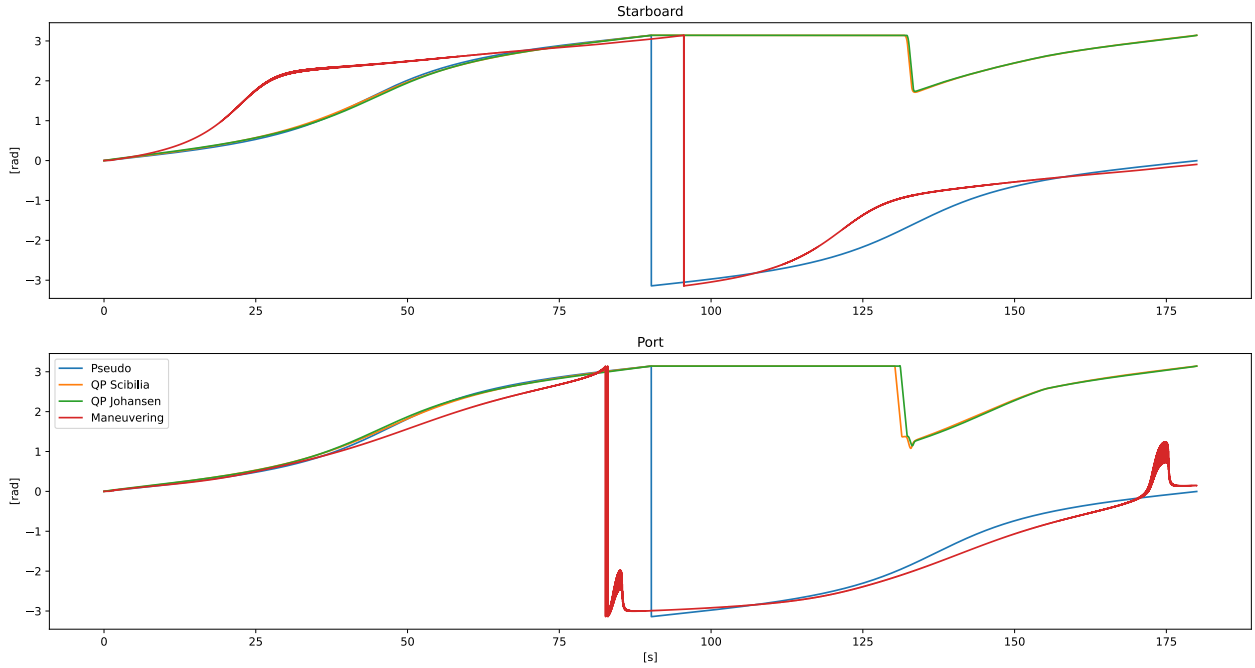


Figure 10.4.9: Angle constraint comparison angle of thrusters

10.4.2 Computation times

To compare the computational performance of the algorithm, the time for each iteration of the algorithm was recorded for the angle constraint test. Figure 10.4.10 shows the time for each iteration. The time for each iteration is acceptable for all of the algorithms. The maneuvering algorithm is the fastest or least computationally heavy, closely followed by the QP algorithm of Scibilla. They are almost twice as fast as the other QP algorithm and the pseudoinverse.

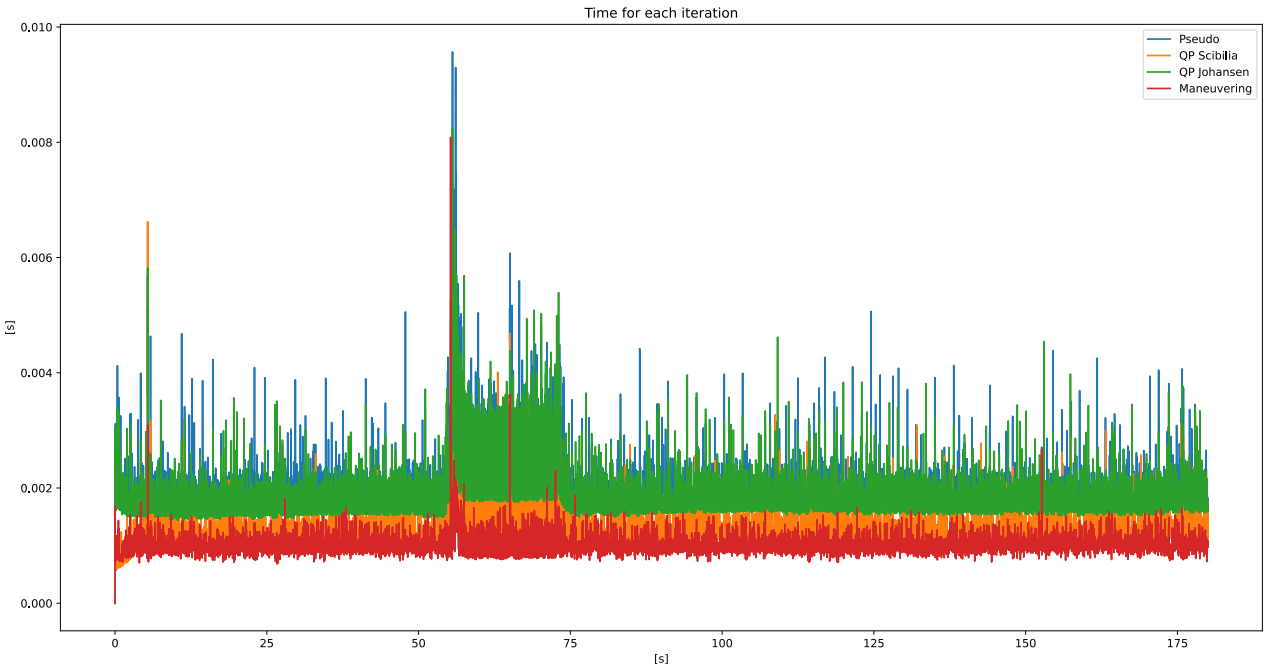


Figure 10.4.10: Computational time for each iteration

10.5 Realtime implementation

The setup from section 9.1 and the commanded force vector from the angle constraint was used to test the functionality of the whole system. Using the force to PWM function and the algorithms running on the raspberry pi. This is to show that the algorithms work in real-time and verify that the force mapping is working. Figure 10.5.1 and figure 10.5.4 show the force created when the thruster does the 360° sweep at 90[s]. In figure 10.5.2 and 10.5.3, we can see the same plateau that arises from the constraint on the angles as seen in figure 10.4.7. The thrust mapping performs quite well when looking at the forces produced in the X and Y directions. There is, however, a significant moment. This can be due to the thrust mapping and the force to PWM function. However, since this large yaw moment was not observed during the basin trial, it could also be measurement errors, which could be due to the lack of constraint in the front of the vessel.

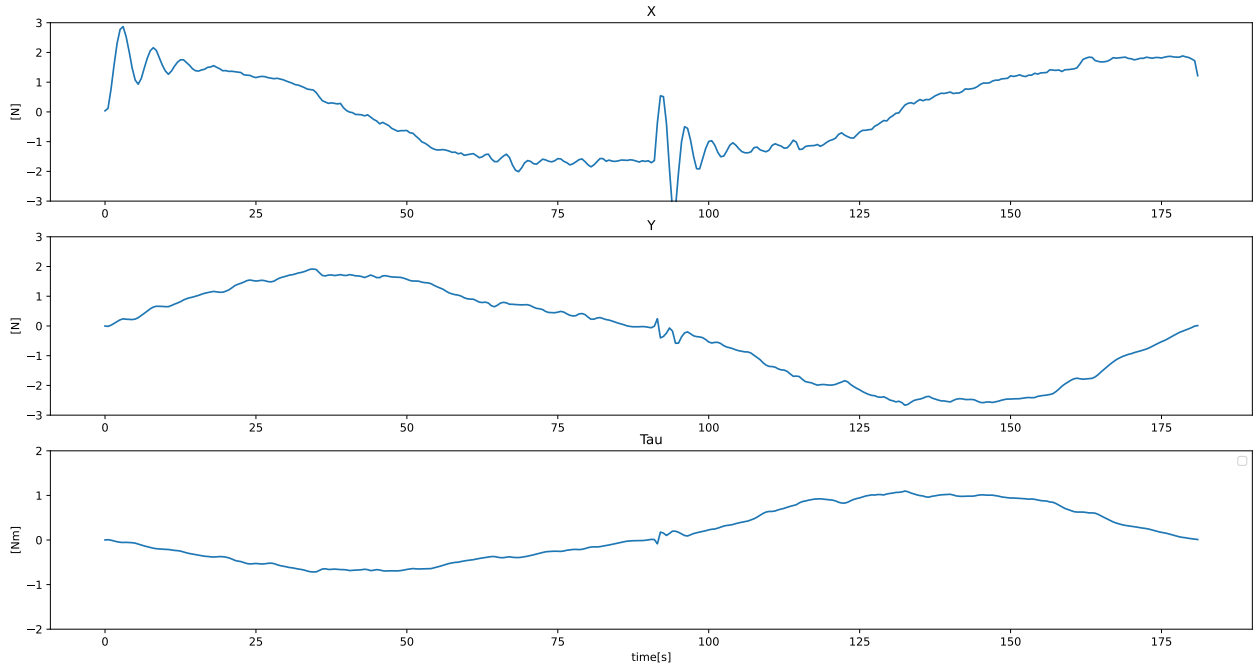


Figure 10.5.1: Realtime implementation pseudoinverse

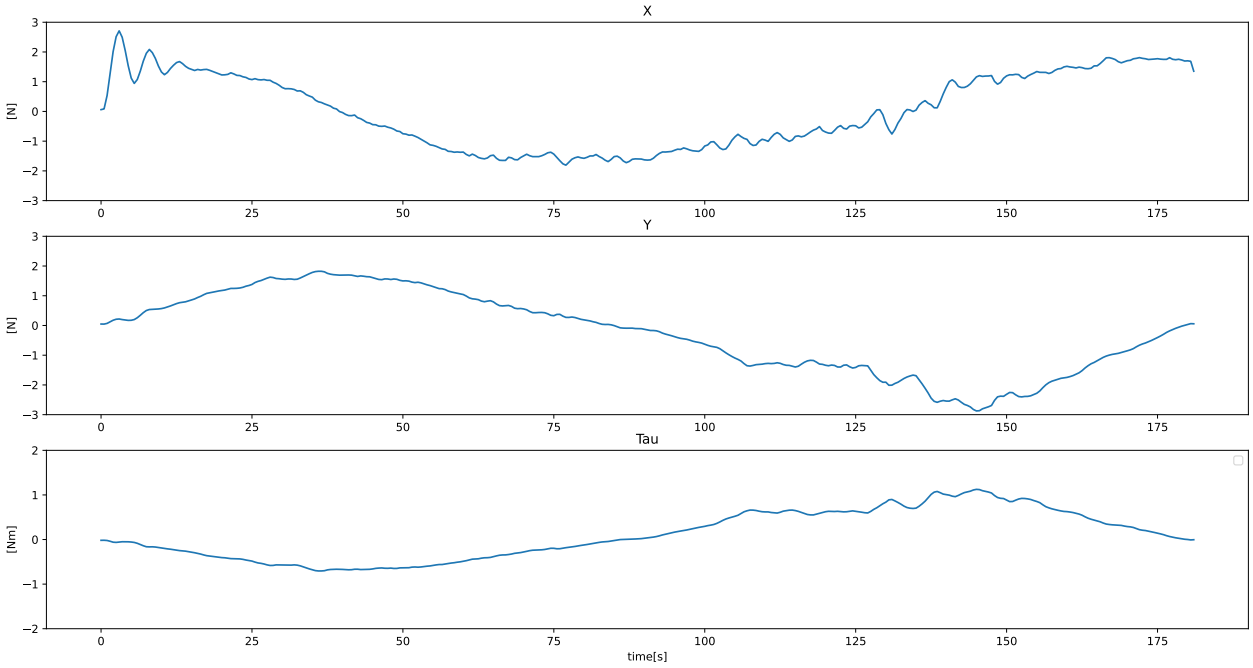


Figure 10.5.2: Realtime implementation QP Johansen

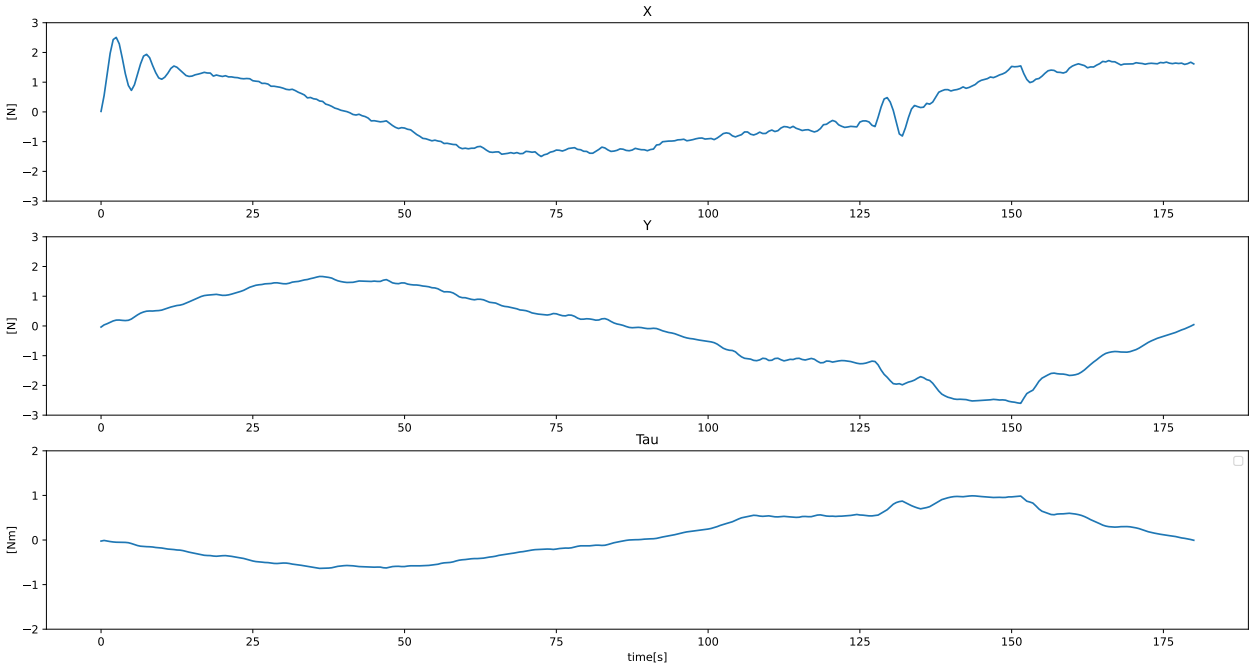


Figure 10.5.3: Realtime implementation QP Scibilla

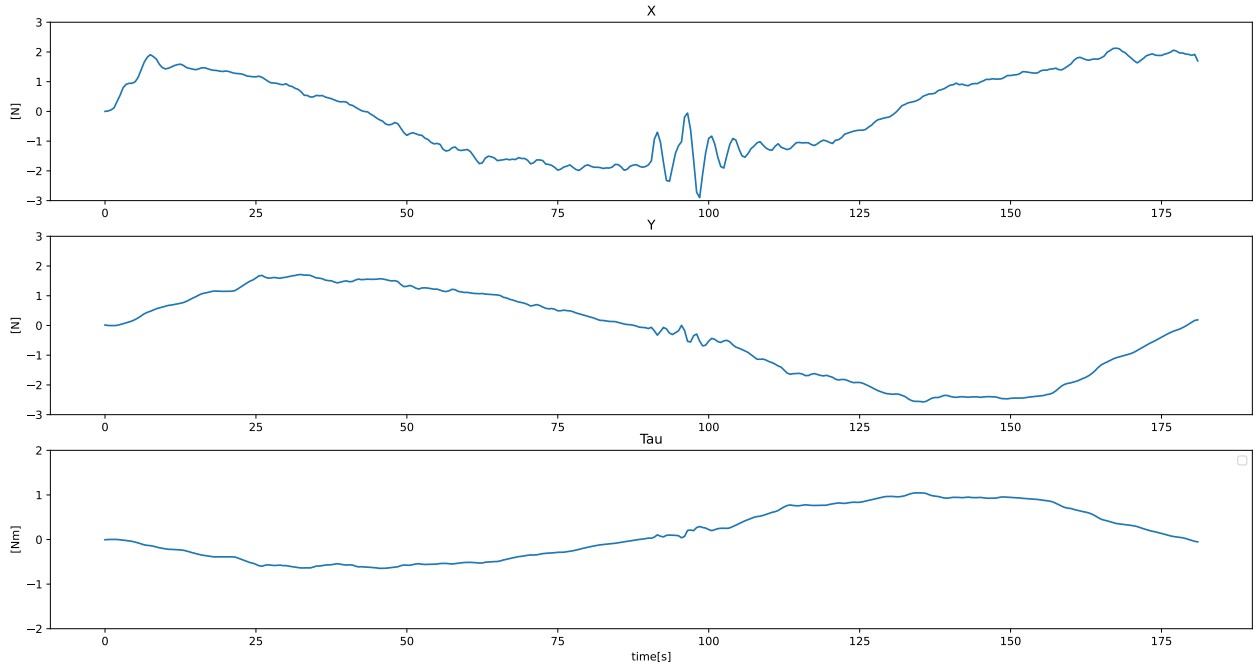


Figure 10.5.4: Realtime implementation maneuvering

CHAPTER
ELEVEN

CONCLUSIONS & FURTHER WORK

11.1 Conclusion

11.1.1 Development of C/S Jonny

The new model is operational and works very well for the most part. The modular design and control box means the model has few loose wires and components. The bow thruster operates smoothly and with minimal vibrations. The rotation of the azimuth thrusters also works very well, with precise movements. The azimuth thrusters have coarse control of the produced force, with only 12 points of control in the positive direction and 6 points in the reverse direction. This problem needs to be addressed to achieve optimal operation of the new vessel.

11.1.2 Thrust mapping

The thrust mapping provides a good representation of the produced forces of the thrusters. It takes into account losses due to thruster hull interactions. The algorithm to map forces to control signals works well. Though, it can vary based on the voltage of the battery. However, it can be beneficial to implement a low-level thrust controller, such as a shaft speed or torque controller.

11.1.3 Thrust allocation

All of the thrust allocations were implemented and tested. They all work well and can be implemented and run in real-time on the raspberry pi. The recommendation for the rest of the fleet is to implement the maneuvering algorithm if no angle constraints are needed. If angle constraints are needed, the recommendation is the QP algorithm of Scibilla. Even though the two QP algorithms are very similar, this algorithm requires less computational power, and is better at singularities avoidance.

11.2 Further work

There remains some work for CSJ to be finished.

- **Improve the azimuth thrusters:** the azimuth thrusters should be improved by either
  - Change thruster to one with a smaller diameter
  - Install gears between the driveshaft coupling and motor. Such that the motor could spin at higher speeds.
- **Hydrodynamic analysis:** Hydrodynamic analysis of CSJ was outside this project's scope. It should be performed to provide the necessary dynamics for a simulator and an observer.
- **Implement the rest of the nodes described in fig 5.4.1 such as:**
  - Observer
  - Controller
  - SimVessel
  - Guidance
- **Implement the thrust allocation algorithms for the rest of the Cyber ship fleet**

BIBLIOGRAPHY

Aeronaut (2022), 'Jonny harbor tugboat'.
URL: *https://aero-naut.de/produkt/jonny-hafenschlepper*

Authority, N. M. (2023), 'Bollard pull test procedure'.
URL: *https://www.sdir.no/en/shipping/legislation/directives/bollard-pull-test-procedure/*

Bauer-Modelle (2022), 'Bow thruster 28/32 x 180mm'.
URL: *https://www.bauer-modelle.com/epages/Bauer$_U$we46269592.sf/en$_G$B/?ObjectPath = /Shops/Bauer$_U$we46269592/Products/*702093

Bjørnø, J. (2016), Thruster-assisted position mooring of C/S Inocean Cat I Drillship, MSc thesis, Norwegian Univ. Sci. & Tech., Trondheim, Norway.

Bjørnø, J. (2015), 'Development of the c/s inocean cat i drillship model'.

Boats, C. M. (2022), 'Cem schottel drive unit 70mm'.
URL: *https://www.cornwallmodelboats.co.uk/acatalog/CEM070.htmlSID=1808*

DFROBOT (2022), 'Gravity i2c_hub'.
URL: *https://www.dfrobot.com/product2179.html?gclid=CjwKCAiA%5C-dCcBhBQEiwAeWidtdms1%5C_Lp9t8ijH1Pem6Px0xFyrLauoZtc2RfRzp0kJW3uK8nraDKPxoC1yAQ*

Digi-Key (2022), 'Sen0250'.
URL: *https://www.digikey.no/no/products/detail/dfrobot/SEN0250/9356335?s=N4IgTCBcDaIM4F*

Distrelec, E. (2022), '815 - pca9685 16-kanalers 12-biters pwm-/servodriver, adafruit'.
URL: *https://www.elfadistrelec.no/no/pca9685-16-kanalers-12-biters-pwm-servodriver-adafruit-815/p/30091222?track=true&no-cache=true&marketingPopup=false*

EEPOWER (2023), 'An intro to pulse-width modulation for control in power electronics'.
URL: *https://eepower.com/technical-articles/an-intro-to-pulse-width-modulation-for-control-in-power-electronics/*

Elefun (2022*a*), '4s 8000mah -100c - gens ace ec5 bashing serie'.
URL: *https://www.elefun.no/p/prod.aspx?v=54518*

Elefun (2022*b*), 'Dualsky eco 3520c v2 820kv 210gram'.
URL: *https://www.elefun.no/p/prod.aspx?v=39078*

Elefun (2022*c*), 'Os oca-150 50a esc w/bec 2-6s'.
URL: *https://www.elefun.no/p/prod.aspx?v=21059*

Elefun (2022*d*), 'Os oma-5010-810kv bl .52 4t-size'.
URL: *https://www.elefun.no/p/prod.aspx?v=23741*

Elvenes, S. A., Midtun, E. and Kvebæk, M. L. (2023), 'Development of the "c/s voyager" model'.

Fossen, T. I. (2011), *Handbook of Marine Craft Hydrodynamics and Motion Control*, John Wiley & Sons Ltd.

Frederich, P. (2016), Constrained optimal thrust allocation for c/s inocean cat i drillship, MSc thesis, Norwegian Univ. Sci. & Tech., Trondheim, Norway.

Idland, T. K. (2015), 'Marine cybernetics vessel cs saucer'.

Johansen, T. A. and Fossen, T. I. (2013), 'Control Allocation - A Survey', *Automatica* **49**, 1087–1103.

Johansen, T. A., Fossen, T. I. and Berge, S. P. (2004), 'Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming', *IEEE Trans. Ctrl. Sys. Tech.* **12**(1), 211–216.

Johansen, T. A., Sørensen, A. J., Nordahl, O. J., Mo, O. and Fossen, T. I. (2007), 'Experiences from hardware-in-the-loop (hil) testing of dynamic positioning and power management systems'.

NTNU (2023), 'Marine cybernetics teaching laboratory'.
URL: *https://www.ntnu.edu/imt/lab/cybernetics*

Pi, R. (2022), 'Buy raspberry pi 4 model b'.
URL: *https://www.raspberrypi.com/products/raspberrypi4modelb/*

Robotis (2022*a*), 'Dynamixel mx-28ar 6pcs bulk'.
URL: *https://www.robotis.us/dynamixel-mx-28ar-6pcs-bulk/*

Robotis (2022*b*), 'U2d2'.
URL: *https://www.robotis.us/u2d2/*

RS-Components (2022*a*), 'Huco bellows coupling, 20mm outside diameter, 5mm bore, 31mm length coupler'.
URL: *https://no.rs-online.com/web/p/couplings/6932473*

RS-Components (2022*b*), 'Mean well dc-dc converter, 5v dc/ 3a output, 9.2 18 v dc input, 15w, chassis moun'.
URL: *https://no.rs-online.com/web/p/dc-dc-converters/0183821*

Scibilia, F. and Skjetne, R. (2012), Constrained control allocation for vessels with azimuth thrusters, *in* 'Proc. IFAC Conf. Manoeuvering and Contr. Marine Crafts', Vol. 9, IFAC, Arenzano, Italy.

Skjetne, R. (2023), Dp thruster configuration and thrust allocation, Technical note, Norwegian Univ. Sci. & Tech., Trondheim, Norway.

Skjetne, R., Jørgensen, U. and Teel, A. R. (2011), Line-of-sight path-following along regularly parametrized curves solved as a generic maneuvering problem, *in* 'Proc. IEEE Conf. Decision & Control', Vol. 50th, Inst. Electrical and Electronics Engineers, Orlando, USA.

Skåtun, H. N. (2011), 'Development of a dp system for cs enterprise i with voith schneider thrusters'.

Sørensen, J. A. (2012), *Marine Control Systems: Propulsion and Motion Control of Ships and Ocean Structures*, 2 edn, Norwegian Univ. Sci. & Tech., Trondheim, Norway. Report UK-12-76.

stock gears, K. (2023), 'Gear module'.
URL: *https://khkgears.net/new/gear-module.html*

Sørdalen, O. (1997), 'Optimal thrust allocation for marine vessels'.

APPENDICES

PSEUDO INVERSE WITH FILTERING

```
import time
import numpy as np
import math
from scipy.signal import butter, lfilter
import matplotlib.pyplot as plt


class pseudo:
    def __init__(self):
        self.pos_x = np.array([-0.425, -0.425, 0.37])
        self.pos_y = np.array([0.07, -0.07, 0])
        self.f_max = np.array([4.4, 4.4, 1.2])
        self.type = np.array(["azi", "azi", "tunnel"])
        self.num_thruster = 3
        self.num_azi = 2
        self.h = 0.1
        self.delta_high = .2
        self.delta_low = .01
        self.lp1 = np.zeros((self.num_azi*2, 1))
        self.lp2 = np.zeros((self.num_azi, 1))
        self.Tfs = np.zeros((self.num_azi, 2))
    def B(self, angles):
        B = np.zeros((self.num_thruster, 3))
        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                B[0][i] = np.cos(angles[i])
                B[1][i] = np.sin(angles[i])
                B[2][i] = self.pos_x[i] * np.sin(angles[i]) - self.pos_y[i] *
                ↪  np.cos(angles[i])
            else:
                B[0][i] = 0
                B[1][i] = 1
                B[2][i] = self.pos_x[i]
        return B
    def B_ext(self):
        B = np.empty((3, 1))
        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                array = np.empty((3,2))
                array[0][0] = 1
                array[1][0] = 0
                array[2][0] = -self.pos_y[i]
                array[0][1] = 0
                array[1][1] = 1
                array[2][1] = self.pos_x[i]
                B = np.append(B,array,axis=1)

            else:
                array = np.empty((3, 1))
```

```
            array[0][0] = 0
            array[1][0] = 1
            array[2][0] = self.pos_x[i]
            B = np.append(B, array, axis=1)
    B = np.delete(B,0,1)
    return B

def pseudoinverse_ext(self, tau_cmd):
    B = self.B_ext()
    F = np.zeros((self.num_thruster, 1))
    alpha = np.zeros((self.num_thruster, 1))
    F_ext = np.matmul(np.linalg.pinv(B), tau_cmd)
    counter = 0
    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            F[i] = np.sqrt((F_ext[i + counter]) ** 2 + (F_ext[i + counter + 1]) **
            ↪  2)
            alpha[i] = math.atan2(F_ext[i + counter + 1], F_ext[i + counter])
            counter += 1
        else:
            F[i] = F_ext[i + counter]
            alpha[i] = np.pi / 2
    return F, F_ext, alpha

def projection(self, a, b):
    if np.linalg.norm(b) == 0:
        projection = 1
    else:
        projection = (np.dot(a, b) / (np.linalg.norm(b)))
    return projection
def lowpass(self, data, cutoff, fs):
    nyq = 0.5 * fs
    temp = cutoff / nyq
    b, a = butter(1, temp, btype="low", analog=False)
    filterd_signal = lfilter(b, a, data)
    return filterd_signal[-1]
def pseudoinverse_filtering(self,tau_cmd):
    Tr = np.zeros((2,2))
    Tds = np.zeros((2,2))
    F, F_ext, alpha = self.pseudoinverse_ext(tau_cmd)
    projection = np.zeros((2,1))
    for i in range(0,self.num_azi):
        a = np.zeros(2)
        b = np.zeros(2)
        a[0] = F_ext[2*i]
        a[1] = F_ext[i*2+1]
        b[0] = self.Tfs[i][0]
        b[1] = self.Tfs[i][1]
        projection[i]=self.projection(a,b)
        Tds[i][0] = F_ext[2*i]
        Tds[i][1] = F_ext[2*i+1]
    self.lp1 = np.append(self.lp1, np.transpose([Tds.flatten()]), axis=1)
    self.lp2 = np.append(self.lp2,projection,axis=1)
    if len(self.lp2[0])>=50/self.h:
        self.lp2 = np.delete(self.lp2,0,1)
        self.lp1 = np.delete(self.lp1, 0, 1)
    for i in range(0,self.num_azi):
        if len(self.lp1[0])>1:
            Tr[i][0] = self.lowpass(self.lp1[i*2].flatten(),1,100)
            Tr[i][1] = self.lowpass(self.lp1[i*2+1].flatten(), 1, 100)
    for i in range(0, self.num_azi):
        if len(self.lp2[0]) > 1:
            sign = np.sign(self.lowpass(self.lp2[i].flatten(),1,100))
```

```
            self.Tfs[i][0] = Tr[i][0]*sign
            self.Tfs[i][1] = Tr[i][1]*sign
    alpha_temp = np.zeros((3,1))
    for i in range(0,self.num_azi):
        if self.type[i] == 'azi':
            alpha_temp[i] =math.atan2(self.Tfs[i][1],self.Tfs[i][0])
    alpha_temp[2] = np.pi/2
    return alpha_temp
def thrustallocation(self,tau_cmd):
    alpha_f = self.pseudoinverse_filtering(tau_cmd)
    U,S,V = np.linalg.svd(self.B(alpha_f))
    S_delta = np.zeros((3,3))
    for i in range(0,len(S_delta)):
        if S[i]/S[0]>self.delta_high:
            S_delta[i][i] = 1/S[i]
        elif self.delta_low<=S[i]/S[0]<=self.delta_high:

            S_delta[i][i]=1/(S[1]*self.delta_high)
        elif S[i]/S[0]<self.delta_low:

            S_delta[i][i] = 0
    F = V.transpose()@S_delta@U.transpose()@tau_cmd
    F_temp = np.zeros((len(F),1))
    for i in range(0,len(F)):
        F_temp[i] = F[i]


    return F_temp, alpha_f
```

```
import matplotlib.pyplot as plt
import qpsolvers as solve_qp


class thrustallocation():
    def __init__(self):
        self.pos_x = np.array([-0.425, -0.425, 0.37])
        self.pos_y = np.array([0.07, -0.07, 0])
        self.f_max = np.array([3.2, 3.2, 1.17])
        self.f_min = np.array([-1.65, -1.65, -0.73])
        self.alpha_max =np.array([np.pi,np.pi])
        self.alpha_min = np.array([-np.pi,-np.pi])
        self.alpha_rate_max = np.array([np.pi / 2, np.pi / 2])
        self.alpha_rate_min = np.array([-np.pi / 2, -np.pi / 2])
        self.type = np.array(["azi", "azi", "tunnel"])
        self.num_thruster = 3
        self.num_azi = 2

        #tuning parameters
        self.h = .01
        self.P = .2
        self.omega = .9
        self.Q =  2
        self.alpha_0 = np.zeros(self.num_thruster)
        self.u_o = np.zeros(self.num_thruster)
    def B(self, angles):
        B = np.zeros((self.num_thruster, 3))

        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                B[0][i] = np.cos(angles[i])
                B[1][i] = np.sin(angles[i])
                B[2][i] = self.pos_x[i] * np.sin(angles[i]) - self.pos_y[i] *
                ↪  np.cos(angles[i])
            else:
                B[0][i] = 0
                B[1][i] = 1
                B[2][i] = self.pos_x[i]
        return B

    def B_derived(self, angles, u_0):
        B = np.zeros((3, self.num_azi))

        for i in range(0, self.num_azi):
            if self.type[i] == "azi":
                B[0][i] = -u_0[i] * np.sin(angles[i])
                B[1][i] = u_0[i] * np.cos(angles[i])
                B[2][i] = u_0[i] * (self.pos_x[i] * np.cos(angles[i]) + self.pos_y[i] *
                ↪  np.sin(angles[i]))
```

```
    return B
def singularity_term_Johansen(self,alpha_0):
    B = self.B(alpha_0)
    term = 0.1/(1+np.linalg.det(B*B.transpose()))

    return term
def thrustallocation(self, tau):

    self.alpha_0[2] = np.pi / 2
    W = self.P
    Q = self.Q * np.identity(self.num_thruster)
    omega = self.omega * np.identity(self.num_azi)
    step =self.h
    H = np.zeros((2 * self.num_thruster + self.num_azi, 2 * self.num_thruster +
    ↪  self.num_azi))
    h = np.zeros(2 * self.num_thruster + self.num_azi)

    for i in range(0, self.num_thruster):
        H[i, i] = W * (3 * self.u_o[i] ** 2 / (4 * np.abs(self.u_o[i]) ** (5 / 2) +
        ↪  0.00001))
        h[i] = W * (3 * self.u_o[i]) / (2 * np.sqrt(np.abs(self.u_o[i])) + 0.00001)
        H[i + self.num_thruster, i + self.num_thruster] = Q[i, i]
    for i in range(0, self.num_azi):
        H[i + 2 * self.num_thruster, i + 2 * self.num_thruster] = omega[i, i]

        term = self.singularity_term_Johansen(self.alpha_0)
        term1 = self.singularity_term_Johansen(self.alpha_0 + step)

        h[i + 2 * self.num_thruster] = (term1 - term) / step
    A = np.zeros((3, 2 * self.num_thruster + self.num_azi))
    B = self.B(self.alpha_0)
    b = tau - B @ self.u_o
    B_der = self.B_derived(self.alpha_0, self.u_o)


    for i in range(0, self.num_thruster):
        A[0][i] = B[0][i]
        A[1][i] = B[1][i]
        A[2][i] = B[2][i]
    for i in range(self.num_thruster, self.num_thruster * 2):
        A[i - self.num_thruster][i] = 1

    for i in range(0, self.num_azi):
        A[0][i + 2 * self.num_thruster] = B_der[0][i]
        A[1][i + 2 * self.num_thruster] = B_der[1][i]
        A[2][i + 2 * self.num_thruster] = B_der[2][i]

    G = np.zeros(2 * self.num_thruster + self.num_azi)
    g = np.zeros(1)
    for i in range(0, self.num_thruster):
        # saturation limits
        g = np.vstack([g, self.f_max[i] - self.u_o[i]])
        g = np.vstack([g, -self.f_min[i] + self.u_o[i]])
        if self.type[i] == "azi":
            holder = np.zeros((6, 2 * self.num_thruster + self.num_azi))

            holder[2][self.num_thruster * 2 + i] = 1
            holder[3][self.num_thruster * 2 + i] = -1
            holder[4][self.num_thruster * 2 + i] = 1
            holder[5][self.num_thruster * 2 + i] = -1

            g = np.vstack([g, self.alpha_max[i] - self.alpha_0[i]])
```

```
                g = np.vstack([g, -self.alpha_min[i] + self.alpha_0[i]])
                g = np.vstack([g, self.alpha_rate_max[i] * step])
                g = np.vstack([g, -self.alpha_rate_min[i] * step])

            else:
                holder = np.zeros((2, 2 * self.num_thruster + self.num_azi))
            holder[0][i] = 1
            holder[1][i] = -1
            G = np.vstack([G, holder])


        x = solve_qp.solve_qp(2*H, h, G, g, A, b, solver="osqp")

        if x is None:
            print("Feil")
        else:
            for i in range(0, self.num_thruster):
                self.u_o[i] += x[i]
            for i in range(0, self.num_azi):
                self.alpha_0[i] += x[self.num_thruster * 2 + i]

        F_temp = np.zeros((3, 1))
        alpha_temp = np.zeros((3, 1))
        for i in range(0, self.num_thruster):
            if self.type[i] == 'azi':
                F_temp[i] = self.u_o[i]
                alpha_temp[i] = self.alpha_0[i]
            else:
                F_temp[i] = self.u_o[i]
                alpha_temp[i] = np.pi / 2

        return F_temp, alpha_temp

    def singularity_term_Johansen(self, alpha_0):
        B = self.B(alpha_0)
        term = 0 / (1 + np.linalg.det(B * B.transpose()))
        return term
```

QP SCIBILLA

```
import time
import numpy as np
import matplotlib.pyplot as plt
import qpsolvers as solve_qp


class thrust_allocation():
    def __init__(self):
        self.pos_x = np.array([-0.425, -0.425, 0.37])
        self.pos_y = np.array([0.07, -0.07, 0])
        self.f_max = np.array([3.2, 3.2, 1.17])
        self.f_min = np.array([-1.65,-1.65,-0.73])
        self.alpha_max =np.array([np.pi,np.pi])
        self.alpha_min = np.array([-np.pi,-np.pi])
        self.alpha_rate_max = np.array([np.pi/2,np.pi/2])
        self.alpha_rate_min = np.array([-np.pi/2,-np.pi/2])
        self.type = np.array(["azi", "azi", "tunnel"])
        self.num_thruster = 3
        self.num_azi = 2

        #tuning parameters
        self.h = 0.01
        self.P =0.1
        self.omega = .9
        self.Q = 2
        self.P_pluss = 0.1
        self.omega_pluss =.4
        self.Q_pluss =.5


        self.alpha_0 = np.zeros(self.num_thruster)
        self.u_o = np.zeros(self.num_thruster)
        self.alpha_0_plus = np.zeros(self.num_thruster)
        self.u_o_plus = np.zeros(self.num_thruster)
        self.F_avg = np.zeros((1, 3))
        self.tau_0 = np.zeros(3)
    def B(self,angles):
        B = np.zeros((self.num_thruster,3))

        for i in range(0,self.num_thruster):
            if self.type[i] == "azi":
                B[0][i] = np.cos(angles[i])
                B[1][i] =np.sin(angles[i])
                B[2][i] =self.pos_x[i]*np.sin(angles[i])-self.pos_y[i]*np.cos(angles[i])
            else:
                B[0][i] = 0
                B[1][i] = 1
                B[2][i] = self.pos_x[i]
        return B
```

```
def B_derived(self,angles,u_0):
    B = np.zeros((3,self.num_azi))

    for i in range(0, self.num_azi):
        if self.type[i] == "azi":
            B[0][i] = -u_0[i]*np.sin(angles[i])
            B[1][i] = u_0[i]*np.cos(angles[i])
            B[2][i] = u_0[i]*(self.pos_x[i] * np.cos(angles[i]) + self.pos_y[i] *
            ↪  np.sin(angles[i]))

    return B

def Qp_scibilla(self, tau):
    m = (self.num_thruster + self.num_azi + 3)
    H = np.zeros((2 * m, 2 * m))
    h = np.zeros((2 * m))
    A = np.zeros((6, 2 * m))
    b = np.zeros((1, 1))
    step = self.h
    G = np.zeros((1, 2 * m))
    g = np.zeros((1, 1))

    P = self.P * np.identity(self.num_thruster)
    Q = self.omega* np.identity(self.num_azi)
    omega = self.omega * np.identity(3)
    P_forward = self.P_pluss * np.identity(self.num_thruster)
    Q_forward = self.Q_pluss * np.identity(self.num_azi)
    omega_forward = self.omega_pluss * np.identity(3)

    horizon = 5/self.h
    tau_plus = np.zeros(3)

    # setting up the cost matrix
    H[0:len(P), 0:len(P)] += P
    H[len(P):len(P) + len(Q), len(P):len(P) + len(Q)] += Q
    H[len(P) + len(Q):m, len(P) + len(Q):m] += omega
    H[m:len(P_forward) + m, m:len(P_forward) + m] += P_forward
    H[len(P_forward) + m:len(P_forward) + len(Q_forward) + m,
    len(P_forward) + m:len(P_forward) + len(Q_forward) + m] += Q_forward
    H[len(P_forward) + m + len(Q_forward):2 * m, len(P_forward) + len(Q_forward) +
    ↪  m:2 * m] += omega_forward

    # setting up the linear cost matrix
    h[0:self.num_thruster] += 2 * self.P * self.u_o

    # setting up tau+
    self.F_avg = np.r_[self.F_avg, [tau]]

    if len(self.F_avg) > horizon:
        self.F_avg = np.delete(self.F_avg, 0, axis=0)
    X, Y, psi = 0., 0., 0.
    for i in range(0, len(self.F_avg)):
        X += self.F_avg[i][0]
        Y += self.F_avg[i][1]
        psi += self.F_avg[i][2]
        if i == len(self.F_avg) - 1:
            tau_plus[0] = 1.3 * X / (i + 1)
            tau_plus[1] = 1.3 * Y / (i + 1)
            tau_plus[2] = 1.3 * psi / (i + 1)

    # constraints
    # Gx<=g
    for i in range(0, self.num_thruster):
```

```
    if self.type[i] == "azi":
        G_holder = np.zeros((12, 2 * m))
        g_holder = np.zeros((12, 1))
    else:
        G_holder = np.zeros((4, 2 * m))
        g_holder = np.zeros((4, 1))

    for x in range(0, int(len(G_holder) / 2)):

        # fmax
        if x == 0:
            G_holder[2 * x][i] = 1
            G_holder[2 * x + 1][i] = -1
            g_holder[2 * x] = self.f_max[i] - self.u_o[i]
            g_holder[2 * x + 1] = -self.f_min[i] + self.u_o[i]
        # fmax +
        elif x == 1:
            G_holder[2 * x][m + i] = 1
            G_holder[2 * x + 1][m + i] = -1
            g_holder[2 * x] = self.f_max[i] - self.u_o_plus[i]
            g_holder[2 * x + 1] = -self.f_min[i] + self.u_o_plus[i]
        # alpha min max
        elif x == 2:
            G_holder[2 * x][self.num_thruster + i] = 1
            G_holder[2 * x + 1][self.num_thruster + i] = -1
            g_holder[2 * x] = self.alpha_max[i] - self.alpha_0[i]
            g_holder[2 * x + 1] = -self.alpha_min[i] + self.alpha_0[i]
        # alpha rate min max
        elif x == 3:
            G_holder[2 * x][self.num_thruster + i] = 1
            G_holder[2 * x + 1][self.num_thruster + i] = -1
            g_holder[2 * x] = self.alpha_rate_max[i] * step
            g_holder[2 * x + 1] = -self.alpha_rate_min[i] * step
        # alpha+ min max
        elif x == 4:
            G_holder[2 * x][self.num_thruster + m + i] = 1
            G_holder[2 * x + 1][self.num_thruster + m + i] = -1
            g_holder[2 * x] = self.alpha_max[i] - self.alpha_0_plus[i]
            g_holder[2 * x + 1] = -self.alpha_min[i] + self.alpha_0_plus[i]
        # alpha+ rate min max
        elif x == 5:
            G_holder[2 * x][self.num_thruster + m + i] = 1
            G_holder[2 * x + 1][self.num_thruster + m + i] = -1
            g_holder[2 * x] = self.alpha_rate_max[i] * step
            g_holder[2 * x + 1] = -self.alpha_rate_min[i] * step

    G = np.append(G, G_holder, axis=0)
    g = np.append(g, g_holder, axis=0)

B = self.B(self.alpha_0)
b = np.append(b, tau - B @ self.u_o)
B_der = self.B_derived(self.alpha_0, self.u_o)

for i in range(0, self.num_thruster):
    A[0][i] = B[0][i]
    A[1][i] = B[1][i]
    A[2][i] = B[2][i]
for i in range(m - self.num_thruster, m):
    A[i - self.num_thruster - self.num_azi][i] = 1
for i in range(0, self.num_azi):
    A[0][i + self.num_thruster] = B_der[0][i]
    A[1][i + self.num_thruster] = B_der[1][i]
    A[2][i + self.num_thruster] = B_der[2][i]
```

```
B = self.B(self.alpha_0_plus)
b = np.append(b, tau_plus - B @ self.u_o_plus)
b = np.delete(b, 0)
B_der = self.B_derived(self.alpha_0_plus, self.u_o_plus)

for i in range(0, self.num_thruster):
    A[3][i + m] = B[0][i]
    A[4][i + m] = B[1][i]
    A[5][i + m] = B[2][i]
for i in range(m - self.num_thruster, m):
    A[3 + i - self.num_thruster - self.num_azi][i + m] = 1
for i in range(0, self.num_azi):
    A[3][i + self.num_thruster + m] = B_der[0][i]
    A[4][i + self.num_thruster + m] = B_der[1][i]
    A[5][i + self.num_thruster + m] = B_der[2][i]

x = solve_qp.solve_qp(2*H, h, G, g, A, b, solver="quadprog")

if x is None:
    print("feil")
    return 0, 0
else:
    for i in range(0, self.num_thruster):
        self.u_o[i] += x[i]
        self.u_o_plus[i] += x[i + m]
    for i in range(0, self.num_azi):
        self.alpha_0[i] += x[self.num_thruster + i]
        self.alpha_0_plus[i] += x[m + self.num_thruster + i]

F_temp = np.zeros((3, 1))
alpha_temp = np.zeros((3, 1))
for i in range(0, self.num_thruster):
    if self.type[i] == 'azi':
        F_temp[i] = self.u_o[i]
        alpha_temp[i] = self.alpha_0[i]
    else:
        F_temp[i] = self.u_o[i]
        alpha_temp[i] = np.pi / 2

return F_temp, alpha_temp
```

```
import time
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.linalg import null_space
class maneuvering:
    def __init__(self):
        import time
        #------- physical properties--------
        self.pos_x = np.array([-0.425, -0.425, 0.37])
        self.pos_y = np.array([0.07, -0.07, 0])
        self.f_max = np.array([3.2, 3.2, 1.17])
        self.type   = np.array(["azi", "azi", "tunnel"])
        self.num_thruster =3
        self.num_azi      =2
        self.h = 0.01
        #Tuning parameters
        self.weighting = np.array([1.2, 1.2, 1])
        self.gamma = 0.5
        self.rho = 1
        self.U = .6
        self.epsilon = 0.001
        self.mu = 1

        #arrays for holding values
        self.tau_0      = np.zeros(3)
        self.tau_derr   = np.zeros(3)
        self.alpha_0    = np.zeros(self.num_thruster)
        self.theta      = np.zeros(self.num_azi)
        self.xi         = np.zeros(self.num_thruster + self.num_azi)
    def B(self,angles):
        B = np.zeros((self.num_thruster,3))

        for i in range(0,self.num_thruster):
            if self.type[i] == "azi":
                B[0][i] = np.cos(angles[i])
                B[1][i] =np.sin(angles[i])
                B[2][i] =self.pos_x[i]*np.sin(angles[i])-self.pos_y[i]*np.cos(angles[i])
            else:
                B[0][i] = 0
                B[1][i] = 1
                B[2][i] = self.pos_x[i]
        return B
    def B_ext(self):
        B = np.empty((3, 1))
        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                array = np.empty((3,2))
                array[0][0] = 1
```

```
                array[1][0] = 0
                array[2][0] = -self.pos_y[i]

                array[0][1] = 0
                array[1][1] = 1
                array[2][1] = self.pos_x[i]
                B = np.append(B,array,axis=1)
            else:
                array = np.empty((3, 1))
                array[0][0] = 0
                array[1][0] = 1
                array[2][0] = self.pos_x[i]
                B = np.append(B, array, axis=1)
        B = np.delete(B,0,1)
        return B
    def weighted_pseudoinverse(self, weighting):
        B = self.B_ext()
        p = self.num_thruster + self.num_azi
        W = np.zeros(1)
        for i in range(0, len(weighting)):
            if self.type[i] == "azi":
                W = np.append(W, weighting[i])
                W = np.append(W, weighting[i])
            else:
                W = np.append(W, weighting[i])
        W = np.delete(W, 0)
        W = np.linalg.inv(np.diag(W))

        return W @ B.transpose() @ np.linalg.inv(B @ W @ B.transpose())
    def J_theta(self, theta, w, Q):
        J_holder = 0
        counter = 0
        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                Q_temp = np.array([Q[counter], Q[counter + 1]])
                J_holder += w[i] * Q_temp.transpose() @ Q_temp
                counter += 2
            else:
                J_holder += w[i] * Q[counter].transpose() @ Q[counter]
                counter += 1

            return theta.transpose() @ J_holder
    def J_theta_azi_rate(self, w, xi_d, Q, alpha_0, epsilon):
        J_holder = 0
        counter = 0
        for i in range(0, self.num_thruster):
            if self.type[i] == "azi":
                a_0 = np.array([np.cos(alpha_0[i]), np.sin(alpha_0[i])])
                Q_temp = np.array([Q[counter], Q[counter + 1]])
                length = np.sqrt(xi_d[counter] ** 2 + xi_d[counter + 1] ** 2)
                xi_d_temp = np.array([xi_d[counter], xi_d[counter + 1]])
                J_holder += (w[i] * (xi_d_temp.transpose()) / (length + epsilon) -
                ↪  a_0.transpose()) @ Q_temp
                counter += 2
            else:
                J_holder += w[i] * (xi_d[counter]) / (np.abs(xi_d[counter]) + epsilon) *
                ↪  Q[counter]
                counter += 1

        return J_holder
    def V_theta(self, w, xi_diff, Q):
        V_theta = 0
        counter = 0
```

```
    for i in range(0, self.num_thruster):

        if self.type[i] == "azi":
            Q_temp = np.array([Q[counter], Q[counter + 1]])
            xi_temp = np.array([xi_diff[counter], xi_diff[counter + 1]])
            V_theta += -w[i] * xi_temp.transpose() @ Q_temp
            counter += 2
        else:
            V_theta += -w[i] * xi_diff[i] * Q[i]
            counter += 1
    return V_theta
def xi_d(self, xi_p, Q, theta):
    xi_d = np.zeros(self.num_thruster + self.num_azi)
    counter = 0
    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            xi_temp = np.array([xi_p[counter], xi_p[counter + 1]])
            Q_temp = np.array([Q[counter], Q[counter + 1]])
            temp = xi_temp + Q_temp @ theta
            xi_d[counter] = temp[0]
            xi_d[counter + 1] = temp[1]
            counter += 2
        else:
            xi_d[counter] = xi_p[counter] + Q[counter] @ theta
            counter += 1
    return xi_d
def tau_derived(self, tau, r):
    self.tau_derr[0] = (1 - r) * self.tau_derr[0] + r * (tau[0] - self.tau_0[0])
    self.tau_derr[1] = (1 - r) * self.tau_derr[1] + r * (tau[1] - self.tau_0[1])
    self.tau_derr[2] = (1 - r) * self.tau_derr[2] + r * (tau[2] - self.tau_0[2])
    self.tau_0 = tau
    return self.tau_derr
def thrustallocation(self, tau):
    r = 0.7
    tau_dot = self.tau_derived(tau, r)
    B = self.weighted_pseudoinverse(self.weighting)
    Q = null_space(self.B_ext())
    xi_p = B@tau

    xi_p_dot = B@tau_dot
    theta = self.theta
    xi_d = self.xi_d(xi_p, Q, theta)

    xi_diff = self.xi - xi_d
    #J_theta = self.J_theta(theta, self.weighting, Q)
    J_theta = self.J_theta_azi_rate(self.weighting, xi_d, Q, self.alpha_0,
    ↪  self.epsilon)

    V_theta = self.V_theta(self.weighting, xi_diff, Q)
    nu = -self.gamma * J_theta.transpose()

    a = np.zeros(self.num_thruster)
    b = np.zeros(self.num_thruster + self.num_azi)

    counter = 0
    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            xi_temp = np.array([self.xi[counter], self.xi[counter + 1]])
            a[i] = self.rho * (xi_temp.transpose() @ xi_temp - self.f_max[i] ** 2)
            b[counter] = 2 * xi_temp[0]
            b[counter + 1] = 2 * xi_temp[1]
            counter += 2
```

```
        else:
            a[i] = self.rho * (self.xi[counter] * self.xi[counter] - self.f_max[i]
            ↪  ** 2)
            b[counter] = 2 * self.xi[counter]
            counter += 1

    counter = 0
    k = np.zeros(self.num_thruster + self.num_azi)
    k[0] = 1

    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            xi_temp = np.array([xi_diff[counter], xi_diff[counter + 1]])
            xi_temp_dot = np.array([xi_p_dot[counter], xi_p_dot[counter + 1]])
            xi_temp_len = np.sqrt(xi_temp[0] ** 2 + xi_temp[1] ** 2) + self.epsilon
            Q_temp = np.array([Q[counter], Q[counter + 1]])

            holder = -self.U * (xi_temp / xi_temp_len) + xi_temp_dot + Q_temp @ nu
            k[counter] = holder[0]
            k[counter + 1] = holder[1]
            counter += 2
        else:
            xi_temp_len = np.sqrt(xi_diff[counter] ** 2)
            k[counter] = -self.U * (xi_diff[counter] / (xi_temp_len + self.epsilon))
            ↪  + xi_p_dot[counter] + Q[counter] @ nu
            counter += 1

    counter = 0
    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            b_temp = np.array([b[counter], b[counter + 1]])
            k_temp = np.array([k[counter], k[counter + 1]])
            if a[i] + b_temp.transpose() @ k_temp <= 0:
                self.xi[counter] += k[counter] * self.h
                self.xi[counter + 1] += k[counter + 1] * self.h
            else:
                temp = (a[i] + b_temp.transpose() @ k_temp) / (b_temp.transpose() @
                ↪  b_temp) * b_temp
                self.xi[counter] += (k[counter] - temp[0]) * self.h
                self.xi[counter + 1] += (k[counter + 1] - temp[1]) * self.h
            counter += 2
        else:
            if float(a[i] + b[counter] * k[counter]) <= 0:
                self.xi[counter] += k[counter] * self.h
            else:
                temp = (a[i] + b[counter] * k[counter]) / (b[counter] * b[counter])
                ↪  * b[counter]
                self.xi[counter] += (k[counter] - temp) * self.h
            counter += 1
    self.theta += (nu - self.mu * V_theta.transpose()) * self.h

    counter = 0
    F = np.zeros((self.num_thruster, 1))

    for i in range(0, self.num_thruster):
        if self.type[i] == "azi":
            F[i] = np.sqrt(self.xi[counter] ** 2 + self.xi[counter + 1] ** 2)
            self.alpha_0[i] = math.atan2(self.xi[counter + 1], self.xi[counter])
            counter += 2
        else:
            F[i] = self.xi[counter]
            counter += 1
    temp = np.zeros((self.num_thruster, 1))
```

```
temp[0] = self.alpha_0[0]
temp[1] = self.alpha_0[1]
return F, temp
```

B - PARTS

this is section is directly taken from Elvenes et al. (2023)

Acquisition and assembling

Equipment

The parts were chosen with Robert Opland's help and were selected based on quality and a desire to standardize drivers in the MC-Lab. Hence, the CSAD components were chosen as a reference, especially the controller boards, such as the ESC, servo controller, and IMU. The following is a description of all the parts chosen.

Hull

As required, Jonny Harbor Tugboat model from Aeronaut was provided (Aeronaut (2022)). The kit includes numerous functional and moving elements, including a towing winch, a headlight, and a fire monitor. These functionalities have been ignored in the building of CSV as they are not relevant for the intended use. The hull is a 1:32 scale model of a tug boat with hull parameters described below.

Hull parameters.

Height	675 [mm]
Width	308 [mm]
length	990 [mm]
draft	110 [mm]

The model is intended to be used with shaft propellers and rudders but includes a mounting plate for conversion to schottle drive. This conversion kit, however, is not compatible with our selection of motors and azimuth thrusters but is used as a guide to making a mounting plate for the thrusters later.

Aziumth thrusters

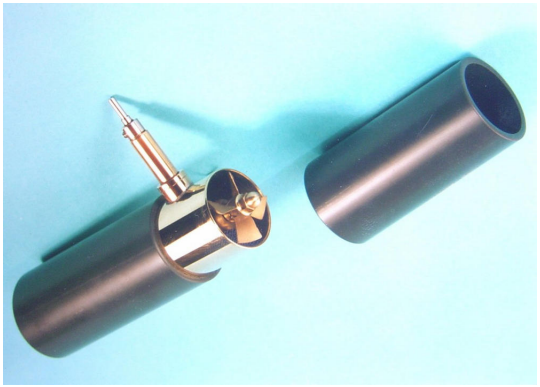
The azimuth thrusters chosen are the "CEM Schottle Drive Unit 70mm" from "Cornwall model boat" (Boats (2022)) illustrated below.

Schottle Drive. Courtesy: Boats (2022).

This thruster has an angle range of 180°. The azimuth thruster will be converted to continuous 360° capabilities, and also a more direct drive coupling with the motor before installation as discussed.

Bow thruster

For the bow thruster, the "Bow thruster 28/32 x 180mm" from Bauer-Modelle was selected (Bauer-Modelle (2022)). This was selected for its size and quality since it is in all brass. It also provides the option to change and choose the desired motor. The figure below illustrates the bow thruster.



Bow thruster. Courtesy: Bauer-Modelle (2022).

DC motors

For the DC motors, two types of brushless motors were selected, one type for the azimuth thrusters and one for the bow thruster. For the azimuth thrusters, the " OS OMA-5010-810kv" was selected for its "lower speed" and higher torque configuration illustrated below (Elefun (2022*d*)).

Aziumth motor. Courtesy: Elefun (2022*d*).

For the bow thruster, the " Dualsky ECO 3520C V2 820KV 210gram" (Elefun (2022*b*)) was selected. Compared to the OS motor it has a higher speed and lower torque since this is more suitable for the smaller higher speed propeller of the bow thruster. The motor for the bow thruster is illustrated below.

Dualsky ECO 3520C V2 820KV, Courtesy: Elefun (2022*b*).

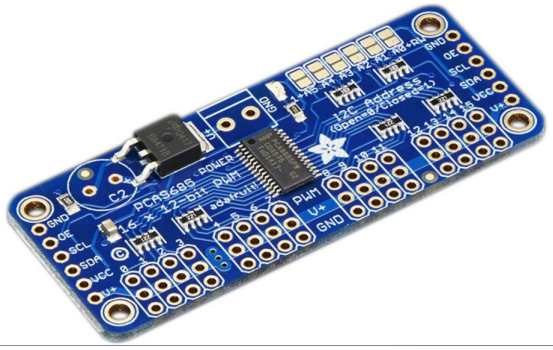
ESC

Since the power required to drive the motors is higher than what the Raspberry Pi can provide, ESCs are needed. This amplifies the Raspberry Pi's PWM (Pulse Width Modulation) signal. The "OS OCA-150" is the chosen ESC since it is used in CSAD and compatible with the motors. The figure below illustrates the chosen ESCs.

Electronic Speed Controller. Courtesy: Elefun (2022*c*).

PWM breakout board

Since the Raspberry Pi only has two PWM pins, a PWM breakout board is needed. The board chosen is the "815 - PCA9685 16-kanalers 12-biters PWM-/servodriver" from Adafruit (Distrelec (2022)) the PWM boards take an Inter-Integrated Circuit (I2C) connection from the Raspberry Pi and enables an output of up to 16 PWM signals. This is the same board as in CSAD. the figure below illustrates the chosen PWM breakout board.



PWM breakout board. Courtesy: Distrelec (2022).

Driveshaft coupling

Driveshaft couplings were needed to connect the motors with the azimuths and bow thrusters. A driveshaft coupling connects the two shafts while having the ability to flex to reduce vibrations and account for small misalignments of the two shafts. The figure below illustrates the chosen driveshafts.

Driveshaft coupling. Courtesy: RS-Components (2022*a*).

Servos

For the servos, the Mx series from Robotis was selected. These are high-quality and precision servos. The MX-28R (Robotis (2022*a*)) was selected. These servos have a precision of 0.088° with a torque capacity of $3.1[\frac{N}{m}]$, and the ability to be daisy chained. The MX-28 is smaller than the MX-106 found in CSAD, but these servos are sufficient for the intended application. However, the drivers are the same for both models. The figure below shows the selected servo.

MX-28. Courtesy:Robotis (2022*a*).

U2D2

The U2D2 (Robotis (2022*b*)) board is required to control the servos. This board connects to the Raspberry Pi through USB and allows daisy chaining of the servos. However, it cannot provide the necessary power, so the servos need to be connected to an external 12V power source. This board is also the same as the one in CSAD so the drivers will be similar. The figure below shows the U2D2.

U2D2. Courtesy:Robotis (2022*b*).

IMU

The IMU "Gravity: BMI160"Digi-Key (2022) is selected. This is a 3.6V 6-axis inertial motion sensor that is small and requires little energy ($< 1[mA]$). It has programmable ranges and frequencies and runs on an I2C bus enabling daisy chaining. The figure below shows the chosen IMU.

IMU. Courtesy:Digi-Key (2022).

I2C breakout board

Connecting all the necessary IMUs directly to the Raspberry Pi is possible. However, to decrease the number of cables and increase plug-and-play capabilities, daisy chaining of the IMUs is desired. One I2C board for each IMU was ordered to enable this, providing the ability for daisy chaining. The breakout board chosen was the "Gravity: I2C HUB"(DFROBOT (2022)) as illustrated below.

I2C breakout board. Courtesy DFROBOT (2022).

DC-to-DC converter

Since the Raspberry Pi requires 5[V] 3[A] a DC-to-DC converter is required. Previously in the other model, a car adapter was used. However, this does not work as optimal. Therefore, the "Mean Well DC-DC Converter"(RS-Components (2022*b*)) was selected. This converter has an input range of 9.2-18[V], and a variable output of 4.75-5.5V[V], with a current of 3[A]. The selected converter is illustrated below

DC-DC converter. Courtesy RS-Components (2022*b*).

Battery

The battery selected is the " 4s 8000mAh -100C - Gens Ace EC5 Bashing Series" (Elefun (2022*a*)). This battery has a voltage of 14.8, 8000[mAh] and a weight of 737[g]. 2 of these is planned to be connected in parallel, providing a total of 16000 [mAh]. This should be sufficient to power the model for a long time. In total 4 batteries were ordered of the model illustrated in the figure below. Enabling quick exchange for charged batteries, and reducing the downtime for the model.

Battery. Courtesy Elefun (2022*a*).

Raspberry Pi

The model is controlled through a Raspberry Pi 4 model B (Pi (2022)). with specifications:

- **Processor:** Quad-core Cortex-A72 @ 1.5 GHz.
- **RAM:** 8 GB LPDDR4-3200 SDRAM
- **Power:** 5V via USB-C

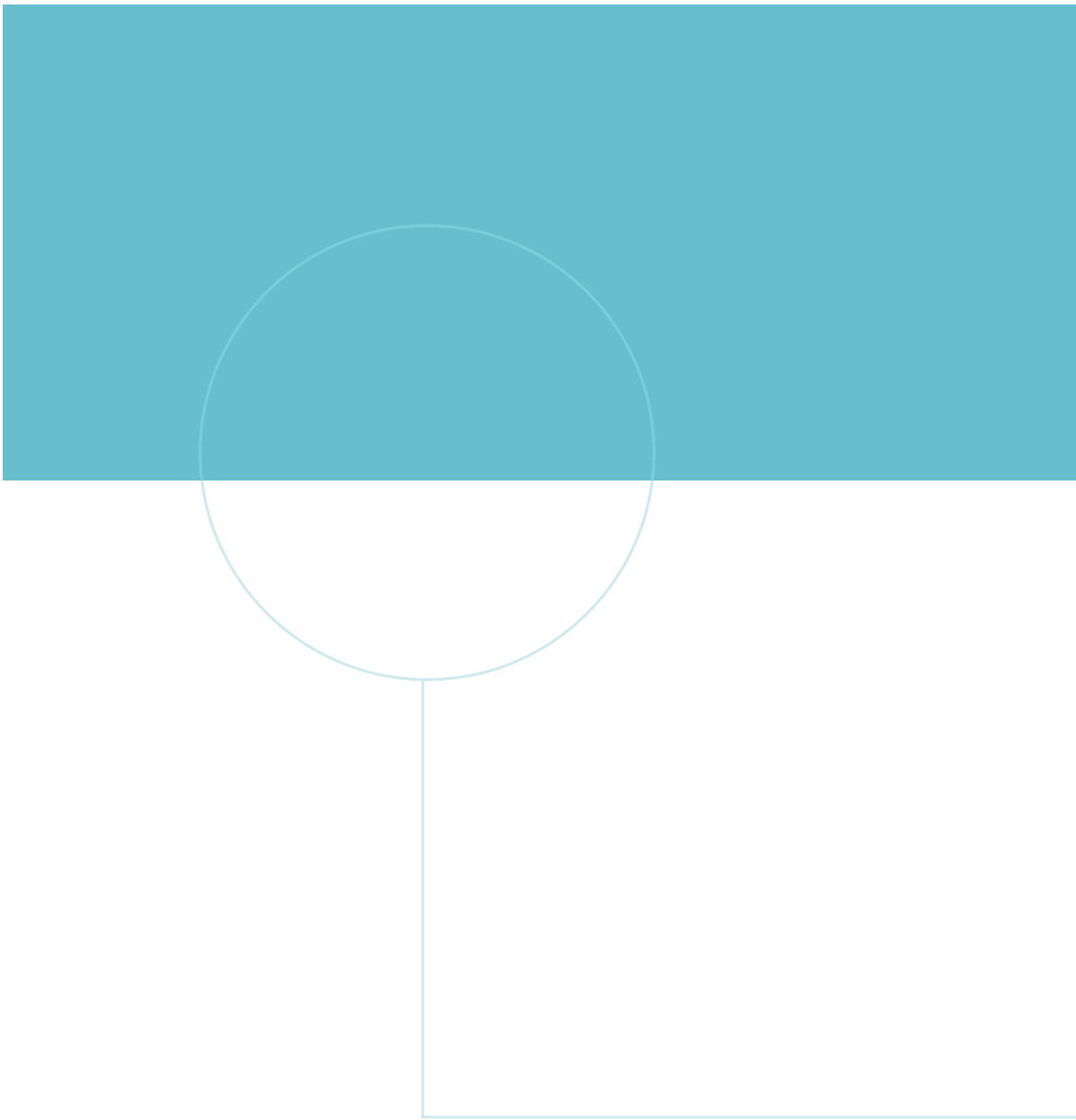
The Raspberry Pi enables connections through, GPIO, Bluetooth, USB and WiFi. This enables each component to be controlled from the Raspberry Pi. The Bluetooth connection is used for connecting a DualShock controller, and WiFi for remote SSH and transfer of data. the figure below isslustrates the Rapsberry PI.

Raspberry Pi courtesy. Pi (2022).

C - ATTACHMENTS

Included in the rapport is an attachment with the structure:

- 3D files
  - STL files (for all 3D modeled components)
  - Freecad files (for all 3D modeled components)
- Code
  - Force to PWM
    - The F2PWM code
  - Forcemapping
    - The code to calculate the force measurements from .mat files
  - Thrustallocations
    - Pseudo filtering
    - QP Scibilla
    - Qp Johansen
    - manuvering
    - code for the calculation and plot for the real time check
-
  - Thrust mapping starboard: all the .bin files for the measurements of the starboard thrust mapping
  - thrust map bow: all the .bin files for the measurements of the bow thruster
  - circle test: all the .bin files for the test of real time implementation
- video
  - video of basin trial in body frame
  - video of basin trial in basin frame
  - video of max speed of the vessel
- Pi: containing a copy of the code on the pi



NTNU

Norwegian University of
Science and Technology