

Mathias Netland Solheim

Integration between lidar- and camera-based situational awareness and control barrier functions for an autonomous surface vessel

Master's thesis in Marine Technology

Supervisor: Roger Skjetne

Co-supervisor: Mathias Marley

July 2022

Mathias Netland Solheim

Integration between lidar- and camera-based situational awareness and control barrier functions for an autonomous surface vessel

Master's thesis in Marine Technology
Supervisor: Roger Skjetne
Co-supervisor: Mathias Marley
July 2022

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



MASTER OF TECHNOLOGY THESIS DEFINITION (30 SP)

Name of the candidate:	Mathias Netland Solheim
Field of study:	Marine cybernetics
Thesis title (Norwegian):	Integrasjon mellom lidar- og kamerabasert situasjonsforståelse og barrierefunksjoner for en autonom overflatefarkost
Thesis title (English):	Integration between lidar- and camera-based situational awareness and control barrier functions for an autonomous surface vehicle

Background

Autonomous vessels require design of embedded safety functions in all control layers, to achieve safe and predictable vessel motions and anti-collision maneuvers in a possibly crowded maritime environment. The autonomous ship control system must build on functions such as mission planning, situation awareness (SA), autonomous guidance with nominal obstacle/collision avoidance (e.g., using control barrier functions), online risk assessment and risk-based control mitigation, mode control incl. minimum risk condition (MRC) mode assignments, and robust and stable maneuvering control. In this project we are considering especially the integration of a situation awareness function with lidar- and camera-based obstacle detection and target classification, with how to define/quantify Control Barrier Functions (CBFs) for safe motion control. The overall objective is the implementation and testing of a safe maneuvering control system in a partly unknown environment for a cybership (C/S) model in the Marine Cybernetics Lab (MC-Lab).

Recently, we have upgraded the control system for C/S Enterprise I (CSE1) and C/S Arctic Drillship (CSAD) with Raspberry Pi 4 (RP4) embedded control computers, running uBuntu with the Robot Operating System (ROS) for integration of sensors and thruster motor controllers with the control software. We want to further develop this system as part of this project, with the same upgrades also for the surface vehicle C/S Saucer (CSS), and then use this drone as test subject for the development.

Scope of Work

1. Perform a background and literature review to provide information and relevant references on:
 - Autonomous surface vessel and situation awareness functions.
 - Sensor fundamentals and modeling of camera and lidar.
 - Machine vision fundamentals, particularly camera- and lidar-based sensor fusion, target tracking algorithms, etc.
 - Object detection and classification based on neural network methods such as CNNs.
 - Relevant theory concerning CBFs for collision avoidance.

Write a list with abbreviations and definitions of terms and symbols, relevant to the literature study and project report.

2. Develop RP4 and ROS-based vessel maneuvering control system for CSS, using the same platform as for CSE1 and CSAD:
 - The vessel should have modules implemented in ROS for Mission Manager, Situation Awareness, Guidance, Observer, Controller, and Thrust Allocation.
 - Since CSS is omnidirectional with almost negligible resistance in yaw, one should consider heading control and position control individually by decoupling yaw control from surge-sway control. This needs particular focus on how to do the thruster configuration and thrust allocation in order to tightly stabilize the heading. Develop correspondingly an optimal thrust allocation algorithm for the CSS. Test several thrust configurations and discuss the results.
 - Perform testing in MC-Lab to tune the observer, guidance, and controller modules in order to get the maneuvering-based control system to perform well.



3. Develop a SA system based on camera and lidar for the CSS. This includes sensor fusion, an object detector and tracking algorithms. A CNN should be considered for detection and classification of relevant objects in MC-lab. Test the system and discuss the module's ability to detect and track targets.
4. Let detected objects classified as relevant targets be associated with a Control Barrier Function (CBF), that is, each target quantifies a new CBF when detected, where the CBF enters the control loop for collision avoidance. Develop new CBF-based maneuvering control modules that ensures safe maneuvering with anti-collision. Consider the logic for how a CBF emerge and later vanishes. Present and discuss how the CBF-based SA, Guidance, and Controller modules interact to ensure safe maneuvering; that is, present this by a detailed flowchart, timing diagram, logics, etc.
5. Test the overall system in MC-Lab for a set of specified test scenarios. Present and discuss/critique the overall functionality and resulting performance.

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 own contributions), technical background and literature review, problem formulation or research questions, 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. 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, 2022

Due date: As specified by the administration.

Supervisor: Roger Skjetne

Co-advisor(s): Mathias Marley

Signatures:

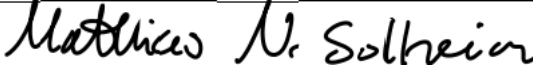
Digitally signed by Roger Skjetne
Date: 2022.03.02 16:31:40 +01'00'

Preface

This thesis marks the conclusion of my master's degree in marine technology at the Norwegian University of Science and Technology (NTNU). It is the results of a five-year-long study that provided me with a significant amount of knowledge about marine technology and especially marine cybernetics. The subject of study is integrating situational awareness in autonomous surface vessels to guarantee safe automatic control.

The thesis work was conducted over the spring semester of 2022. First, the necessary knowledge information was attained. This had been largely done as part of the project thesis [Solheim, 2021], but the decision to incorporate CBFs into the thesis required more studies. Resources for this were provided by my supervisor, Roger Skjetne, and co-supervisor, Mathias Marley, in the form of technical notes and relevant published literature. Next, the focus was on developing the appropriate control system modules for the CS Saucer using Python and ROS. The implementations of the guidance module and backstepping controllers specifically were based on the technical notes of my supervisors as well as the Matlab implementations of previous master students Moen, 2021 and Åsheim, 2021. Finally, I started work on the SA system parallel to implementing the control system. Most sensor drivers and calibration software, along with the implementations of CNN utilized in this thesis, are open-source ROS packages freely available online and are not the author's own work. However, some modification of the CNN, fusion method and calibration code has been a necessity.

Senior Engineer Torgeir Wahl procured the relevant hardware. Senior Engineer Robert Opland installed the Qualisys tracking markers onto the top module. Both Torgeir and Robert were also helpful in replacing wiring and circuitry onboard the vessel. I installed the remaining hardware myself.



Mathias Netland Solheim
July 2022, Trondheim

Abstract

The development of fully autonomous surface vessels (ASVs) has seen significant progress in recent years and is at the cusp of becoming an actuality. One of the most critical aspects in making ASVs a success is safety assurance. In order to operate safely, the vessel must be able to perceive, understand and adapt to the environment around it. This is known as situational awareness (SA). Typically, SA is achieved by combining the measurements from several sensors. For an ASV, electro-optical sensors such as camera and lidar are good candidates to enable environmental perception.

This thesis considers the implementation, and subsequent integration, of a lidar- and camera-based SA-system with control barrier functions (CBF) for safe motion control, using the CS Saucer (CSS) as the experimental platform. The CSS is a model-scale vessel operating in the marine cybernetics laboratory at the NTNU. It was chosen for the project due to its modularity in payload configurations, making sensor integration easy. Additionally, the vessel required a control system upgrade. This included replacing hardware and implementing a new control software architecture based on Python and ROS.

The SA system incorporates the convolutional neural network(CNN) SSDMobilescan to achieve visual detections of ships around the vessel. Then a projection model for the camera and lidar was derived, and the sensors were geometrically calibrated to obtain a rigid body transformation between the coordinate frames. Lidar points are projected onto any visual detections and added to a list of valid points if located within the detected bounding box. This list is then parsed to the clustering algorithm k-means, which computes a center point for each cluster present. This center point corresponds to the obstacle position and is converted to the NED-frame using the derived rigid body transformations.

Secondly, a maneuvering control system was integrated with the SA system. The system employs CBFs in the guidance layer to ensure collision-free path following. The controller utilized followed a cascade backstepping design that decoupled heading control from positional control. If the SA system detects any environmental obstacle, the guidance module will generate safe reference values if the current path is deemed unsafe.

All methods were tested through simulations and physical experiments in the MC lab. The SA system proved effective, detecting and accurately estimating the position of obstacles within a certain margin of error. It was, however, sensitive to false point projections due to subpar calibrations. The CNN was also found to have a limited detection range of 3m due to environment and model training. The motion control system as a whole performed satisfactorily, generating safe path signals in the presence of up to two obstacles and following the path to an acceptable degree.

Sammendrag

Utviklingen av autonome overlatefarkoster har de siste årene sett betydelig fremgang, og ideen om et hel-autonomt skip er i ferd med å bli en virkelighet. En av de mest kritiske aspektene for å gjøre autonome skip suksessfulle, ligger i evnen til å forsikre sikkerhet. For å trygt kunne operere, må et autonomt skip kunne oppfatte, tolke og tilpasse seg omgivelsene sine til en hver tid. Dette defineres som et fartøys situasjonsforståelse (SF). SF i fremkomstmidler oppnås hovedsakelig gjennom kombinasjonen av forskjellige sensorer. For autonome skip, vil elektro-optiske sensorer som kamera og lidar være gode kandidater.

Denne avhandlingen tar for seg implementasjonen, og, i ettetid, integrasjonen av et lidar- og kamera-basert SF system sammen med kontrollbarriere funksjoner (CBFer) for å oppnå trygg bevegelseskontroll av det autonome fartøyet CS Saucer (CSS). CSS er et modellskala overflate fartøy som operer i Marine Kybernetikk laboratoriet (MC lab) ved NTNU. Den ble valgt som eksperimentell platform grunnet fartøyet modularitet i nyttelast, som gjør integrasjon av nye sensorer enkelt. CSS hadde også behov for en oppgradering av kontrollsystemet sitt. Dette omfattet erstatting av gammel maskinvare og implementasjon av en programvare arkitektur basert på Python of Robot Operating System.

SF systemet inkorporerer det konvolusjonelt nevralt nettverket SSDMobilsan for å visuelt gjenkjenne skip i fartøyets nærhet. En projeksjonsmodell for lidaren og kameraet er så blitt avledet, og sensorene geometrisk kalibrert for å oppnå en transformasjon mellom koordinat-systemene. Punktene fra lidaren blir så projisert over på det visuelle gjenkjenningen, og lagt til i en liste med gyldige punkter dersom de er innenfor gjenkjenningens avgrensingsboks. Denne listen er så sent videre til grupperings algoritmen k-means som identifiserer et midtpunkt for hver gruppe i listen. Dette midtpunktet tilsvarer posisjonen til hindringen, og er transformert til NED-rammen.

Dette systemet er så integrert med et manøvrerings kontrollsystem. Systemet tar i bruk CBFer i veilednings modulen for å garantere kollisjonsfrie baner for skipet. Kontrolleren følger et cascade-backstepping design, og frakobler posisjons-kontroll fra gir kontroll. Dersom SF-systemet gjenkjenner hindringer i omgivelsene, vil veiledningsmodulen generere nye trygge baner om CBFene fastslår at den nåværende banen er utrygg.

Alle metoder har blitt testet gjennom simulasjoner og fysiske eksperimenter i MC labben. SF systemet virket til å være effektivt og klarte å gjenkjenne hindringer i omgivelsene, samt nøyaktig gjengi posisjonen deres, innenfor en feilmargin. Systemet var dessverre sensitivt ovenfor feilprojeksjoner som et resultat av dårlig kalibrering. Gjenkjenningsavstanden til Mobilescan var også begrenset til 3 meter, grunnet lav oppløsning i bilder og trening av nettverket. Manøvreringssystemet som en helhet derimot, fungerte tilfredstillende og genererte trygge baner for skipet med opp til to hindringer tilstede. Fartøyet klarte også å følge banene til en akseptabel grad.

Acknowledgements

First, I would like to thank my supervisor, Roger Skjetne, for the good guidance and suggestions when I expressed a wish to incorporate computer vision in my thesis. He has given me much trust and independence in pursuing my own interests, while always providing help when needed. The same goes for my co-supervisor, Mathias Marley.

I would also like to thank Senior Engineers at IMT, Torgeir Wahl and Robert Opland for providing excellent support in procuring the necessary hardware, installing and replacing electronics and understanding the Qualisys track system in the MC Lab. Without their help, I would never have gotten the vessel moving.

Most of the work on this thesis has been performed in office C1.058 at MTS, where I have shared the space with 5 other people. Therefore, I would like extend a thank you to my office colleagues for all the fruitful discussions, daily trivia quizzes, trips to Valentinlyst and countless matches of billiards. They made writing this thesis a lot more enjoyable.

Contents

Contents	ii
List of Figures	ix
List of Tables	xi
Nomenclature	xiii
List of Abbreviations	xiii
List of Symbols	xiv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives and Scope of Work	2
1.3 Contributions	3
1.4 Outline	3
2 Background and literature review	5
2.1 Situational Awareness in autonomous surface vessels	5
2.2 LiDAR	6
2.3 Camera	7
2.3.1 Deriving the Pinhole model	7
2.3.2 Distortion	9
2.3.3 The complete camera model	11
2.4 Supervised Learning for Machine Vision	11
2.4.1 Convolutional Neural Network	11

2.4.2	Single stage detectors vs. Dual Stage Detectors	13
2.5	Path generation for maneuvering	13
2.5.1	One-variable path parametrization	14
2.5.2	Two-variable path parametrization	14
2.6	Control Barrier Functions	15
3	Experimental Platform: CS Saucer	17
3.1	Background and motivation	17
3.2	Technical specification	18
3.3	Software	20
3.3.1	Robot Operating System	20
3.3.2	ROS architecture	21
3.4	The Marine Cybernetics Laboratory	22
3.5	Vessel model	23
3.5.1	Reference frames	23
3.5.2	Model	24
4	Problem formulation	27
4.1	Sensors	27
4.2	Situational awareness	27
4.3	Guidance problem	28
4.4	Control problem	28
4.4.1	The maneuvering problem	28
4.4.2	Stabilizing heading	29
4.5	Limitations and assumptions	29
5	Situational awareness	31
5.1	Sensor integration	31
5.2	Calibration	32
5.2.1	Camera calibration	32
5.2.2	Lidar-Camera Calibration	33
5.3	Transformation to NED-frame	35
5.4	Visual detection	36

5.4.1	Implementation and training	37
5.5	Lidar segmentation	37
5.5.1	K-Means	38
5.6	Sensor fusion	39
6	State estimation	43
6.1	Observer design	43
7	Guidance system	45
7.1	Path parametrization	45
7.2	Speed assignment	46
7.3	Control Barrier Function	46
7.4	Heading reference	48
7.5	Mission management	48
8	Control System	49
8.1	Cascade backstepping	49
8.1.1	Step 2: Kinetic design	49
8.1.2	Step 1: Kinematic design	50
8.2	Force limitations	52
8.3	Thrust allocation	53
8.3.1	Fixed thruster allocation	54
8.3.2	Thruster configurations	54
9	Results and Discussion	57
9.1	Testing scenarios	57
9.1.1	Situational Awareness	57
9.1.2	Collision avoidance	57
9.2	Situational awareness	58
9.2.1	Display and performance	58
9.2.2	Range test	59
9.2.3	Basin test	61
9.3	Complete system tests	63

9.3.1	Scenario 1: Single stationary obstacle	63
9.3.2	Scenario 2: Multiple stationary obstacles	65
9.4	Final discussion	68
10	Conclusion	71
10.1	Further work	72
	References	75
A	Getting started with ROS	I
A.1	Raspberry Pi Image	I
A.2	Communicating with the Raspberry Pi	II
A.3	How to ROS: A step by step guide	II
A.3.1	Sourcing ROS	II
A.3.2	Creating a workspace	II
A.3.3	The src-directory	III
A.3.4	Running ROS nodes	III
A.3.5	Launch files	IV
A.3.6	Topics	V
A.3.7	Storing data	V
A.3.8	Other usefull ROS-commands	V
B	Control system manual	VII
B.1	System requirements	VII
B.2	Running the control system nodes	VIII
B.2.1	Dualshock 4 driver	VIII
B.2.2	Camera	VIII
B.2.3	Lidar	VIII
B.2.4	Arduino	VIII
B.2.5	Motion Control System	IX
B.2.6	Object detection	IX
B.3	Dynamic Reconfigure	IX
C	Calibration Procedures	XI

C.1 Camera calibration	XI
C.2 Camera-Lidar calibration	XII
D Heading priority allocation	XV
E DS4 Controller Mapping	XVII

List of Figures

2.1	Architecture of an ASV [Smogeli, 2021]	6
2.2	Lidar coordinate system	7
2.3	Pinhole model geometry and mapping, courtesy of HediVision [2021]	8
2.4	Camera distortions	10
2.5	Different activation functions, courtesy of Hamdan [2018]	12
2.6	Max pooling illustration, by of Podareanu et al. [2019]	12
2.7	Example of CNN architecture; courtesy of Ferguson et al. [2017]	13
2.8	Construction of the desired position based on two path parameters, adapted from Moen [2021]	14
3.1	The CS Saucer with the latest module installed	18
3.2	Signal and power flow between system components.	20
3.3	Basic ROS concept	21
3.4	ROS-architecture of the CS Saucer.	22
3.5	The CSS, CSE1 and CSAD deployed in the MC lab basin	23
3.6	Body-fixed coordinate frame	24
4.1	Block diagram of feedback control system	28
5.1	Block diagram of SA system	31
5.2	Sensor integration for the CS Saucer	32
5.3	Camera calibration process	33
5.4	Camera view along with 2D point cloud from lidar, displayed in RViz	34
5.5	Picking laser coordinate in image pixels	35

5.6	Visual detection of the CS Enterprise in the MC Lab	37
5.7	Laser scan sample of the MC lab basin	38
5.8	K-means principle	39
5.9	Situational awareness framework	40
5.10	Object detection after fusing image and lidar measurements, using Cyber- ship II as the target.	41
8.1	Fixed thruster configuration on the CS Saucer.	54
9.1	Detection of the Cybership II	58
9.2	Initial range test	59
9.3	Range measurements after calibration	60
9.4	Influence of wall points on centroid placement	61
9.5	Estimated obstacle postions from object detection	62
9.6	Centroid computations compared to qualisys measurements	62
9.7	North-East plot for Scenario 1	64
9.8	Position and velocity estimates for Scenario 1	64
9.9	Force commands for Scenario 1	65
9.10	Actuator commands for Scenario 1	65
9.11	North-East plot for Scenario 2	66
9.12	Position and velocity estimates for Scenario 2	67
9.13	Force commands for Scenario 2	67
9.14	Actuator commands for Scenario 2	68
9.15	Failure in evasive maneuver due to overlapping obstacle regions	69
B.1	Tuning GUI	X
C.1	Calibration window	XII
C.2	RViz window for calibration	XIII
C.3	Choosing corresponding pixel value	XIII
E.1	Mapping of DS4 controller for manual and atomatic control	XVII

List of Tables

- 3.1 Technical specification for the CSS 19
- 3.2 Basin dimensions 22

- 8.1 Thruster configuration 55

- B.1 Required software VII

Nomenclature

List of Abbreviations

2D	–	Two dimensional
3D	–	Three dimensional
AIS	–	Automatic identification system
ASV	–	Autonomous surface vessel
CBF	–	Control Barrier Function
CCD	–	Charged-Coupled Devices
CLF	–	Control Lyapunov Function
CNN	–	Convolutional Neural Network
CO	–	Coordinate Origin
COLAV	–	Collision Avoidance
CSAD	–	Cybership Arctic Drillship
CSE1	–	Cybership Enterprise I
CSS	–	Cybership Saucer
DOF	–	Degree(s) of freedom
DP	–	Dynamic Positioning
fps	–	Frames-per-second
GNC	–	Guidance, Navigation & Control
GNSS	–	Global navigation satellite system
GPU	–	Graphical Processing Unit
IMU	–	Inertial Measurement Unit
LiDAR	–	Light Detection and Ranging
MC-Lab	–	Marine Cybernetics Laboratory
NED	–	North-East-Down
NN	–	Neural Network
OS	–	Own Ship
QP	–	Quadratic Programming
RADAR	–	Radio Detection and Ranging
ROS	–	Robot Operating System
SA	–	Situational Awareness

List of Symbols

Perception nomenclature

- \mathcal{C} = Camera coordinate frame
- \mathcal{L} = Lidar coordinate frame
- \mathbf{K} = Camera calibration matrix
- \mathbf{H} = Camera projection matrix

Control nomenclature

- $\boldsymbol{\eta}$ = Generalized position vector
- $\hat{\boldsymbol{\eta}}$ = Estimated position vector from observer
- $\boldsymbol{\eta}_d$ = Reference position vector
- \mathbf{p} = Position vector in NE-frame
- $\boldsymbol{\nu}$ = Generalized velocity vector
- $\hat{\boldsymbol{\nu}}$ = Estimated velocity vector from observer
- $\boldsymbol{\nu}_d$ = Reference velocity vector
- \mathbf{b} = System bias vector
- $\hat{\mathbf{b}}$ = Estimated bias vector from observer
- $\boldsymbol{\tau}$ = Generalized force vector
- X = Force acting in surge
- Y = Force acting in sway
- N = Moment acting in yaw
- \mathbf{M}_{RB} = Rigid Body Mass/Inertia matrix
- \mathbf{M}_A = Added mass matrix
- \mathbf{C} = Hydrodynamic restoring force matrix
- \mathbf{D} = General hydrodynamic damping matrix
- \mathbf{D}_ν = Nonlinear hydrodynamic damping matrix
- \mathbf{R} = Rotation matrix
- ψ = Vessel heading
- s = Path parameter
- α = Azimuth angle
- $\mathbf{B}(\alpha)$ = Thruster configuration matrix
- \mathbf{B}_{ext} = Extended thruster configuration matrix
- $B(s)$ = Control Barrier Function

Introduction

1.1 Motivation

In recent years the field of autonomy has seen significant progress. Great advances in sophisticated perception systems and sensors combined with computational power skyrocketing over the last decade enable the concept of fully autonomous vehicle control to become a reality. The automotive industry is perhaps the sector that is furthest along in autonomous control, with companies like Google developing and performing tests of self-driving cars. However, autonomy is also a highly relevant subject in the marine sector. At the time of writing, research has progressed to the point of autonomous commercial projects being launched. The most famous example of this is likely the *Yara Birkeland*. The Birkeland is intended to function as a fully autonomous container vessel traveling between port of Porsgrunn and Breivik in eastern Norway.

Smaller surface vessels operating at a lesser scale, over shorter distances, and with specific objectives is another focus area where autonomy has high potential. The Autoferry project at the Norwegian University of Science and Technology (NTNU) is a cross-disciplinary research project that investigates new concepts and methods to enable the development of autonomous passenger ferries for the transport of people in urban water channels [NTNU, 2021]. The project utilizes the *milliAmpere*, a small-scale pilot ferry, as an experimental platform. At the time of writing, a 3-hour, fully autonomous operation has been conducted with this vessel to great success [Zeabus, 2021].

Although significant progress has been made, the path to full autonomy is still long, and much research and development are still required. One of the most significant challenges for autonomy lies in safety assurance. If ASVs are to achieve commercial success, technical and perceived safety must be ensured. Technical safety is guaranteed in the design of the vessel control system. This requires the design of embedded safety functions in all control layers. Such functions include guidance with nominal obstacle/collision avoidance, online risk assessment, risk-based control mitigation, and minimum risk conditions (MRC). Commonly, such methods require a coherent image of the environment the vessel is operating in, or *situational awareness* (SA).

Situational awareness is achieved by interpreting data from sensors. The number and complexity of these may vary depending on the operation's needs. For example, a smaller ASV operating in urban environments may benefit from a LiDAR and optical sensors such

as cameras. On the other hand, a conventional ship on open sea would prefer a radar with a more extensive range. The necessary hardware to achieve situational awareness exists today; the challenge lies in combining the data into something meaningful. Each sensor comes with its own frame of reference that must be related to a common world frame through the process known as sensor fusion.

1.2 Objectives and Scope of Work

This thesis aims to develop a situational awareness system for the experimental vessel CS Saucer, based on a monocular camera and a 2D lidar scanner. This system should be able to detect obstacles during maneuvering operations. When an obstacle is detected, control barrier functions are employed to ensure that the maneuvering trajectories are safe and collision is avoided.

A secondary objective of this thesis is to upgrade the CS Saucer. This means upgrading the vessel's hardware and harmonizing the control system with the remaining vessels available in the MC Lab. To achieve this, the following scope of work is defined:

1. Perform a background and literature review to provide information and relevant references on:
 - Autonomous surface vessel and situation awareness functions.
 - Sensor fundamentals and modeling of camera and lidar.
 - Machine vision fundamentals, particularly camera- and lidar-based sensor fusion, target tracking algorithms, etc.
 - Object detection and classification based on neural network methods such as CNN.
 - Relevant theory concerning CBFs for collision avoidance.

Write a list with abbreviations and definitions of terms and symbols relevant to the literature study and project report.

2. Develop RP4 and ROS-based vessel maneuvering control system for CSS, using the same platform as for CSE1 and CSAD:
 - The vessel should have modules implemented in ROS for Mission Manager, Situation Awareness, Guidance, Observer, Controller, and Thrust Allocation
 - Since CSS is omnidirectional with almost negligible resistance in yaw, one should consider heading control and position control individually by decoupling yaw control from surge-sway control. This needs particular focus on how to do the thruster configuration and thrust allocation in order to tightly stabilize the heading. Develop an optimal thrust allocation algorithm correspondingly for the CSS. Test several thrust configurations and discuss the results.
 - Perform testing in MC-Lab to tune the observer, guidance, and controller modules in order to get the maneuvering-based control system to perform well.
3. Develop a SA system based on camera and lidar for the CSS. This includes sensor fusion and object detection. A CNN should be considered for detecting and classifying relevant objects in MC-lab. Test the system and discuss the module's ability to detect and track targets.

4. Let detected objects classified as relevant targets be associated with a Control Barrier Function (CBF), that is, each target quantifies a new CBF when detected, where the CBF enters the control loop for collision avoidance. Develop new CBF-based maneuvering control modules that ensure safe maneuvering with anti-collision. Consider the logic for how a CBF emerges and later vanishes. Present and discuss how the CBF-based SA, Guidance, and Controller modules interact to ensure safe maneuvering; that is, present this by a detailed flowchart, timing diagram, logics, etc.
5. Test the overall system in MC-Lab for a set of specified test scenarios. Present and discuss/critique the overall functionality and resulting performance.

1.3 Contributions

The first contribution of this thesis is the development of a complete autonomous control system for the experimental platform CS Saucer, which aims at safe maneuvers and obstacle avoidance. This includes a situational awareness module, observer, guidance system, control system with thrust allocation, and drivers for the hardware components. The situational awareness system is centered around two sensors; a monocular camera and a 2D LiDAR. The sensors are mounted on a new top plate for the vessel, and methods for detecting environmental obstacles and fusing the measurements are presented and implemented. The guidance functionality generates safe maneuvering references by incorporating a control barrier function. The control system utilizes a cascade backstepping control Lyapunov design that decouples the heading control from the positional control. Fixed and varying thrust allocation modules are also provided, one of which introduces a heading priority scheme. The control system is developed predominately in Python, using the Robot Operating System as its framework. The entire system consists of some 3000 lines of code, all made available for future use in the MC Lab. This thesis also contributes tools, documentation, and procedures for the system.

Physical experiments are performed as part of this thesis. As such, the thesis contributes a study on the practical feasibility of a CBF-based control architecture for ASVs.

1.4 Outline

The report is structured as follows. Chapter 2 gives the theoretical background for the presented work along with a literature review of existing studies relevant to the thesis. Chapter 3 gives background information and a description of the experimental platform, including specifications of critical hardware and software. Following the background, Chapter 4 formulates this thesis's control problem this thesis aims to solve. Then, chapters 5, 6, 7 and 8 detail the proposed implementation of each module in the autonomous control system. These are the situational awareness-, observer-, guidance- and control module, respectively. Together, these modules should solve the stated problem. Chapter 9 presents the different scenarios the system was tested under and the corresponding results with discussion. Finally, 10 concludes the thesis and its problem statement. Further work is also presented here.

Background and literature review

This chapter provides relevant references and background information on subjects essential to the control of an autonomous surface vessel. It includes situational awareness, sensor fundamentals, computer vision fundamentals, relevant reference frames, guidance and control functions for autonomous ships, and collision avoidance.

Relevant parts of this literature review were done in the pre-project study in [Solheim, 2021] and are reproduced for this thesis.

2.1 Situational Awareness in autonomous surface vessels

Formally, situation awareness (SA) is defined as a person's "perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future" [Endsley, 1988]. This definition presents three levels to SA [Ottesen, 2014], which we can relate to an autonomous surface vessel (ASV):

1. Perceiving the environment around the vessel
2. Comprehending the current situation around the vessel
3. Projecting the future states around the vessel

In general, situation awareness for autonomous vessels is achieved through sensor measurements and subsequent processing of the provided data. Figure 2.1 provides a typical control architecture for an autonomous system. The starting module of this is the SA-block. Here measurements from different sensors are gathered, processed (Perceiving), and fused to achieve a coherent view of the environment (Comprehending). Then this is used in either supervisory control or motion planning for the given vessel (Projecting).

ASVs utilize many sensor configurations, depending on the operational goal, vessel specifications, limitations, and the operation environment. For example, a conventional ship typically achieves SA employing RADAR, automatic identification system (AIS) messages, and GNSS [Schöller et al., 2020]. ASVs, on the other hand, generally incorporate electro-optical sensors to compensate for the lack of human operators. These include infrared-

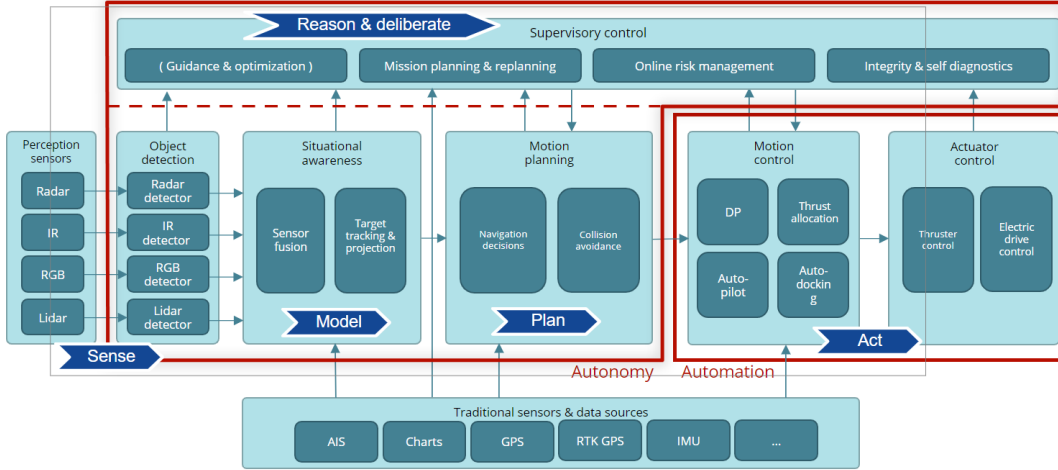


Figure 2.1: Architecture of an ASV [Smogeli, 2021]

(IR) and RGB cameras or lidar that work with traditional sensors and data sources such as inertial measurement units (IMU), GPS, or radar. An example is the autonomous ferry *Milliampere 2*, which is equipped with a sensor array consisting of GNSS, Lidar, Radar, cameras, IMU, and ultrasonic distance sensors. [Moen, 2021]. The electro-optical sensors are processed separately and used for detection before they are fused with measurements from the traditional sensors.

2.2 LiDAR

Light Detection and Ranging, or LiDAR for short, is an electro-optical sensor that measures distances to an object. This is achieved by sending out a laser pulse and measuring the elapsed time until the signal is bounced back to the receiver. The range is then

$$r = \frac{c}{2}(t_{rx} - t_{tx}) \quad (2.1)$$

where c is the speed of light, t_{rx} is the reception time and t_{tx} is the transmission time.

Lidars can take many forms, but in autonomous marine operations, a 360° 3D lidar is generally preferred. This type of lidar utilizes a rotating detector array, which measures the azimuth angle, elevation angle, and range. It uses this information to produce a point cloud that can be utilized for mapping the environment or estimating the position of moving objects [Debeunne et al., 2020].

The lidar produces measurement signals in the polar coordinate frame, which can easily be converted to the Cartesian coordinate frame through basic trigonometry:

$$x = r \cos \omega \sin \alpha \quad (2.2)$$

$$y = r \cos \omega \cos \alpha \quad (2.3)$$

$$z = r \sin \beta \quad (2.4)$$

Here α denotes the rotation around the z-axis of the lidar, ω is the rotation around the x-axis, and β is the rotation around the y-axis.

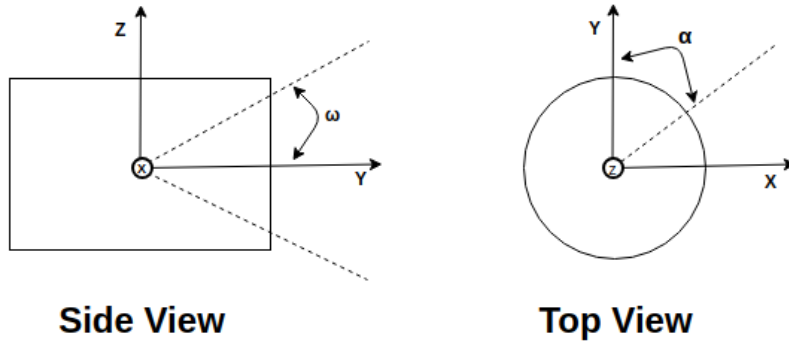


Figure 2.2: Lidar coordinate system

This thesis utilizes a 2D laser scanner, meaning it only provides measurements in the horizontal xy -plane. Thus, the z -axis and angle ω can be ignored. A given point measured by the 2D scanner is then

$$P_j^l = \begin{bmatrix} r_j \sin \alpha_j \\ r_j \cos \alpha_j \end{bmatrix} \quad (2.5)$$

in Cartesian coordinates. Here α_j is the azimuth angle and r_j the range of the j^{th} scan. The subscript l denotes the point in a Cartesian frame centered in the lidar.

2.3 Camera

A camera is also an electro-optical instrument that projects the 3D environment in its field of view into a 2D image. Mathematically, it can be modeled according to the *pinhole camera model* presented by Hartley et al. [2015].

2.3.1 Deriving the Pinhole model

We let the center of the projection be the origin of a Euclidean coordinate system, and define the image plane $z = f \in \mathbb{R}^2$. Our model will then map a point $[x \ y \ z]^T$ to the image plane $[\frac{fx}{z} \ \frac{fy}{z} \ f]^T$ using similar triangles, as illustrated in Figure 2.3. We ignore the last point, giving us the central projection mapping from \mathbb{R}^3 world space to \mathbb{R}^2 image coordinates:

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \rightarrow \begin{bmatrix} \frac{fx}{z} & \frac{fy}{z} \end{bmatrix}^T \quad (2.6)$$

Assuming homogeneous coordinates for both image and real world, one could then in theory express the central projection as a linear mapping between coordinates by matrix multiplication:

$$\mathbf{x}_c = \mathbf{A}\mathbf{x} \quad (2.7)$$

where

- \mathbf{x} is the real world point given by homogeneous coordinates

$$\mathbf{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \quad (2.8)$$

- \mathbf{A} is the homogeneous camera projection matrix

$$\mathbf{A} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.9)$$

- \mathbf{x}_c is the corresponding point in image plane

$$\mathbf{x}_c = \begin{bmatrix} fx & fy & z \end{bmatrix}^T \quad (2.10)$$

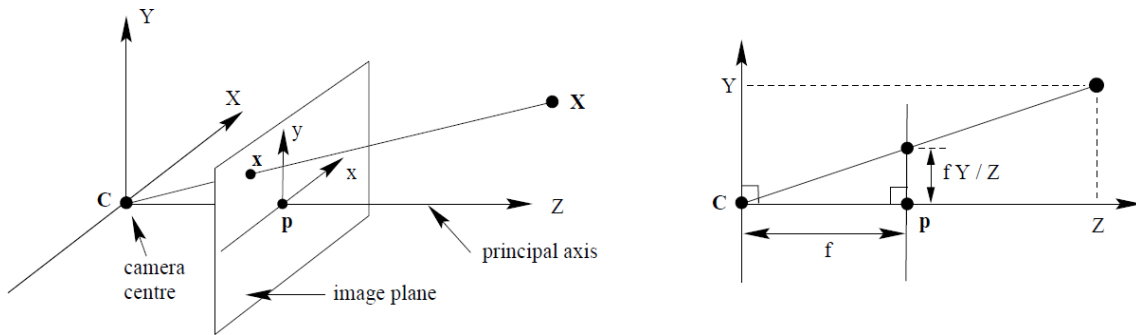


Figure 2.3: Pinhole model geometry and mapping, courtesy of HediVision [2021]

In practice we can not assume homogeneous coordinates since the image origin is typically not aligned with the principal point. Thus, we must augment our transformation to compensate for this:

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \rightarrow \begin{bmatrix} \frac{fx}{z} + p_x & \frac{fy}{z} + p_y \end{bmatrix}^T. \quad (2.11)$$

Here $[p_x, p_y]$ are the coordinates of the principal point. Accordingly, we must expand our mapping in (2.7) to:

$$\begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_{1 \times 3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad (2.12)$$

where

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.13)$$

is the camera calibration matrix. This calibration matrix assumes that all pixels are square. For digital cameras that utilize charge-coupled device (CCD) sensors, this may not be the case [Hartley et al., 2015]. Therefore, the calibration matrix for the camera must be further generalized. Defining the number of pixels per unit distance in image coordinates as m_x and m_y we multiply (2.13) with the factor matrix $diag(m_x, m_y, 0)$. This gives us the new generalized camera calibration matrix:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.14)$$

where $\alpha_i = fm_i$ represents the cameras focal length in pixel units in x-y-coordinates and $x_0 = m_x p_x$, $y_0 = m_y p_y$ represent the coordinates of the principal point in terms of pixel dimensions. These parameters are known as the *intrinsic* camera parameters and are unique for each model. Numerous methods to determine these values have been proposed, and typically revolve around calibration using a checker-board as detailed in Zhang [2000] and Debeunne et al. [2020].

So far all coordinates are assumed to be of the camera coordinate frame, but for the intents and purposes of this project, points should be expressed in terms of a common euclidean coordinate frame. Consequently, the camera frame should be transformed to the *world coordinate frame* (WCF). Any geometric relation between two Euclidean coordinate frames can be expressed with a rotation and a translation. For the camera frame and WCF, this is:

$$\mathbf{x}_c = \mathbf{R}\mathbf{x}_{wcf} + \mathbf{t} \quad (2.15)$$

Combining (2.12) with (2.15) we get the *camera-matrix* for relating a point represented in world coordinates with its pixel coordinates, that is,

$$\mathbf{M}_c = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}, \quad (2.16)$$

here \mathbf{R} and \mathbf{t} are a rotation-matrix and a translation vector relating the camera orientation to the world coordinate frame, respectively. These are referred to as the cameras *extrinsic parameters* and are subject to change. In total the camera matrix M_c has 10 degrees of freedom; 4 for \mathbf{K} , 3 for \mathbf{R} and 3 for \mathbf{t} .

2.3.2 Distortion

So far, the assumption is that linear models are sufficient to model a camera's image process accurately. Unfortunately, this assumption does not hold for most real-world lenses as they are subject to different nonlinear disturbances. One of the significant contributors here is *radial distortion*. This is prevalent in lenses with a wide field of view-angle and small focal length; attributes typical for lenses used in machine vision applications [Hartley et al., 2015].

Radial distortion is caused by light rays bending more around the lens edges than at the optical center, as visualized in Figure 2.4.

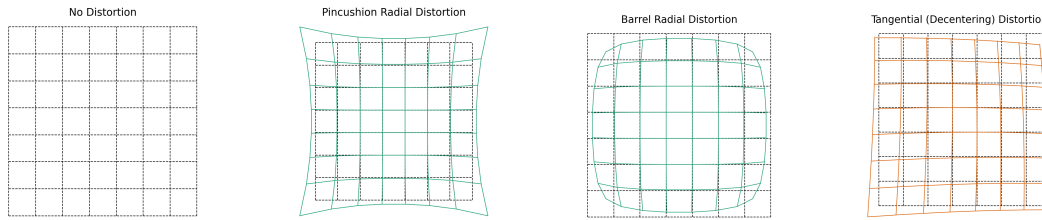


Figure 2.4: Camera distortions

This can, however, be compensated for by applying a correction to the image as a pre-processing step, effectively making the model linear again. Let the image plane coordinates for a given point in the camera frame be denoted by $[\tilde{x} \ \tilde{y}]^T$. From (2.6) we get that for the ideal, non-distorted pinhole camera, a world point is projected in the camera coordinate system to the image plane in normalized coordinates as

$$[\tilde{x} \ \tilde{y} \ 1]^T = \frac{1}{z} [x \ y \ z] \quad (2.17)$$

The actual coordinates of the image are then related to the ideal point by a radial displacement, which is modeled as the polynomial [Hartley et al., 2015]

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x}(1 + k_1 r^2 + k_2 r^4 + \dots) \\ \tilde{y}(1 + k_1 r^2 + k_2 r^4 + \dots) \end{bmatrix} \quad (2.18)$$

where $[x_d \ y_d]^T$ is the distorted image coordinate and $(1 + k_1 r^2 + k_2 r^4 + \dots)$ is a distortion factor. This factor is a function of the radius $r = \sqrt{\tilde{x}^2 + \tilde{y}^2}$. $k_i, i \in \{1, 2, \dots\}$ are the radial distortion parameters, which are typically determined in a camera calibration process. For most lenses, two parameters are sufficient to correct the radial distortion [Heikkila et al., 1997].

The second type of distortion that can occur in a camera lens is *tangential distortion*. This is typically caused by the lens and sensor not sitting parallel to each other, causing the image to become skewed.

We can compensate for the tangential distortion in pre-processing, similar to radial distortion. Heikkila et al. [1997] present a method of modeling the distortion as

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + 2p_1 \tilde{x} \tilde{y} + p_2 (r^2 + 2\tilde{x}^2) \\ \tilde{y} + p_2 (r^2 + 2\tilde{y}^2) + 2p_1 \tilde{x} \tilde{y} \end{bmatrix} \quad (2.19)$$

where p_1 and p_2 are the tangential distortion parameters, also determined in the calibration process. Combining (2.18) with (2.19) the full distortion model is given by

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + \tilde{x}(1 + k_1 r^2 + k_2 r^4 + \dots) + 2p_1 \tilde{x} \tilde{y} + p_2 (r^2 + 2\tilde{x}^2) \\ \tilde{y} + \tilde{y}(1 + k_1 r^2 + k_2 r^4 + \dots) + p_2 (r^2 + 2\tilde{y}^2) + 2p_1 \tilde{x} \tilde{y} \end{bmatrix} \quad (2.20)$$

The distorted pixel coordinates are then related to the normalized distorted coordinate by way of the camera matrix \mathbf{K} , that is,

$$\begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}. \quad (2.21)$$

2.3.3 The complete camera model

If given measurements from a camera in 2D coordinates and pixel values, while assuming the distortion as described in Section 2.3.2 has been corrected, the complete camera model can be expressed as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{HK} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \frac{1}{z_w} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \quad (2.22)$$

where $\begin{bmatrix} u & v \end{bmatrix}^T$ are the pixel coordinates and

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad (2.23)$$

is a projection matrix that removes the bottom row from the homogeneous representation. The factor $\frac{1}{z_w}$ scales the point \mathbf{x} to normalized image coordinates. As with any projection from \mathbb{R}^3 to \mathbb{R}^2 , there will be some loss of information when using a single camera. However, under the stated assumptions of corrected distortions this mapping can be inverted to give the normalized image plane projection of the image coordinates.

2.4 Supervised Learning for Machine Vision

Supervised learning is a subcategory of Machine Learning (ML) concerned with making predictions on unknown data based on experiences from known data. A framework carrying out this task is denoted a "Machine Learning model" or simply "Model". Most commonly, the model performance is quantified with a loss function, which turns the development of the model into an optimization problem, where the discrepancies between model behavior and desired behavior are minimized [S.-C. Wang, 2003].

A common approach for carrying out supervised learning in various applications is the emerging developments in neural networks, which emulate the connections between neurons in a biological brain. Especially in the field of visual classification and motion-tracking, the use of deep neural networks has recently gained substantial traction as a viable solution to problems such as classification and tracking.

2.4.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a subclass of Neural Networks utilized for detecting and classifying objects in images [Bishop, 2007]. The use of CNNs was popularized after Krizhevsky et al. [2012] saw great success with their method for reliably classifying images using deep neural networks. A CNN uses a cascade of convolutional neuron layers

in decreasing size, with the task of extracting features from an image into simpler parts such that a subsequent regular NN structure can more easily classify based on the features extracted from the preceding layers. CNN's are therefore often divided into two parts, the Feature Extractor and the Classifier.

Feature extraction

The feature extractor in a CNN consists of three separate layers: Convolution, Activation and Pooling [Gu et al., 2018]. *Convolution* is a mathematical operation relating two functions. This is mathematically expressed in (2.24), where the input \mathbf{x}_{ij}^l is a portion of points in the input (for instance, an array encoding pixel values in a small region of an image) whose position is indexed by i, j . The output is a feature map z_{ijk}^l . Several feature maps are often made in parallel, in which case they are numbered by the index k , yielding

$$z_{ijk}^l = (\mathbf{w}_k^l)^\top \mathbf{x}_{ij}^l + b_k^l. \quad (2.24)$$

The vector $(\mathbf{w}_k^l)^\top$ is a collection of weights describing the convolution filter, commonly called the *kernel*, for layer l . Each convolution operation also contains a bias parameter b_k^l . The purpose of the kernel is to isolate and highlight features in different parts of the input data.

Following the convolution operation, the data is forwarded through an activation function, which aims to introduce nonlinearity to the network, thus allowing the network to model nonlinear features [Gu et al., 2018]. Commonly used activation functions are sigmoid, tanh, and Rectified Linear Unit (ReLU) [LeCun et al., 2012], shown in Figure 2.5

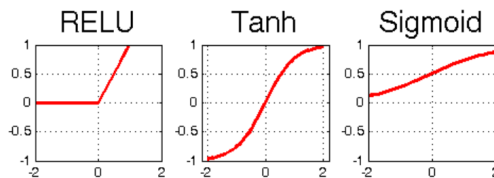


Figure 2.5: Different activation functions, courtesy of Hamdan [2018]

The pooling operation compounds and quantifies patterns in the input data, reducing the image size to produce a data structure containing the necessary features for classification. Typically, Max Pooling is used, extracting the highest value in a subset of the input grid, although other methods such as Average and Min Pooling are also used [Gu et al., 2018]. An illustrative example of Max Pooling is shown in Figure 2.6.

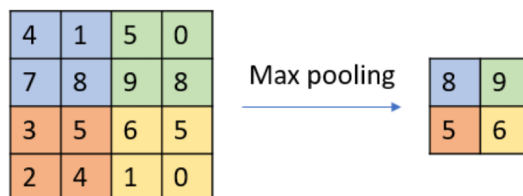


Figure 2.6: Max pooling illustration, by of Podareanu et al. [2019]

Note that this three-part architecture is often repeated several times, each repetition gradually reducing the size and complexity of the data.

Classifier

The classifier receives the extracted features from the feature extractor. Then, it produces an output with information on the captured image, in most cases classifying detected objects as a specific type of object and the location of detected objects in the form of a *bounding box*. The classifier usually consists of *dense layers* where the neurons in each layer are fully connected to the neurons in the layers directly preceding and following them [Gu et al., 2018].

The overall architecture of the network usually consists of several convolutional layers in a cascade, each followed by pooling layers, before a final classifier model with dense layers at the end. An example of this structure is shown in Figure 2.7 using *VGG16*, a model developed by the Visual Graphics Group at the University of Oxford [Simonyan et al., 2014], as an example. VGG16 is one of the most established CNNs in classification and motion-tracking systems, being utilized in several popular ML models.

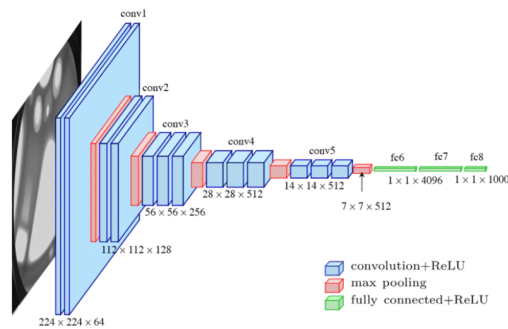


Figure 2.7: Example of CNN architecture; courtesy of Ferguson et al. [2017]

2.4.2 Single stage detectors vs. Dual Stage Detectors

Typically, object detection models are categorized into two major architectural types: single-stage detectors (SSD) and dual-stage detectors (DSD). The significant difference between the two is that the region of interest is first determined in the image in the two-stage object detection models. The detection is then performed only in the determined region of interest. The implication is that the extra stage in the DSD generally provides better accuracy at the cost of more computational power and time. Examples of widely used SSDs are "MobileScan v2" and the "You Only Look Once (YOLO)"-detectors, while "Regional Convolutional Neural Network (R-CNN)" and its newer iteration "Faster R-CNN" are famous examples of DSDs.

2.5 Path generation for maneuvering

This section presents the relevant theory for guidance when maneuvering an autonomous vessel. It assumes that the mission is to create a path between two or more waypoints (WP) $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$.

2.5.1 One-variable path parametrization

A typical way to generate a path is to continuously parameterize around a scalar path variable s_1 , which indicates how far along the desired position is. To this end, we define the desired position and its three derivatives.

$$\mathbf{p}_d(s_1) := \begin{bmatrix} x_d(s_1) \\ y_d(s_1) \end{bmatrix}, \quad \mathbf{p}_d^s(s_1) := \begin{bmatrix} x_d^s(s_1) \\ y_d^s(s_1) \end{bmatrix}, \quad \mathbf{p}_d^{s_1^2}(s_1) := \begin{bmatrix} x_d^{s_1^2}(s_1) \\ y_d^{s_1^2}(s_1) \end{bmatrix}, \quad \mathbf{p}_d^{s_1^3}(s_1) := \begin{bmatrix} x_d^{s_1^3}(s_1) \\ y_d^{s_1^3}(s_1) \end{bmatrix} \quad (2.25)$$

For a simple straight line between points, the desired position and its derivatives become

$$\mathbf{p}_d(s_1) = (1 - s_1)\mathbf{p}_1 + s_1\mathbf{p}_2 \quad (2.26)$$

$$\mathbf{p}_d^{s_1}(s_1) = \mathbf{p}_2 - \mathbf{p}_1 \quad (2.27)$$

2.5.2 Two-variable path parametrization

In cases where it is desirable to leave a straight line between the path, for example when encountering an obstacle, a second path parameter $s_2 \in \mathbb{R}$ can be introduced. This gives a path parameter vector $s = [s_1 \ s_2]^T$. Marley [2021] proposes a two parameter desired path by using the tangent vector \mathbf{T} between waypoints and the normal vector \mathbf{N}

$$\mathbf{p}_d(\mathbf{s}) := \mathbf{p}_0 + \mathbf{L}(s_1\mathbf{T} + s_2\mathbf{N}) \quad (2.28)$$

where $\mathbf{L} := |\mathbf{p}_1 - \mathbf{p}_0|$. As long as $s_2 = 0$, this path will correspond to the scalar value straight line path from Section 2.5.1. In instances where $s_2 \neq 0$, the variable corresponds to the desired paths deviation from the nominal straight line path. An illustration of the parametrization can be found in Figure 2.8. In path following it is also common to set a speed assignment $\mathbf{s} = \mathbf{v}_s$ which determines the speed at which a vessel follows the given path.

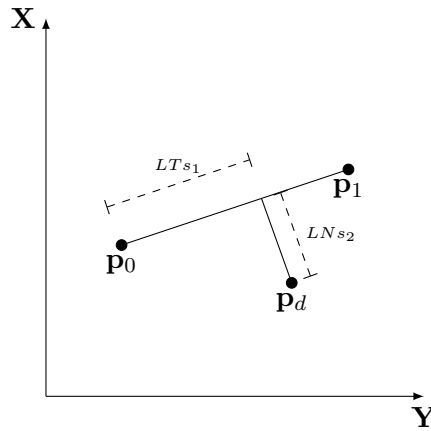


Figure 2.8: Construction of the desired position based on two path parameters, adapted from Moen [2021]

2.6 Control Barrier Functions

This section will briefly introduce control barrier functions (CBF), a promising new tool for examining the safety of a control system as outlined by Ames et al. [2019]. The development of CBFs is motivated by the significant focus placed on safety in modern control systems. Simply put, they play a similar role to a Lyapunov function, guaranteeing the safety of a control system where the Lyapunov function guarantees stability [Ames et al., 2019]. For this, we consider a generic nonlinear control affine system on the form

$$\dot{x} = f(x) + g(x)u, \quad x(0) = 0 \tag{2.29}$$

Here $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ are locally Lipschitz. $x \in \mathbb{R}^n$ is the state vector, and $u \in \mathbb{R}^m$ is the input vector. Next, we assume that a set \mathcal{C} exists that can be considered safe. The safety can then be guaranteed by restricting our system to the given set \mathcal{C} , ensuring that it never leaves. Accordingly, a barrier function should be defined as a continuously differentiable function $h : \mathbb{R}^n \rightarrow \mathbb{R}$ such that:

$$\mathcal{C} = \{x \in \mathbb{R}^n : h(x) \geq 0\} \tag{2.30}$$

$$\partial\mathcal{C} = \{x \in \mathbb{R}^n : h(x) = 0\} \tag{2.31}$$

$$\text{Int}(\mathcal{C}) = \{x \in \mathbb{R}^n : h(x) > 0\} \tag{2.32}$$

where $\text{Int}(\mathcal{C})$ is the interior of the set \mathcal{C} . The barrier function is kept positive in the interior of the safe set and thus guaranteeing safety. An example of a CBF application is Marley, Skjetne, Breivik, et al. [2020], where a CBF is combined with a hybrid feedback controller for heading control for obstacle avoidance. Later Marley, Skjetne, and Teel [2021] presented a way of incorporating a CBF into the guidance function of a vessel motion control system to plan a collision-free path.

Experimental Platform: CS Saucer

This chapter will provide a background on the chosen experimental surface vessel, CS Saucer (CSS), and its operational environment, the Marine Cybernetics lab (MC-lab). Significant effort has been put into upgrading the hardware and software of CSS as part of the pre-project [Solheim, 2021] and this thesis. Thus, a section is dedicated to outlining the upgrades and reasoning for the decisions made in the process.

3.1 Background and motivation

Initially designed by Idland [2015], in collaboration with Ph.D. candidate Andreas Reason Dahl, the CSS is a highly maneuverable drone with a symmetric, circular hull. It was designed this way so behavior would be similar in surge and sway, thus yielding quicker response and maneuverability than a conventional ship model. Modularity was also an essential consideration in the design process, so the vessel uses interchangeable top modules that allow for various payload configurations. Sharoni [2016], for example, installed an inverted pendulum, while Ueland [2016] installed a LiDAR-scanner on a separate cover. This modularity was the main draw of the Saucer, as it would make integrating a camera with the already existing lidar relatively easy.

Recently, the other ships of the cyber fleet, namely the CS Enterprise 1 (CSE1) and the CS Arctic Drillship (CSAD), had their National Instruments (NI) compactRIO embedded computers wholly replaced. The new state-of-the-art system uses a Raspberry Pi (RPi) running a Python and Robot Operating System (ROS) based control system. Idland [2015] initially implemented the control system of the CSS on a NI LabVIEW platform as well, where the embedded hardware device NI myRIO functioned as the central processing unit. Ueland [2016] and Sharoni [2016] later replaced this system with a Robot Operating Software (ROS) based solution, running on an RPi 2 as the embedded computer in conjunction with an Arduino as part of their master theses. This provided even more flexibility in development due to the accessibility of ROS-compatible hardware and software. Unfortunately, the hardware and software Ueland [2016] developed is now primarily deprecated, so a second motivation for picking the CSS as the experimental platform was to upgrade the control system to state-of-the-art.



Figure 3.1: The CS Saucer with the latest module installed

3.2 Technical specification

As part of the control system upgrade performed in this thesis, several new components were installed on the vessel. Table 3.1 provides a technical specification of the current state-of-the-art hardware utilized on the vessel. It also includes software utilized during operations. Figure 3.2 illustrates the signal and power flow between each hardware component. For a more detailed review of the components, the reader is referred to either Solheim [2021] or Ueland [2016]. Components and software added as part of the control system upgrade are marked with a star (\star).

Technical Specification	
Software	
Operating System	Ubuntu 20.04 LTS (Server version)*
ROS distribution	Noetic*
Hardware	
Component	Description
Raspberry Pi 4b*	Embedded computer for the vessel. Handles running ROS-nodes and communication between components in the system. Processor: Quad-core Cortex-A72 @ 1.5 GHz. RAM: 8 GB LPDDR4-3200 SDRAM Power: 5V via USB-C
Arduino Mega	Embedded circuit board responsible for transmitting appropriate PWM signals to each component. Communicates with the RPi 4b via USB. Duty cycle between 4.3 and 9.4 %.
Raspberry Pi HQ camera*	Camera module Resolution: 4056x3040 (12.3 Megapixels) Framerate: 30 fps (at highest resolution) Sensor: SONY IMX477 Lens: 6mm wide angle
RPLidar A1	Low cost 360° 2D laser scanner with adjustable rotation speed. Effective range: 12 m Sampling rate: 2000 Hz Angular resolution: 1 deg Range accuracy: 1% < 3 m, 2% ∈ [3, 5] m, 2.5% > 5 m
Torpedo 800	Motor drive for the three azimuth thrusters. Can spin the propellers clockwise or counterclockwise.
Graupner Schuttel drive unit II	Servo driver to set azimuth angle. Can rotate on the interval $[-114^\circ, 114^\circ]$.
Traxxas LiPo	Three cell, 11.1 V lithium polymer battery. Powers all devices on the vessel. At full charge, either 640 mAh or 500 mAh, depending on the battery used, it can power the system for several hours.

Table 3.1: Technical specification for the CSS

A second computer, referred to as the operator computer, is used together with the embedded computer on the CSS. The primary purpose of the computers is to run the computationally expensive visual detection and fusion nodes. Ideally, the computer should have a GPU to power the CNN, but since the system is designed with low computational availability in mind, a powerful CPU suffices. Therefore, for most of this thesis, a Dell laptop equipped with an 8-core Intel i5 vPro processor was utilized. Like RPi, the computer ran Ubuntu 20.04 LTS and ROS Noetic.

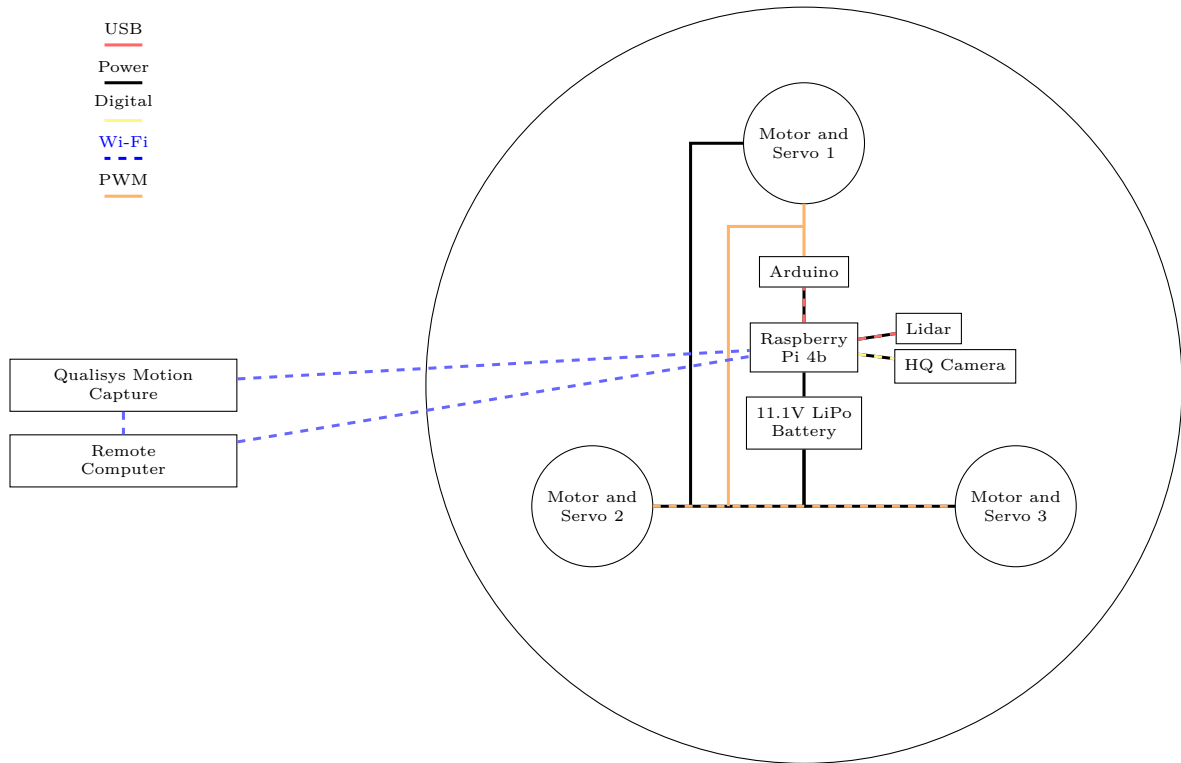


Figure 3.2: Signal and power flow between system components.

3.3 Software

3.3.1 Robot Operating System

The framework of the vessel’s new control system is entirely based on the Robot Operating System (ROS). This section will, therefore, provide a brief introduction to ROS and its basic concepts.

Introduced in 2007, ROS is an open-source project that provides tools, libraries, and conventions for robot applications. It functions as a meta-operating system (OS) handling services you would expect from a conventional OS. These include hardware abstraction, message-passing between processes, and package management.

A ROS process is represented as a node in a graph architecture. Nodes are connected to edges known as topics, through which they can pass messages to one another. They can also provide and make service calls to each other and send or retrieve data from a common parameter server known as the ROS-master. The ROS-master registers all active nodes to itself and establishes the peer-to-peer communication network of the nodes. Figure 3.3 illustrates the basic communication of a ROS-system.

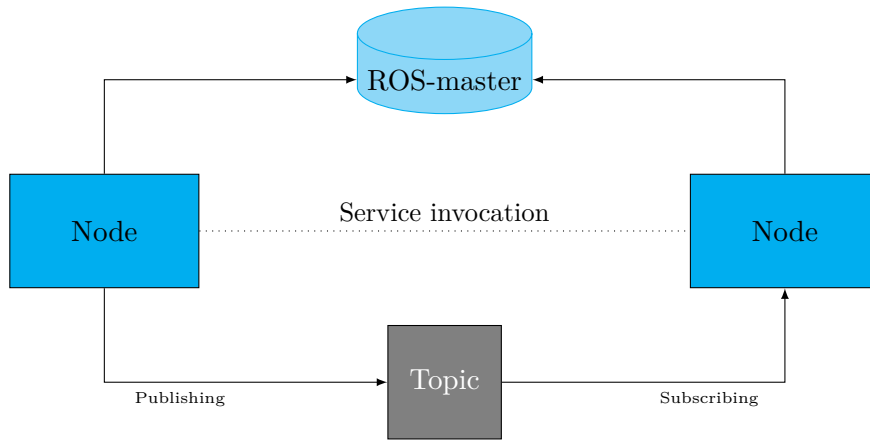


Figure 3.3: Basic ROS concept

This decentralized architecture is the main strength of ROS, as it allows nodes to be run on separate, networked hardware. As each node process is isolated and messages passed between standardized, the implementation language of the node is also irrelevant. Effectively this means that one can run one or more nodes written in C++ in conjunction with nodes written in, for example, Python. This ties in with the last strength, the ROS ecosystem. ROS offers many easy and accessible software for robots as an open-source project, making integrating sensors a simple task. Most hardware comes with ROS support from the manufacturer or a third-party individual. As mentioned before, the software language is irrelevant, so one can easily download a C++ ROS driver and run it with a primarily Python-based system.

3.3.2 ROS architecture

Figure 3.4 illustrates the different node processes and message flow in the proposed ROS vessel control system. Nodes are depicted as circles, with orange and red being sensor-related, blue being control system modules, and blue-grey being hardware nodes. *Topics* are rectangles connected to nodes. If an arrow points from a node to the topic, the node is publishing to the relevant topic. Accordingly, arrows pointing to nodes from topics mean that the node subscribes to the given topic. The *gain server* is a parameter-server that receives control- and observer gains from the operator, allowing the user to tune the system during operation.

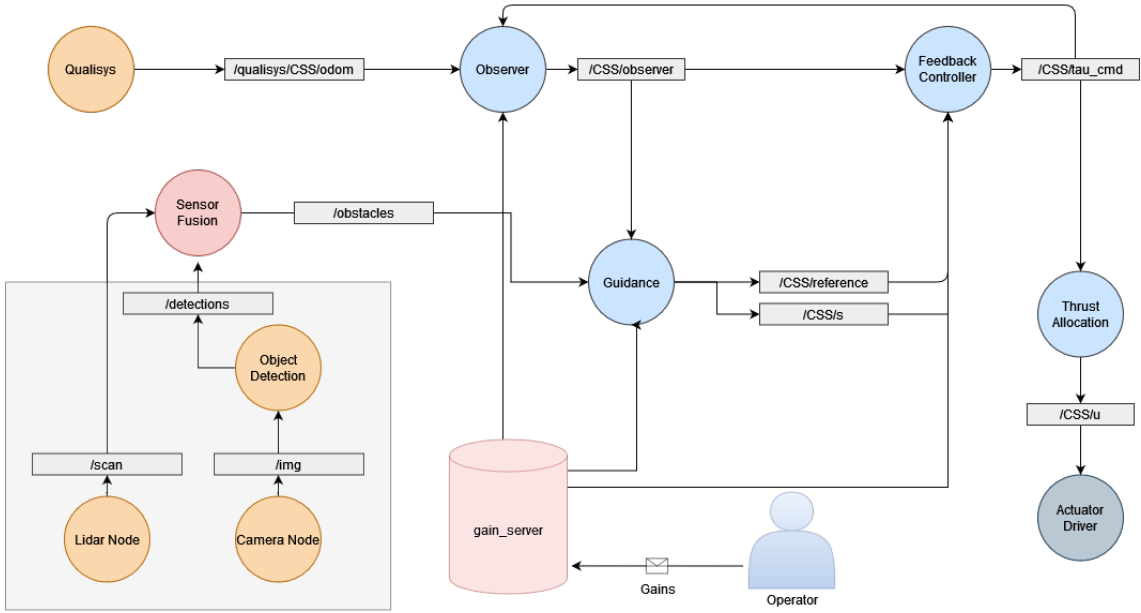


Figure 3.4: ROS-architecture of the CS Saucer.

3.4 The Marine Cybernetics Laboratory

The CSS will be operated inside the Marine Cybernetics Laboratory (MC-Lab), a small wave basin suitable for testing model scale vessels. The basin has a wave generator and an advanced instrumentation package, including a Qualisys Motion Capture system. This, along with relatively small dimensions

Length	Width	Depth
40 m	6.45 m	1.5 m

Table 3.2: Basin dimensions

make it a particularly suitable environment to test motion control systems for marine vessels. It is, however, important to note that the entire 40 meters can not be used for operations. The control system relies on the feedback from Qualisys, meaning it can only operate when in the range of the cameras. Thus, the actual operation environment constitutes an area of about 8×6 meters in the middle of the basin.

The MC lab is also the home of the 'Cyberfleet,' a collection of model scale vessels specifically utilized to test control systems. To test the collision avoidance capabilities of the CSS, these vessels will act as obstacles in the basin during operations. They can either be maneuvered manually using a joystick controller or fixed in place on the surface using weights as illustrated in Figure 3.5. For more detailed information about the MC Lab and its instrumentation, the reader is referred to the laboratories' home page [NTNU, 2022].



Figure 3.5: The CSS, CSE1 and CSAD deployed in the MC lab basin

3.5 Vessel model

A mathematical model describing vessel dynamics is desirable in designing a control system and performing simulations. To this end, we first need to establish some notation and frames of reference.

3.5.1 Reference frames

This thesis uses two frames of reference when describing the position and orientation of the CS Saucer. These are the North-East-Down (NED) frame and the BODY frame.

Six distinct degrees of freedom are generally required to describe a floating vessel. These are usually defined in generalized coordinates

$$\boldsymbol{\eta} := [x \ y \ z \ \varphi \ \theta \ \psi]^\top, \quad (3.1)$$

where x , y , z (surge, sway, heave) define position, and φ , θ , ψ (roll, pitch, yaw) are the Euler angles, describing orientation [Fossen, 2021]. However, as most surface vessels are considered metacentrically stable, the restoring forces in heave, roll, and pitch will counteract any inclinations away from equilibrium points [Fossen, 2021]. This essentially means that only the DOFs surge, sway and yaw are required to describe a surface fairing marine craft. This reduces the position vector to

$$\boldsymbol{\eta} = [x \ y \ \psi]^\top \quad (3.2)$$

The NED-frame is used for the absolute position of the vessel. A GPS, for example, provides a description of the vessel's position along the cardinal directions for guidance

and navigation purposes [Fossen, 2021]. However, the CSS is not equipped with a GPS but receives absolute positions from the Qualisys Track Management system. Thus, the reference frame utilized by Qualisys will be referred to as NED for the purposes of this thesis.

The BODY frame is an inertial reference frame fixed to the rigid body in question, with axes defined from a Coordinate Origin (CO) of the body, aligning with the principal axes of motion, as seen in Figure 3.6.

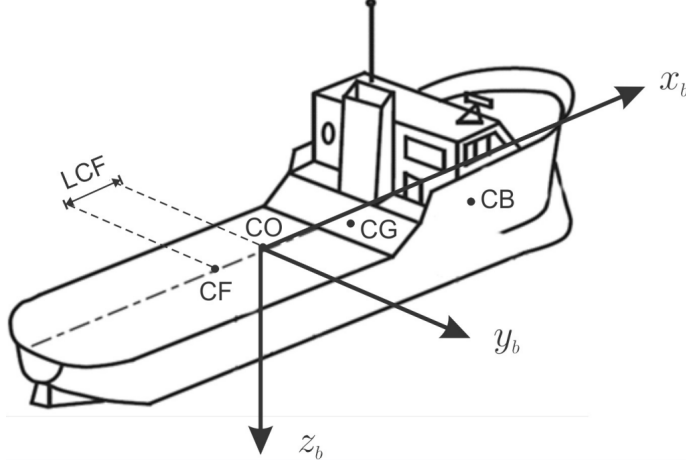


Figure 3.6: Body-fixed coordinate frame; from Fossen [2021]

The BODY-frame is helpful for control as its axis aligns with the principal axis of motion. This simplifies expressing velocities, accelerations, and forces acting on a vessel. The 6-DOF velocity vector $\boldsymbol{\nu}$, in generalized coordinates are given in the body-frame as

$$\boldsymbol{\nu} := [u \ v \ w \ p \ q \ r]^\top, \quad (3.3)$$

where the components describe linear and angular body-frame velocity in the 6 DOFs [Fossen, 2021]. For a surface vessel, (3.3) can be reduced to

$$\boldsymbol{\nu} = [u \ v \ r]^\top, \quad (3.4)$$

3.5.2 Model

Starting from the equation of motion for a vessel at sea [Fossen, 2021], one can derive the necessary equations for the control model

$$\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}\boldsymbol{\nu} + \mathbf{M}_A\dot{\boldsymbol{\nu}}_r + \mathbf{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}\boldsymbol{\nu}_r + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}_{ext}, \quad (3.5)$$

where:

- $\boldsymbol{\nu} = [u \ v \ r]^\top$ is the body-fixed velocities in surge, sway and yaw.
- $\boldsymbol{\nu}_r$ is the body-fixed velocities relative to local current in surge, sway and yaw
- \mathbf{M}_{RB} and \mathbf{M}_A is the vessel inertia matrix for mass and added mass

- \mathbf{C}_{RB} and $\mathbf{C}_A(\nu_r)$ is the vessel Coriolis matrix for rigid body and added mass respectively.
- $\mathbf{D}(\nu_r)$ is the nonlinear damping matrix
- \mathbf{D} is the linear damping matrix
- $\mathbf{g}(\eta)$ is the hydro-static restoring matrix
- $\boldsymbol{\tau}_{ext} = [X \ Y \ N]^\top$ is the external forces acting in surge, sway and yaw

A model derivation was conducted by Ueland [2016] for his master thesis, which will be reused in this project. The following assumptions were made for the model:

1. Zero current in the MC-lab, $\nu_r = \nu$.
2. The CSS is self-stabilizing by hydrostatic forces in heave, roll, and pitch. The rotations are considered small, so movements in surge, sway, and yaw are not affected by the configuration in pitch and roll.
3. No hydrostatic restoring forces in surge, sway and yaw, $\mathbf{g}(\eta) = 0$.
4. Constant frequencies, meaning that damping and added mass are also considered constant.
5. The hull of the CSS is assumed to be completely symmetric.

The resulting control design model is equivalent to the simplified model presented by Fossen [2021], which is a good representation of a 3DOF marine craft not affected by environmental forces, that is,

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{R}(\psi)\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + (\mathbf{C} + \mathbf{D} + \mathbf{D}(\nu))\boldsymbol{\nu} &= \boldsymbol{\tau}, \end{aligned}$$

where:

- The inertia matrix \mathbf{M}

$$\mathbf{M} = \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & 9.51 & 0 \\ 0 & 0 & 0.116 \end{bmatrix} \quad (3.6)$$

- The Coriolis matrix \mathbf{C}

$$\mathbf{C} = \begin{bmatrix} 0 & -9.51r & 0 \\ 9.51r & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.7)$$

- The linear damping matrix \mathbf{D}

$$\mathbf{D} = \begin{bmatrix} 1.96 & 0 & 0 \\ 0 & 1.96 & 0 \\ 0 & 0 & 0.168 \end{bmatrix} \quad (3.8)$$

- The non-linear damping matrix $\mathbf{D}(\boldsymbol{\nu})$

$$\mathbf{D}(\boldsymbol{\nu}) = \begin{bmatrix} 7.095|u| & 0 & 0 \\ 0 & 7.095|v| & 0 \\ 0 & 0 & 7.095|r| \end{bmatrix} \quad (3.9)$$

- The rotation matrix $\mathbf{R}(\psi)$

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

Problem formulation

The overall goal of this thesis is to develop a complete control system for the CSS. The system is to integrate a camera and lidar-based situational awareness system with control barrier functions so that the vessel can perform maneuvering operations with collision avoidance. The problem can be broken down into several sub-problems that need to be solved, tested separately, and subsequently integrated. This includes the following sub-systems; perception-sensors, situational awareness functions, and a motion control system consisting of a guidance module, state estimator, control system, and thrust allocation.

4.1 Sensors

To enable perception of the vessel's environment, a suite of sensors must be integrated into the CSS. This includes a monocular camera, a 2D lidar scanner, and baubles to enable positional tracking by the Qualisys motion capture system provided in the MC lab. A large portion of this work was done in the specialization project for this thesis, but as discussed in Solheim [2021], a more robust mounting of sensors is required for better precision. The sensor module should provide a steady stream of information through images, point cloud data, and positional coordinates to the situational awareness module.

4.2 Situational awareness

The situational awareness module takes the input from the vessel's sensor array, processes the signals, and creates a representation of the surrounding environment. In this thesis, we wish to detect and map obstacles in the vicinity of the vessel. To this end, computer vision methods must be applied. First, an object detector based on a CNN must be incorporated. This will provide a visual classification of the obstacles in the form of bounding boxes. Next, the point cloud and range data from the lidar must also be related to the detected obstacles. This is achieved through a clustering algorithm, which relates points into a cluster corresponding to the obstacle.

The next step is fusing the measurements from the detector and lidar-clustering algorithms into a common frame, relating each data to the correct obstacle. The result from the sensor fusion module will be an obstacle position and the range from the vessel to the obstacle.

4.3 Guidance problem

The guidance system generates the desired path of the vessel and continuously provides the feedback controller with a reference signal. This signal corresponds to the desired position and orientation of the vessel and can be expressed $\boldsymbol{\eta}_d = [x_d, y_d, \psi_d]^\top$. The ultimate goal of the guidance system is to generate a safe and feasible path for the vessel. This means it must ensure that the path is collision-free with regard to static and dynamic obstacles.

Collision avoidance is incorporated into the system through the use of CBFs. A CBF should be formulated so the vessel can maintain a safe distance from any identified obstacles. If the CBF deems the current vessel path unsafe, a collision-avoidance functionality will step in, generating new safe references.

4.4 Control problem

4.4.1 The maneuvering problem

The control system computes the generalized forces $\boldsymbol{\tau}_d$ that satisfy the vessel following the desired path. This type of control problem is generally called a "Maneuvering Problem" and was originally formulated by Skjetne [2005].

For a system output $\eta \in \mathbb{R}^m$, the desired path is all points represented by the set

$$\mathcal{P} := \{\eta \in \mathbb{R}^m : \exists \theta \in \mathbb{R} \text{ s.t. } \eta = \eta_d(s)\} \quad (4.1)$$

where η_d is continuously parameterized by s . Given the desired path (4.1) and a dynamic assignment, the maneuvering problem can be broken down into two tasks:

1. **Geometric Task:** For any continuous function $s(t)$, force the output η to converge to the desired path η_d ,

$$\lim_{t \rightarrow \infty} |\eta - \eta_d(s(t))| = 0 \quad (4.2)$$

2. **Dynamic Task, Speed Assignment:** Force the path speed \dot{s} to converge to a desired speed $v_s(s, t)$,

$$\lim_{t \rightarrow \infty} |\dot{s}(t) - v_s(s(t), t)| = 0 \quad (4.3)$$

Here the speed assignment is only one of several viable, dynamic tasks. It is, however, the most fitting for the operations performed in this thesis.

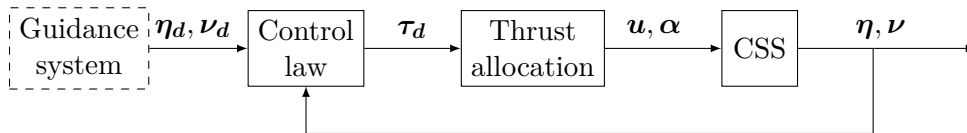


Figure 4.1: Block diagram of feedback control system

Figure 4.1 illustrates the intended control-module. Here the desired positions and velocities $\boldsymbol{\eta}_d, \boldsymbol{\nu}_d$ are fed from the guidance system to control law. The control law then computes the desired forces and moments $\boldsymbol{\tau}_d$, distributed to the vessel actuators via the thrust allocation module. For nominal control, a cascade backstepping controller shall be utilized.

4.4.2 Stabilizing heading

As the system will rely on a single monocular camera, a stable heading will be critical for maneuvering operations. Inconveniently, the hull of the CSS is circular, rendering the damping in yaw close to negligible. Physically, this means that the vessel tends to oscillate around the desired reference heading. Given that the heading is so critical, measures to counteract this must be implemented. The maneuvering control system must consider positional and heading control separately by decoupling yaw control from surge-sway control. Additional focus will also be placed on the thrust allocation to stabilize the heading. Here a strict allocation algorithm that prioritizes heading is required.

4.5 Limitations and assumptions

The scope of this thesis is quite large, so to make sure enough time is available to provide an adequate solution to the problem, some assumptions are made:

- Obstacles are assumed somewhat stationary. This means that when an obstacle is detected, it is reasonable to assume that it will stay in that position for the entire operation, with some drift.
- Obstacles will be limited to the class of 'boat,' i.e., the different cyber ship vessels available in the MC-lab.
- Operations will be performed in a calm sea state, i.e., one can safely assume no environmental forces such as current, waves, or wind.

Situational awareness

This chapter presents the procedures of the Situational Awareness (SA) system implemented for the CSS. The groundwork for deriving transformations and calibration procedures was done as part of the pre-project study and is repurposed for this thesis. Additionally, the chapter presents the implementation of a CNN, clustering algorithms, and sensor-fusion processes for the system. Figure 5.1 presents an initial block-diagram for the proposed SA-system.

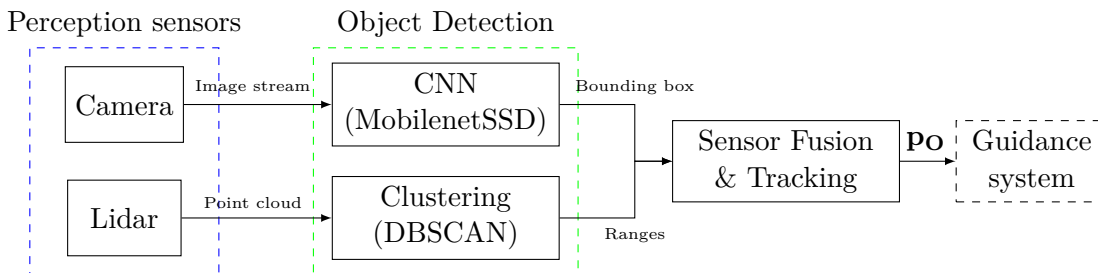


Figure 5.1: Block diagram of SA system

5.1 Sensor integration

A new top module was issued for the CSS as part of this thesis. On top of it, the two sensors were integrated along with tracking markers for the Qualisys motion capture system present in the MC-lab. The Qualisys system will act as a GPS for the vessel, providing the control system with measurements of the vessel's position and orientation. The markers are mounted on slender rods of varying heights along the circumference of the module. Then, the lidar is mounted in the center, directly above the CO of the CSS. The sensor is oriented so the x_l and y_l axis of the lidar align with the surge and sway directions, respectively. While the lidar provides a 360-degree scan, the interval of $[-90^\circ, 90^\circ]$ is most interesting. Thus markers are mounted to cause as little interference as possible and still provide an accurate estimation of the vessel states. The same constraints apply to the placement of the camera. It requires an unobstructed view but can not interfere with the lidar in the critical area. Therefore it is mounted on a slender rod right behind the lidar with enough clearance to the lidar that it sees the environment clearly. The camera is tightly connected to the vessel heading, always pointing in the direction that is considered

forward. Accordingly, it is mounted so that the z_c - and x_c -axis align with the surge and sway directions of the CSS.

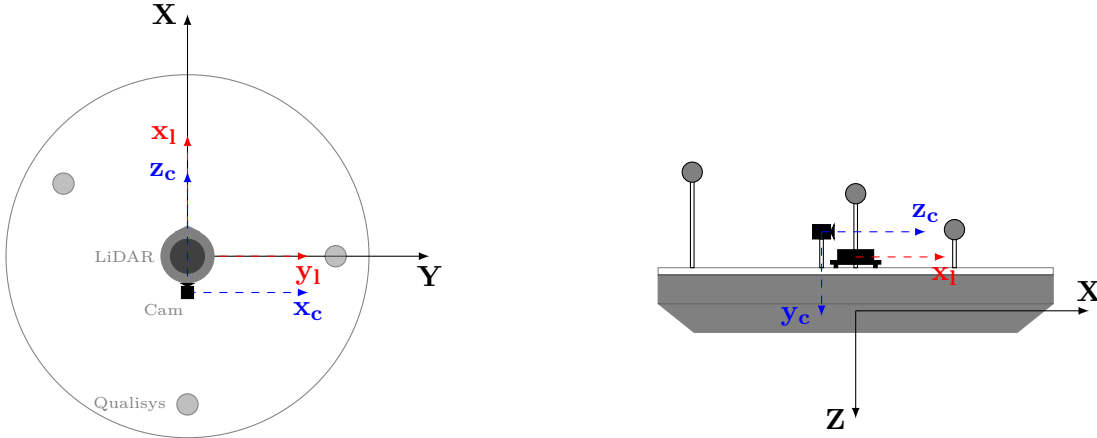


Figure 5.2: Sensor integration for the CS Saucer

The open-source computer vision library OpenCV is utilized to establish an image stream between the camera and the object detection module. It provides a generic ROS camera driver that can be utilized with all Video4Linux (V4L2) compatible cameras and publishes the image stream to the ROS-topic `/cv_camera/image_raw`. In addition, Slamtec, the manufacturer of the RPLidar, provides a ROS driver for their hardware. The driver publishes the measured ranges to the topic `/scan`.

5.2 Calibration

The first step in fusing measurements from the perception sensors is obtaining precise knowledge of the relative pose between each sensor. This is an extrinsic calibration problem with the goal of achieving a relative transformation between two coordinate frames \mathcal{C} and \mathcal{L} . Then, when a common frame of reference is achieved, one can associate the data from each sensor for whatever application one wishes.

5.2.1 Camera calibration

For the camera to associate real-world points with pixel values in an image, we must establish the intrinsic calibration and distortion parameters outlined in Section 2.3. Zhang [2000] proposes a method for intrinsic camera calibration by using a checkerboard pattern. In short, the method requires the camera to observe a checkerboard pattern shown at several different orientations. The camera or the pattern can be freely moved, and the motion need not be known. Then the intrinsic camera parameters and radial lens distortion are modeled using a procedure consisting of a closed-form solution, followed by a nonlinear refinement based on the maximum likelihood criterion.

OpenCV provides a ROS-compatible implementation of the procedure utilized in this thesis. Executing the calibration is straightforward. The calibration node subscribes to the ROS topic `/cv_camera/image_raw` and needs to be provided with some information regarding the checkerboard. The required input includes the number of inner corners in the pattern and the length of each square in millimeters. The board used in this thesis

was 9x7 squares, meaning 8x6 inner corners, each with a length of 40 mm. After the node launches, the checkerboard is moved around in the camera frame until the program is provided with enough samples. Then the calibration routine is run by pressing the calibrate button. This yielded the following intrinsic parameters for the camera:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 580.01081311 & 0.00000000 & 285.24692299 \\ 0.00000000 & 578.3171528 & 230.85573207 \\ 0.00000000 & 0.00000000 & 1.00000000 \end{bmatrix} \quad (5.1)$$

and the distortion parameters:

$$\begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix} = \begin{bmatrix} -0.46695407 & 0.18277399 & -0.00357946 & 0.01039586 \end{bmatrix} \quad (5.2)$$

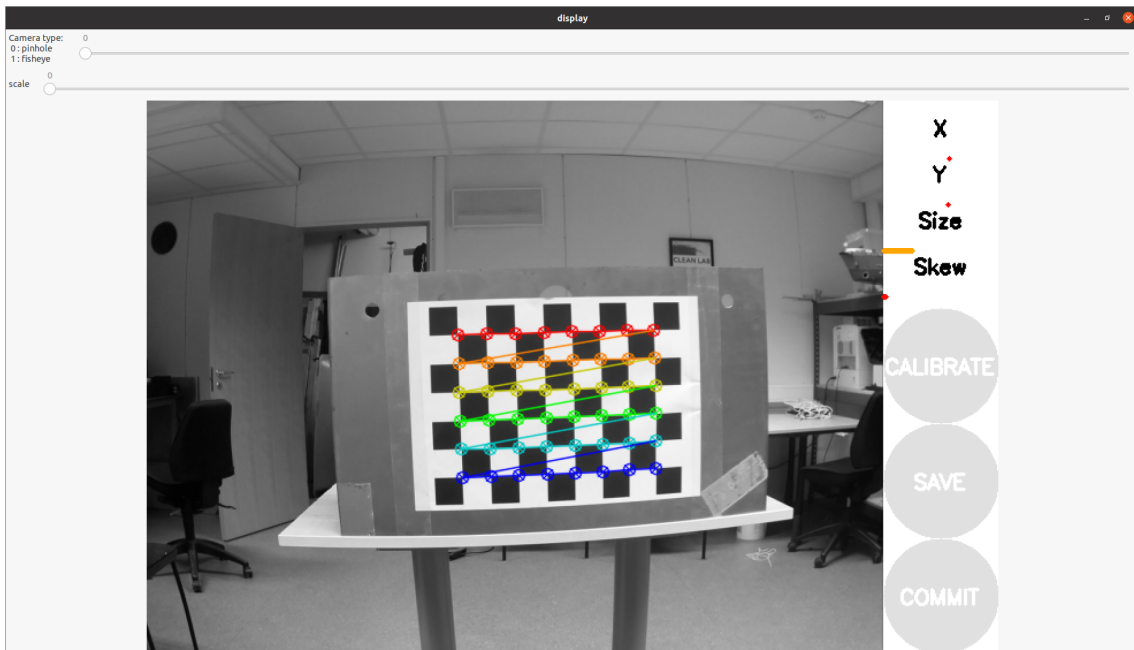


Figure 5.3: Camera calibration process

These parameters are saved to the workspace and reused every time the camera driver is launched. If changes are made to the resolution, focus, or camera calibration, it is good practice to re-calibrate the camera.

5.2.2 Lidar-Camera Calibration

The next step is to determine the relative pose between the camera and the lidar. In essence, this means associating the pixels in an image with laser points in the plane. Thus we must determine a transformation from the lidar coordinate system to the camera coordinate system. Figure 5.2 displays the configuration of the sensors and their respective coordinate systems. Any transformation between two euclidean coordinate frames can be expressed via a rotation and a translation. Thus, *six* extrinsic parameters must be established as part of the camera-lidar calibration.

Software for calibrating the camera and lidar is available on Github [ehong-tl, 2019]. Some updates were made to it, as parts of the code were deprecated and incompatible with the Noetic distribution of ROS. The main portion of the code and the methods utilized for calibrating is, however, the same. The software solves a Perspective-n-Point (PnP) problem [Gao et al., 2003]. The rigid body transformation is computed by a set of object points, along with their corresponding projection in an image and the camera’s intrinsic parameters and distortion coefficient. For this case, the object points will be the 2D lidar scan. In essence, the full camera model (2.22) is utilized to estimate the transformation. However, we substitute for lidar points instead of real-world coordinates.

The challenge with this method is knowing exactly where each point in the lidar scan lies in the image and real world. Especially when the scan is only 2D, spatial height is hard to determine. A solution is to use easily identifiable objects in the environment to make the task easier. One such example is a corner. Their geometry is easily distinguishable in an image and a laser point cloud. Figure 5.4 illustrates a part of the MC-lab used for calibration. It had three corners that could easily be related to the image, scan, and real world. To determine where lidar points were located along the z-axis, the height from the floor up to the lidar was measured. This method is, of course, not perfect, as the lidar will not necessarily be ideally oriented. Nevertheless, the procedure provided a satisfactory result when given enough data.

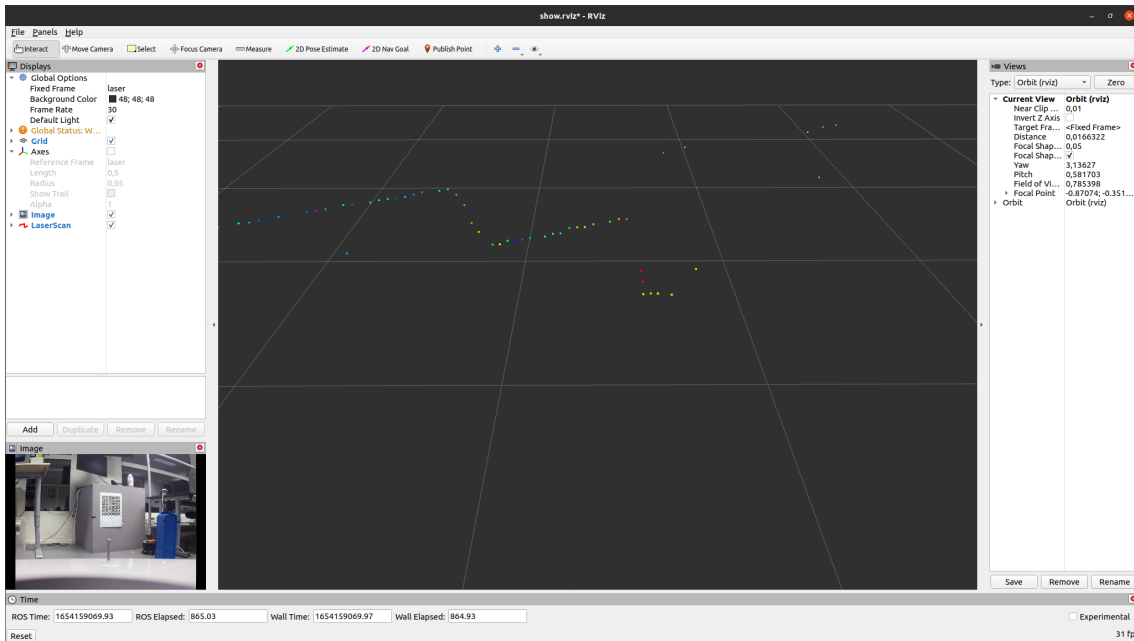


Figure 5.4: Camera view along with 2D point cloud from lidar, displayed in RViz

After the height was measured, the location of the scan was marked with tape in each corner, and then data was collected using the 2D Nav goal tool on a laser point in RViz. This prompts a new window to appear with the undistorted image from the camera. The user can then place the laser point in the image as illustrated in Figure 5.5. This would relate to the tape in the image. In total, this should be repeated until the result is satisfactory. The software provides a method for projecting the scan points onto the video stream in real-time, which can be used immediately to check the calibration. The method must be provided with a minimum of 4 data points to function.

The resulting rotation and translation between the camera and lidar were found to be.



Figure 5.5: Picking laser coordinate in image pixels

$$\mathbf{R}_l^c = \begin{bmatrix} 0.01926926 & -0.99888518 & 0.04309405 \\ 0.05734153 & 0.04413521 & 0.99737858 \\ -0.99816865 & -0.01674766 & 0.05812806 \end{bmatrix} \quad (5.3)$$

$$\mathbf{t}_l^c = \begin{bmatrix} 0.019954 & 0.035992 & 0.053483 \end{bmatrix}^\top \quad (5.4)$$

To verify the calibration, the placement of the lidar and camera was measured by hand. The difference in calibration and hand measurements was then found to be less than a centimeter in each translation. Of course, the hand measurements can not be considered ground truth but are a good indicator that the calibration is within reason.

5.3 Transformation to NED-frame

With a relative transformation between the camera and lidar established, the next step is to establish a transformation to the coordinate frame of the MC-Lab. This reference frame will be referred to as the basin frame. To achieve this, we must first transform the measurements into the vessel's local frame of reference, BODY. To simplify this transformation, the camera was mounted so that certain axes aligned with the body frame's axes. This would be z_c and x_b , x_c and y_b , and finally y_c and z_b , as illustrated in Figure 5.2. The rotation mapping between camera and BODY is then

$$\mathbf{R}_c^b = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \quad (5.5)$$

The translation can be measured by hand and expressed as the distance from the vessel CO to the sensor. In the last calibration performed this was:

$$\mathbf{t}_c^b = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^\top = \begin{bmatrix} 0.06 & 0.005 & -0.075 \end{bmatrix} \quad (5.6)$$

The corresponding transformation matrix becomes:

$$\mathbf{T}_b^c = \begin{bmatrix} \mathbf{R}_b^c & \mathbf{t}_b^c \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}. \quad (5.7)$$

The final step is the transformation from BODY to NED. There are no external forces in the operating environment of the MC-lab, so we can safely assume that rotations are considered small for the Saucer, such that movements in surge, sway, and yaw are not affected by the configuration in pitch and roll. Then the rotation between NED and body becomes

$$\mathbf{R}_b^n(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.8)$$

where ψ is vessel heading. The qualisys system is calibrated so that the measurement corresponds to the vessel CO. This means we can say that the translation from BODY to NED is $\mathbf{t}_b^n = [\hat{x}, \hat{y}, 0]^\top$, where \hat{x} and \hat{y} are the estimated positions of the vessel in North and East. During testing, the z -plane in the qualisys system was defined above the water plane. Technically, setting the translation in z to 0 is wrong, but considering only the xy plane is of interest, it has no consequence on the result and can be safely disregarded. Thus the transformation becomes:

$$\mathbf{T}_b^n = \begin{bmatrix} \mathbf{R}_b^n & \mathbf{t}_b^n \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}. \quad (5.9)$$

Finally, we can compound the transformation matrices derived for the transformations from lidar to NED-frame and camera to NED frame, respectively

$$\mathbf{T}_l^n = \mathbf{T}_n^b \mathbf{T}_b^c \mathbf{T}_c^l, \quad (5.10)$$

$$\mathbf{T}_b^n = \mathbf{T}_n^b \mathbf{T}_b^c. \quad (5.11)$$

5.4 Visual detection

The framework for detecting obstacles in images utilized in this thesis is based on the SSD Mobilescan V2 CNN. The decision to use a single shot detector was made due to the limitations on computational power. While the RPi 4b contains a decently powerful CPU for its size, it would not be able to handle real-time object detection at sufficient frames per second while simultaneously running the remaining nodes of the maneuvering control system. Additionally, none of the available remote operator computers had access to GPU computation and relied solely on the power of CPUs. Choosing a more computationally expensive network was therefore not an option. On the other hand, SSD Mobilescan V2 is known for its lean network and novel depthwise separable convolutions, making it a suitable network to deploy on low-competition devices such as an RPi, or a CPU-reliant laptop. Another reason for picking the Mobilescan is the availability of pre-trained networks and ready ROS-compatible implementations using Python or C++.

5.4.1 Implementation and training

The model used in this thesis is an open-source implementation available on Github [Zitewitz, 2018] and is trained on the Common Objects in Context (COCO) dataset. This is large-scale object detection, segmentation, and captioning dataset widely used for training CNNs. It contains over 90 different classes, including marine objects such as boats. As mentioned, the COCO dataset contains 90 different classes, some of which will be irrelevant to marine operations. Ideally, such classes should be removed from the model to avoid false detections, but that would require training a model entirely from scratch. This was not a priority, as it would be too time-consuming and outside the scope of this thesis. Access to GPU computation was also limited, and one could filter out all detections of unwanted classes in the sensor fusion module.

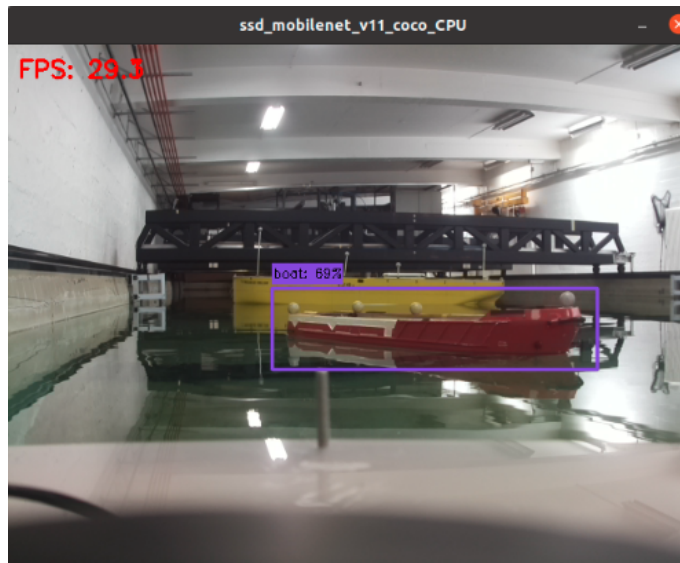


Figure 5.6: Visual detection of the CS Enterprise in the MC Lab

For this thesis, the camera recorded image at a resolution of 640×480 pixels at a frame rate of 30 frames-per-second (fps).

5.5 Lidar segmentation

The lidar used in this thesis can generate a point cloud with hundreds of points per scan. In this cloud, an obstacle can be the source of multiple points, as illustrated by Figure 5.7. This shows a laser scan of the MC Lab basin from the lidar reference frame. Three prominent clusters are visible, the walls of the basin on the sides and one in the middle corresponding to the CS Enterprise 1. As we wish to track only one measurement per obstacle, clustering techniques are performed on the point cloud data. This provides the SA-system with at most one range measurement per detected object.

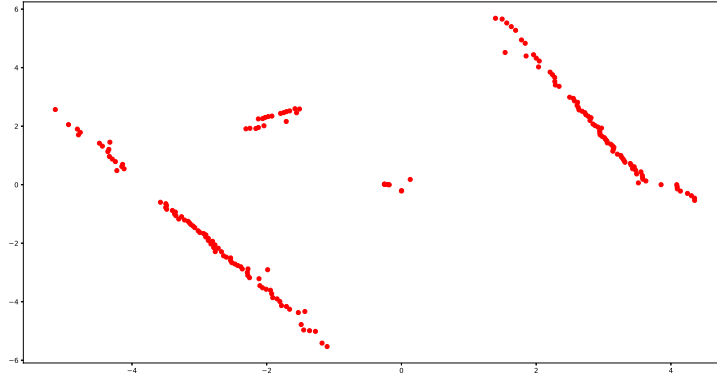


Figure 5.7: Laser scan sample of the MC lab basin

5.5.1 K-Means

To identify clusters, the K-Means algorithm will be utilized. K-Means is one of the most renowned clustering algorithms. Its simple and efficient nature makes it easy to adapt for different purposes. Originally, the principle was published by Steinhaus et al. [1956], but refined into an algorithm by Forgy [1965] and Lloyd [1982] independently of each other. Thus, it is also known under the moniker of 'Lloyd-Fordy Algorithm.'

The algorithm is considered a partitional clustering algorithm because it uses a predefined number of clusters to either maximize or minimize a predefined numerical criterion [Everitt et al., 2011]. The algorithm outputs a centroid for each cluster, which corresponds to the arithmetic mean point of a cluster. This is what gives the *means* part of the algorithm name. The *k* part of the name refers to the predefined number of clusters given to the algorithm. The algorithm steps are described below.

Algorithm 1 K-Means clustering algorithm, adapted from Singh et al. [2013]

- 1: Assign number of clusters k . In this thesis, this is determined by the number of detections in the camera field of view.
- 2: Choose initial centroids v_k . Either assigned randomly, given as an input or assigned by some other means.
- 3: Compute the distance between each data point and cluster centers using the Euclidean distance metric as follows

$$d_k = \sqrt{\sum_{k=1}^m (x_k + y_k)^2} \quad (5.12)$$

- 4: Assign data point to cluster center according with the minimum distance to point
- 5: Compute new cluster centroids by using the arithmetic mean

$$v_i = \left(\frac{1}{C_i}\right) \sum_1^{C_i} x_i \quad (5.13)$$

where C_i denotes the the number of data point in each cluster.

- 6: Recompute the distance between each data point and cluster centroid
 - 7: If no new data point is assigned to a cluster then stop, else repeat steps 4 to 6.
-

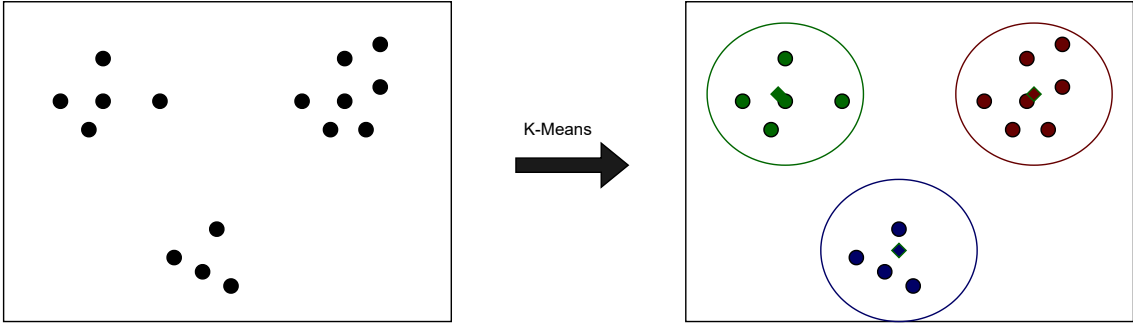


Figure 5.8: K-means principle

5.6 Sensor fusion

The approach to sensor fusion is adapted from Kim [2018], whom provides an open source solution to real-time human detection using a monocular camera and a 2D lidar. The code provided by Kim [2018] serves as the base for the fusion module, but is adapted to the needs of this thesis. The core approach remains much the same, but the code has been significantly updated to suit the needs of this thesis. A flowchart over the sensor fusion approach is presented in Figure 5.9

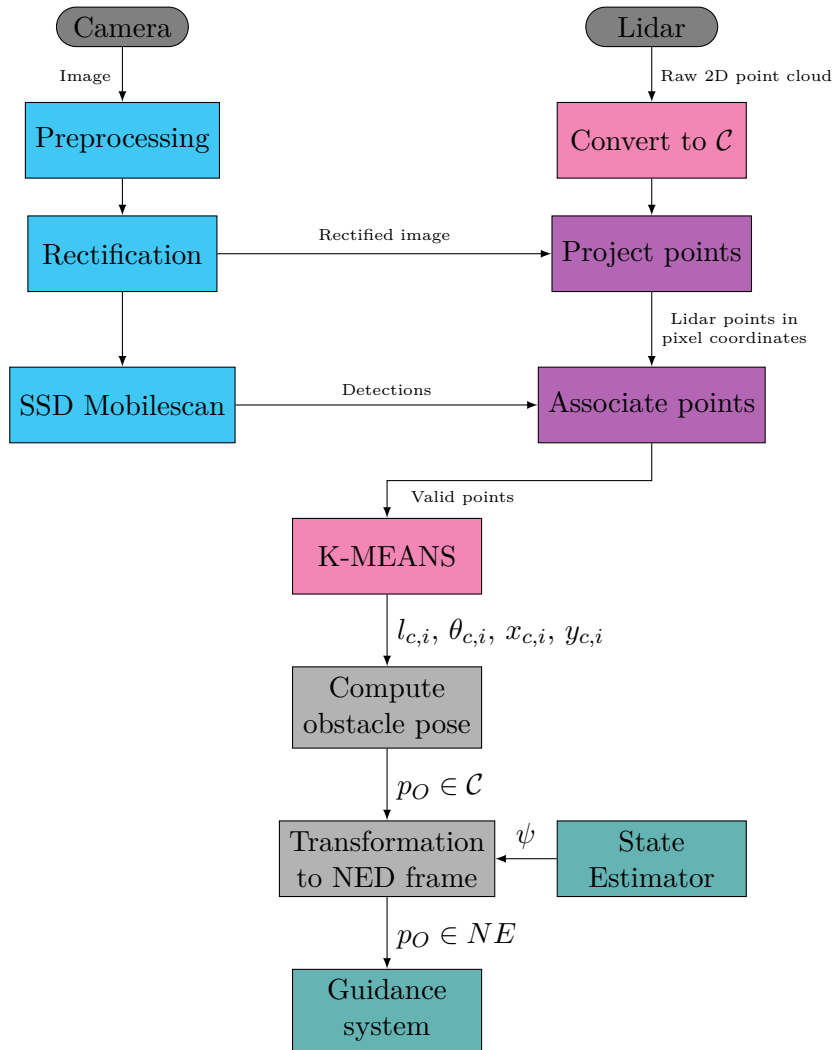


Figure 5.9: Situational awareness framework

The fusion module starts with a processing pipeline for each sensor. The image is first processed and rectified to remove distortions from the camera lens before the image is passed through the object classifier. Here all object detections are sorted and discarded if not labeled with the class 'boat.' Then, the bounding box coordinates of the detected obstacles are passed to merge with the lidar measurements.

Meanwhile, the lidar's processing pipeline converts the range scans to point-cloud coordinates. Then, the point cloud is converted to the camera frame of reference using the transformations derived in Section 5.2.2. The points within the camera's field of view are identified and projected onto the rectified image by solving the same PnP problem as in the calibration for the lidar points. The pixel coordinates of the lidar measurements are then associated with any object detection. If the pixel coordinate lies within the area surrounded by a bounding box, the corresponding real-world point cloud measurement in \mathcal{L} is added to a list of valid points. This list is then passed on to the K-means algorithm.

The K-means clustering algorithm sorts the valid points into clusters and identifies the centroid of each cluster. The number of clusters will correspond to the number of detections in each frame. Next, the distance to the centroid is computed along with the relative angle between the vessel and the obstacle. Remembering that the distance and angle are still

in the lidar frame of reference, one can use (2.5) to find the obstacle pose. This is then transformed to the NED-frame via the transformations derived in 5.3. Finally, the obstacle poses are fed to the guidance module. Figure 5.10 displays what a successful detection looks like after fusing measurements.

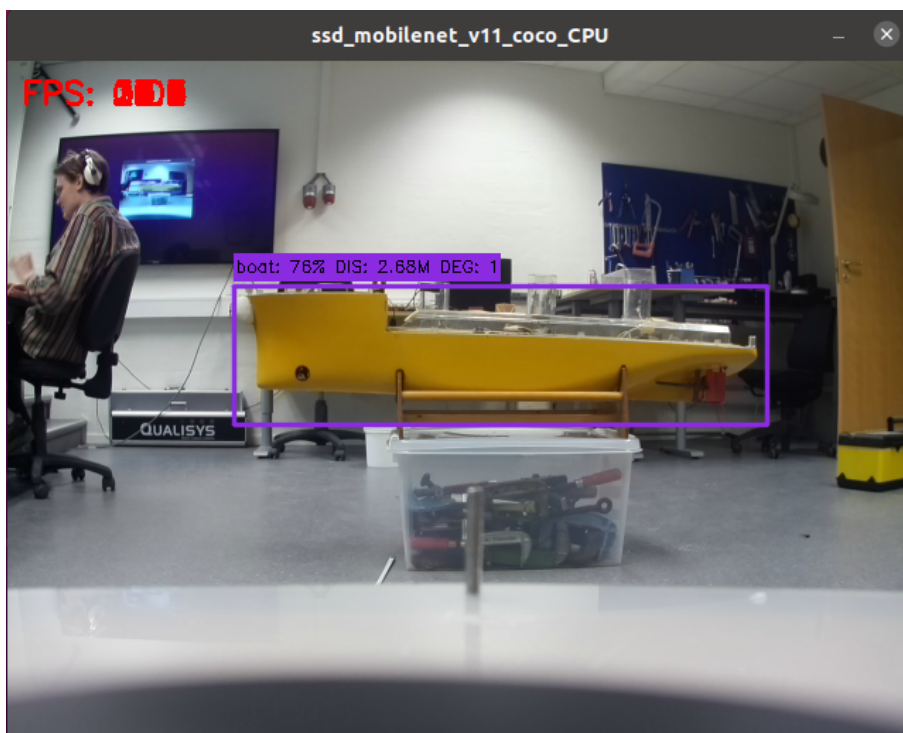


Figure 5.10: Object detection after fusing image and lidar measurements, using Cybership II as the target.

State estimation

This chapter presents the vessel state estimator, or observer, implemented for the CSS. The observer's primary function is to reconstruct the vessel states from partial measurements. In this thesis, only positional measurements $\boldsymbol{\eta}$ are available from the MC-lab Qualisys Motion Capture system. However, the controller used in this thesis requires the vessel's velocity states and an unknown bias term to compensate for the unmodeled dynamics present in the system. A second function of the observer is also to filter out any noise in the measurements. To this end, a simple 'DP observer' is implemented.

6.1 Observer design

The observer is a version of the nonlinear passive observer proposed by Fossen [2021]. In addition, Værnø et al. [2017] outlines the design and proof for a variant that relies solely on positional measurements.

First, model the low-speed dynamics of the CS Saucer:

$$\dot{\boldsymbol{\eta}} = \mathbf{R}(\psi)\boldsymbol{\nu} \tag{6.1a}$$

$$\mathbf{M}\dot{\boldsymbol{\nu}} = -\mathbf{D}\boldsymbol{\nu} + \mathbf{R}(\psi)^\top \mathbf{b} + \boldsymbol{\tau} \tag{6.1b}$$

$$\dot{\mathbf{b}} = 0 \tag{6.1c}$$

$$\mathbf{y} = \boldsymbol{\eta} \tag{6.1d}$$

Here, $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ are generalized vectors for position and velocity used to describe the vessel's motion in three degrees of freedom. $\boldsymbol{\tau}$ represents the generalized forces and moments generated by the actuators, and the bias \mathbf{b} accounts for immeasurable dynamics and uncertainties. $\boldsymbol{\eta}$ and b are given in the NED reference frame, while $\boldsymbol{\nu}$ and $\boldsymbol{\tau}$ are given in the localized body frame.

Next, the estimation errors are defined as $\bar{\boldsymbol{\eta}} := \boldsymbol{\eta} - \hat{\boldsymbol{\eta}}$, $\bar{\boldsymbol{\nu}} := \boldsymbol{\nu} - \hat{\boldsymbol{\nu}}$ and $\bar{b} := b - \hat{b}$. These are inserted these (6.1a) together with the injection gains $\mathbf{L}_i, i \in \{1, 2, 3\}$. This gives dynamics for the observer algorithm

$$\dot{\hat{\boldsymbol{\eta}}} = \mathbf{R}(\psi)\hat{\boldsymbol{\nu}} + \mathbf{L}_1\bar{\boldsymbol{\eta}} \quad (6.2)$$

$$\mathbf{M}\dot{\hat{\boldsymbol{\nu}}} = -\mathbf{D}\hat{\boldsymbol{\nu}} + \mathbf{R}(\psi)^\top\hat{\mathbf{b}} + \mathbf{R}(\psi)^\top\mathbf{L}_2\bar{\boldsymbol{\eta}} + \boldsymbol{\tau} \quad (6.3)$$

$$\dot{\hat{\mathbf{b}}} = \mathbf{L}_3\bar{\boldsymbol{\eta}} \quad (6.4)$$

Here the injection gains \mathbf{L}_1 , \mathbf{L}_2 , and \mathbf{L}_3 are diagonal positive definite matrices. The gains were tuned using a simulator, and later in physical experiments in the MC-lab. Through this process, the following observer gains are determined:

$$\mathbf{L}_1 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{L}_2 = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \quad \mathbf{L}_3 = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.15 \end{bmatrix} \quad (6.5)$$

Guidance system

This chapter presents the proposed guidance system that aims to solve guidance problem from Section 4.3.

7.1 Path parametrization

This thesis implements a guidance system based on a parametrization using two path parameters. It follows the design of Marley [2021] and Skjetne [2021] as well as the implementations of Åsheim [2021] and Moen [2021]. As a baseline, the function is provided two waypoints, an initial point \mathbf{p}_0 and a terminal point \mathbf{p}_1 , by a supervisory module. Next the two path parameters $s_1 \in [0, 1]$ and s_2 are used to continuously parameterize the desired path

$$\mathbf{p}_d(s) := \mathbf{p}_0 + L(s_1 \mathbf{T} + s_2 \mathbf{N}), \tag{7.1}$$

where $L = |\mathbf{p}_1 - \mathbf{p}_0|$ is the distance between the waypoints, \mathbf{T} is the unit tangent vector along a straight line between the points and \mathbf{N} is the normal unit vector defined as

$$\mathbf{T} := \frac{\mathbf{p}_1 - \mathbf{p}_0}{|\mathbf{p}_1 - \mathbf{p}_0|}, \quad \mathbf{N} := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \frac{\mathbf{p}_1 - \mathbf{p}_0}{|\mathbf{p}_1 - \mathbf{p}_0|}, \tag{7.2}$$

respectively. In this parametrization, s_1 determines the position along a straight-line path, while s_2 determines the deviation from the nominal path in the normal direction of the straight line.

The control law used in this thesis requires the derivatives of the path as well, so we differentiate (7.1) with respect to \mathbf{s} , giving

$$\mathbf{p}_d^s = L \begin{bmatrix} \mathbf{T} & \mathbf{N} \end{bmatrix}. \tag{7.3}$$

The second derivative is simply $\mathbf{p}_d^{s^2} = \mathbf{0}_{2 \times 2}$

7.2 Speed assignment

To solve the dynamic task of the maneuvering problem, we define the speed assignments for each of our path variables. For the nominal straight-line path, s_1 , we let the desired speed along the path $u_d : \mathbb{R}_{\geq} \times \mathbb{R} \rightarrow \mathbb{R}$ be given as an input from the operator. Then the speed assignment is given by

$$v_s(t, s_1, s_2) = \frac{u_d(t)}{|p_d^{s_1}(s_1)|} \quad (7.4)$$

We also define a unit-tangent gradient update law for the path speed, only acting in positional space. We choose:

$$w = -\frac{\mu}{|p_d^{s_1}(s_1)| + \epsilon} \rho_1(p, s_1) \quad (7.5)$$

where $\rho_1(p, s_1) = -p_d^{s_1 \top}(p - p_d)$, ϵ is a small number to avoid division by zero and μ is a gain set by the operator. This gives the path speed of

$$\dot{s}_1 = \frac{u_d(t)}{|p_d^{s_1}(s_1)|} - \frac{\mu}{|p_d^{s_1}(s_1)| + \epsilon} \rho_1(p, s_1) \quad (7.6)$$

For the second path variable, we choose

$$\dot{s}_2 = k \tanh \frac{s_2}{\lambda_{\dot{s}_2}} \quad (7.7)$$

This ensures that $\dot{s}_2 = 0$ when $s_2 = 0$, and that the path speed converges on zero again after collision avoidance. k and $\lambda_{\dot{s}_2}$ are gains that determine the rate at which \dot{s} decreases and the slope of s_2 , respectfully.

7.3 Control Barrier Function

A CBF is implemented to check if the assigned path derivatives $\dot{\mathbf{s}}$ are safe. If the function determines that the path is not safe, $\dot{\mathbf{s}}$ is updated, so safety is achieved. Let the CBF be defined as

$$B(\mathbf{s}) := |\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o| - r_o, \quad (7.8)$$

with the derivative

$$\dot{B}(\mathbf{s}) = \frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}) \dot{\mathbf{s}}. \quad (7.9)$$

Here \mathbf{p}_o is the position of the obstacle in NED, and r_o the radius of an obstacle region that encapsulates the unsafe domain. When the vessel is outside of the obstacle region,

B is positive. If the vessel enters the region, B will become negative. To guarantee safe parameters,

$$\dot{B}(\mathbf{s}) \geq -\alpha(B(\mathbf{s})) \quad (7.10)$$

must be ensured. Here, α is an extended class- κ function chosen to be the same as in Marley [2021]

$$\alpha(B(\mathbf{s})) = \frac{1}{T_b} B(\mathbf{s}), \quad (7.11)$$

with $T_b > 0$ being a time constant unique to the vessel properties. The constraint decreases B when the vessel is far away from an obstacle region while forcing the derivative $\dot{B}(s)$ to be non-negative when the vessel is close to the obstacle region.

If the CBF deems the current \dot{s} unsafe, a new pair of derivatives is found by minimizing the deviation from the desired safe path derivatives. This is an optimization problem and can be expressed as a quadratic programming (QP) problem. To this end, the objective vector $\mathbf{x} := \dot{s}_s - \dot{s}_d$ is defined as the deviation between safe and desired path derivatives. This gives the QP problem

$$\min_{\mathbf{x}} \frac{1}{2} \mathbf{x}^\top \mathbf{Q} \mathbf{x} \quad (7.12)$$

$$\text{s.t. } \mathbf{A} \mathbf{x} \leq \mathbf{b}. \quad (7.13)$$

The constraint for the QP was previously expressed in (7.10). \mathbf{A} and \mathbf{b} are found by substituting (7.9) and (7.11) into (7.10).

$$-\frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}) \dot{s} \leq \frac{1}{T_b} B(\mathbf{s}) \quad (7.14)$$

$$-\frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}) (\dot{s}_s - \dot{s}_d) \leq \frac{1}{T_b} B(\mathbf{s}) + \frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}) \dot{s}_d \quad (7.15)$$

$$, \quad (7.16)$$

gives,

$$\mathbf{A} = -\frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}), \quad \mathbf{b} = \frac{1}{T_b} B(\mathbf{s}) + \frac{(\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o)^\top}{|\mathbf{p}_d(\mathbf{s}) - \mathbf{p}_o|} \mathbf{p}_d^s(\mathbf{s}) \dot{s}_d. \quad (7.17)$$

The safe path can then be computed as

$$\dot{s}_s = \mathbf{x} + \dot{s}_d. \quad (7.18)$$

This guarantees that s_2 is activated when collision avoidance with obstacles is necessary.

7.4 Heading reference

As the desired heading is decoupled from positional guidance in this system, the references can be assigned arbitrarily according to operational needs. In this thesis, it is set to be tangential to the path and can be expressed as the sum of two terms

$$\psi_d = \psi_T + \sigma_\psi \psi_N \quad (7.19)$$

where ψ_T is the heading tangential to the straight-line path, ψ_N is a heading correction corresponding to the change of heading due to deviation from the straight line, and $\sigma_\psi : \mathbb{R}_{\geq 0} \rightarrow \{0, 1\}$ is an activation function. σ_ψ is active until the vessel gets close to the terminal waypoint p_2 , where it is deactivated, causing the heading to only be tangential to the straight line. The threshold is set according to

$$\sigma_\psi = \begin{cases} 1, & \text{if } s_1 \leq 0.95 \\ 0, & \text{if } s_1 > 0.95 \end{cases} \quad (7.20)$$

The heading terms are computed as:

$$\psi_T = \text{atan2}(y_d^{s_1}, x_d^{s_1}), \quad (7.21)$$

$$\psi_N = \text{atan2}\left(\frac{\dot{s}_2}{\mathbf{T}}, \dot{s}_1\right). \quad (7.22)$$

7.5 Mission management

Due to the size of the operational space in the MC Lab, this thesis will predominately operate with two waypoints. The initial waypoint will be considered as the vessel's position when the guidance module is activated. To decide the terminal waypoint, the operator specifies the desired length of the straight-line path. The mission management function in the guidance module then proceeds to compute the terminal waypoint's location based on the given length and the vessel's current heading. The final waypoint position can therefore be expressed

$$\mathbf{p}_1 = \left[l_p \cos(\psi) \quad l_p \sin \psi + p_0 \right]^\top \quad (7.23)$$

where l_p is the given path length, p_0 is the initial waypoint, and the ψ is the vessel heading at the activation time.

Control System

This chapter outlines the proposed control design for solving the maneuvering control problem presented

in Section 4.4. In the pre-project [Solheim, 2021] for this thesis, a nominal control design was implemented using cascade backstepping and a one-dimensional path variable. For this thesis the backstepping controller has been expanded to incorporate the two dimensional path variable $\mathbf{s} = [s_1, s_2]^\top$ presented in Chapter 7. A strict thrust allocation scheme is also presented as a solution to the instability in heading.

8.1 Cascade backstepping

As described, the nominal control mode is handled using a cascade backstepping design. This controller takes the state estimations $\hat{\boldsymbol{\eta}}, \hat{\boldsymbol{\nu}}$ and $\hat{\mathbf{b}}$ provided by the observer, along with the a desired reference signals $\boldsymbol{\eta}_d$ and its relevant derivatives. The design follows that of Skjetne [2021] and the implementations of Moen [2021] and Åsheim [2021]. It divides the design into two subsystems: $\boldsymbol{\eta} \rightarrow \boldsymbol{\eta}_d$ and $\boldsymbol{\nu} \rightarrow \boldsymbol{\alpha}$. This corresponds to two design steps: a kinematic step and a kinetic. In this case, it is most convenient to complete the second step first [Skjetne, 2021].

8.1.1 Step 2: Kinetic design

Suppose $\boldsymbol{\alpha} \in \mathbb{R}^3$ is a virtual control for the kinematic states, where $\dot{\boldsymbol{\alpha}}$ is an available signal. We define the error state

$$\mathbf{z}_2 = \boldsymbol{\nu} - \boldsymbol{\alpha}. \quad (8.1)$$

The objective is then to control (8.1) exponentially to zero. We differentiate (8.1) and substitute the low-speed vessel model of the vessel presented in Section 3.5. This yields

$$\mathbf{M}\dot{\mathbf{z}}_2 = \mathbf{M}\dot{\mathbf{z}}_2 - \mathbf{M}\dot{\boldsymbol{\alpha}} \quad (8.2)$$

$$= -\mathbf{D}\boldsymbol{\nu} + \boldsymbol{\tau} + \mathbf{R}(\psi)^\top - \mathbf{M}\dot{\boldsymbol{\alpha}}. \quad (8.3)$$

Let the CLF candidate be:

$$V_2 = \frac{1}{2}\mathbf{z}_2^\top \mathbf{M}\mathbf{z}_2, \quad (8.4)$$

with the derivative

$$\dot{V}_2 = \frac{1}{2}\mathbf{z}_2^\top \mathbf{M}\dot{\mathbf{z}}_2 \quad (8.5)$$

$$= \frac{1}{2}\mathbf{z}_2^\top (-\mathbf{D}\boldsymbol{\nu} + \boldsymbol{\tau} + \mathbf{R}(\psi)^\top - \mathbf{M}\dot{\boldsymbol{\alpha}}). \quad (8.6)$$

Choosing the control law

$$\boldsymbol{\tau} = -\mathbf{K}_2\mathbf{z}_2 + \mathbf{D}\boldsymbol{\nu} + \mathbf{R}(\psi)^\top \hat{\mathbf{b}} + \mathbf{M}\dot{\boldsymbol{\alpha}}, \quad \mathbf{K}_2 = \mathbf{K}_2^\top > 0 \quad (8.7)$$

yields

$$\dot{V}_2 = \frac{1}{2}\mathbf{z}_2^\top (\mathbf{D} + \mathbf{K}_2)\dot{\mathbf{z}}_2 \leq 0 \quad (8.8)$$

and

$$\mathbf{M}\dot{\mathbf{z}}_2 = -\mathbf{D}\mathbf{z}_2 - \mathbf{K}_2\mathbf{z}_2. \quad (8.9)$$

Here, $\mathbf{K}_2 \in \mathbb{R}^3$ is the control gain. This renders the equilibrium $\mathbf{z}_2 = 0$ for the subsystem $\dot{\mathbf{z}}_2$ uniformly globally exponentially stable (UGES).

8.1.2 Step 1: Kinematic design

The task is now to design the virtual control $\boldsymbol{\alpha}$ such that the maneuvering objective is solved for the kinematic subsystem. To this end, we decouple surge-sway from yaw, letting

$$\boldsymbol{\nu} = \begin{bmatrix} v \\ r \end{bmatrix} \in \mathbb{R}^2 \times \mathbb{R}, \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_v \\ \alpha_r \end{bmatrix} \in \mathbb{R}^2 \times \mathbb{R}, \quad \mathbf{z}_2 = \begin{bmatrix} z_{2,v} \\ z_{2,r} \end{bmatrix} = \begin{bmatrix} v - \alpha_v \\ r - \alpha_r \end{bmatrix} \in \mathbb{R}^2 \times \mathbb{R} \quad (8.10)$$

be the velocity, virtual control law and error state of the kinetic subsystem, respectively. For the kinematic step, we solve the position subsystem as maneuvering problem and the heading as a tracking problem. We define the error states for position and heading:

$$\mathbf{z}_{1,p} := \mathbf{R}_2(\psi)^\top [\mathbf{p} - \mathbf{p}_d] \quad (8.11)$$

$$z_{1,\psi} := [\psi - \psi_d] \quad (8.12)$$

Starting with the position system, the error state is differentiated:

$$\dot{\mathbf{z}}_{1,p} = \dot{\mathbf{R}}_2(\psi)^\top [\mathbf{p} - \mathbf{p}_d] + \mathbf{R}_2(\psi)^\top [\mathbf{R}_2(\psi)\mathbf{v} - \mathbf{p}_d^{s_1}\dot{s}_1 - \mathbf{p}_d^{s_2}\dot{s}_2] \quad (8.13)$$

$$= -r\mathbf{S}_2\mathbf{z}_{1,p} + \mathbf{z}_{2,v} + \boldsymbol{\alpha}_p - \mathbf{R}_2(\psi)^\top \mathbf{p}_d^{s_1}(w_1 + v_{s_1}) - \mathbf{R}_2(\psi)^\top \mathbf{p}_d^{s_2}(w_2 + v_{s_2}), \quad (8.14)$$

where

$$\mathbf{S}_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \quad (8.15)$$

The CLF candidate for position is defined as

$$\mathbf{V}_{1,p} := \frac{1}{2}\mathbf{z}_{1,p}^\top \mathbf{z}_{1,p} \quad (8.16)$$

with the derivative

$$\dot{\mathbf{V}}_{1,p} = \frac{1}{2}\mathbf{z}_{1,p}^\top \dot{\mathbf{z}}_{1,p} \quad (8.17)$$

$$= \mathbf{z}_{1,p}^\top \left[-r\mathbf{S}_2\mathbf{z}_{1,p} + \mathbf{z}_{2,p} + \boldsymbol{\alpha}_v - \mathbf{R}_2(\psi)^\top (\mathbf{p}_d^{s_1}(w_1 + v_{s_1}) + \mathbf{p}_d^{s_2}(w_2 + v_{s_2})) \right] \quad (8.18)$$

Choosing the virtual control

$$\boldsymbol{\alpha}_v = -\mathbf{K}_{1,p}\mathbf{z}_{1,p} + \mathbf{R}_2(\psi)^\top \mathbf{p}_d^{s_1}(w_1 + v_{s_1}) - \mathbf{R}_2(\psi)^\top \mathbf{p}_d^{s_2}(w_2 + v_{s_2}), \quad (8.19)$$

yields

$$\dot{\mathbf{V}}_{1,p} = -\mathbf{z}_{1,p}^\top \mathbf{K}_{1,p}\mathbf{z}_{1,p} + \mathbf{z}_{1,p}^\top \mathbf{z}_{2,p} \quad (8.20)$$

where $\mathbf{K}_{1,p} \in \mathbb{R}^2$ is the control gain for position, and is chosen such that $\mathbf{K}_{1,p} = \mathbf{K}_{1,p}^\top > 0$.

Next, we consider the heading control. By assuming that derivative of the desired heading $\dot{\psi}_d$ is a continuously available signal; the control problem is solved with a tracking design. First, the heading error state is differentiated, giving

$$\dot{z}_{1,\psi} = \dot{\psi} - \dot{\psi}_d \quad (8.21)$$

$$= z_{2,\psi} + \alpha_\psi - \dot{\psi}_d. \quad (8.22)$$

Let the CLF candidate be

$$V_{1,\psi} = \frac{1}{2}z_{1,\psi}^2, \quad (8.23)$$

with the derivative

$$\dot{V}_{1,\psi} = z_{1,\psi} \left[z_{2,\psi} + \alpha_\psi - \dot{\psi}_d \right]. \quad (8.24)$$

Then, the virtual control

$$\alpha_\psi = -k_{1,\psi}z_{1,\psi}^2 - \dot{\psi}_d \quad (8.25)$$

renders

$$\dot{V}_{1,\psi} = -k_{1,\psi}z_{1,\psi}^2 + z_{1,\psi}z_{2,\psi}. \quad (8.26)$$

Here $k_{1,\psi} > 0$ is the heading gain. This parameter is tuned along with the $\mathbf{K}_{1,p}$ and \mathbf{K}_2 such that the system behavior performs adequately. The ultimate control gains for the tuned system are as follows:

$$k_{1,\psi} = 10, \quad \mathbf{K}_{1,p} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} 0.12 & 0 & 0 \\ 0 & 0.12 & 0 \\ 0 & 0 & 0.33 \end{bmatrix} \quad (8.27)$$

These were determined by tuning the system in simulations and later in the MC Lab.

8.2 Force limitations

As stability is such an important factor, a safety measure is implemented as a force saturation to avoid violent control inputs. Ueland [2016] proposed the following logic to ensure that the forces acting on the vessel are evenly saturated, and the saturation does not alter the orientation of the applied force vector:

$$F_{max} = 1 \text{ N}, \quad T_{max} = 0.3 \text{ Nm}, \quad c_k = \frac{F_{max}}{\sqrt{X^2 + Y^2}} \quad (8.28)$$

$$X_{sat} = \begin{cases} c_k X, & \text{if } c_k < 1 \\ X, & \text{if } c_k \geq 1 \end{cases}, \quad Y_{sat} = \begin{cases} c_k Y, & \text{if } c_k < 1 \\ Y, & \text{if } c_k \geq 1 \end{cases}, \quad (8.29)$$

$$N_{max} = \begin{cases} \text{sign}(N)T_{max}, & \text{if } |N| \geq T_{max} \\ N, & \text{if } |N| < T_{max} \end{cases} \quad (8.30)$$

where X, Y, N are the forces and moment in surge, sway, and yaw, respectively.

8.3 Thrust allocation

The CSS is equipped with three azimuth thrusters, distributed symmetrically in a circle with 120° spacing and a radius of $r = 0.138$ m to the vessel CO. The freely rotating azimuth thrusters can produce forces in any direction, making the vessel over-actuated. In previous experimental setups, the CSS has been utilized in a fixed thruster configuration Ueland [2016]. This thesis implements two allocation schemes, one fixed angle setup and another heading priority allocation scheme to combat the instability in heading.

A thrust allocation problem can be solved by examining the relationship between the generalized forces in BODY-frame and the actuator inputs, that is,

$$\boldsymbol{\tau} = \mathbf{B}(\boldsymbol{\alpha})\mathbf{u}, \quad (8.31)$$

where $\boldsymbol{\tau}$ is the thrust load vector, $\mathbf{u} \in \mathbb{R}^3$ is the control input, $\boldsymbol{\alpha} \in \mathbb{R}^3$ is the azimuth angle and $\mathbf{B}(\boldsymbol{\alpha})$ is the thruster configuration matrix describing the layout of the actuators. Each thruster's location can be expressed through a lever arm \mathbf{l}_i , a vector describing the thruster's position relative to the ship's CO in polar coordinates. For the CSS, the lever arms are

$$\mathbf{l}_i = \begin{bmatrix} l_{i,x} \\ l_{i,y} \end{bmatrix} = \begin{bmatrix} r \cos \beta_i \\ r \sin \beta_i \end{bmatrix}, \quad i = 1, 2, 3 \quad (8.32)$$

where $\beta = [0, \frac{2\pi}{3}, \frac{4\pi}{3}]^\top$ corresponds to the angular reference points of the given thrusters. The force produced by each thruster, u_i , can similarly be decomposed into x and y directions and expressed by

$$\mathbf{u}_i = \begin{bmatrix} u_{i,x} \\ u_{i,y} \end{bmatrix} = \begin{bmatrix} u_i \cos \alpha_i \\ u_i \sin \alpha_i \end{bmatrix} \quad (8.33)$$

Then, knowing that the corresponding thrust load for each thruster is expressed by Fossen, 2021

$$\boldsymbol{\tau}_i = \begin{bmatrix} u_i \\ \mathbf{l}_i \times \mathbf{u}_i \end{bmatrix} = \begin{bmatrix} u_{i,x} \\ u_{i,y} \\ l_{i,x}u_{i,y} - l_{i,y}u_{i,x} \end{bmatrix}, \quad (8.34)$$

and using the trigonometric relation of:

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta \quad (8.35)$$

the generalized thruster configuration matrix for the CSS becomes

$$\mathbf{B}(\boldsymbol{\alpha}) = \begin{bmatrix} \cos \alpha_1 & \cos \alpha_2 & \cos \alpha_3 \\ \sin \alpha_1 & \sin \alpha_2 & \sin \alpha_3 \\ r \sin \alpha_1 & r \sin \left(\alpha_2 + \frac{2\pi}{3} \right) & r \sin \left(\alpha_3 + \frac{4\pi}{3} \right) \end{bmatrix} \quad (8.36)$$

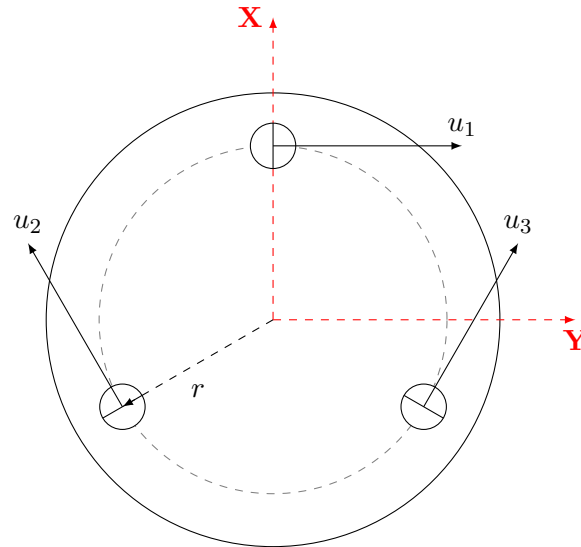


Figure 8.1: Fixed thruster configuration on the CS Saucer.

8.3.1 Fixed thruster allocation

Multiple fixed thruster configurations were tested as part of this thesis to find a configuration that provided a good force distribution while keeping the heading of the vessel stable. In the pre-project [Solheim, 2021], the same layout proposed by Ueland [2016] was utilized with the azimuth angles $\alpha_{fix} = [90^\circ, -30^\circ, 150^\circ]^\top$. For this thesis, the third thruster was rotated 180 degrees such that the azimuth angles become $\alpha_{fix} = [90^\circ, -30^\circ, 30^\circ]^\top$. This orientation makes the 'forward' direction more defined, which is important for the maneuvering objectives of this thesis. The configuration is illustrated in Section 8.3. Accordingly, the thruster configuration matrix is simplified to

$$\mathbf{B}(\alpha_{fix}) = \begin{bmatrix} 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ 1 & -\frac{1}{2} & \frac{1}{2} \\ r & r & -r \end{bmatrix} \quad (8.37)$$

The desired forces on each thruster can now be determined by solving (8.31) with respect \mathbf{u} , yielding

$$\mathbf{u} = \mathbf{B}^\dagger(\alpha_{fix})\boldsymbol{\tau} \quad (8.38)$$

where $\mathbf{B}^\dagger(\alpha_{fix})$ is the Moore-Penrose pseudoinverse of the thruster configuration matrix.

8.3.2 Thruster configurations

Throughout this thesis, several thruster configuration setups were utilized. These are outlined in Table 8.1 and can be viable for different operations. The main objective was to find a configuration that yielded a stable heading, so when tuning the controller and testing the configurations, the vessel was made to maneuver in an ellipsoid path. This type of path generation was developed as part of the pre-project Solheim, 2021 and chosen as a suitable test due to the way the heading is assigned. Similar to the straight-

line maneuvering developed for this thesis, the desired heading is set to be tangent to the path. The reference value is constantly changing for a circular path, making it an ideal challenge for the control system.

Thruster configurations			
	α_1 [deg]	α_2 [deg]	α_3 [deg]
Configuration 1	90	-30	30
Configuration 2	90	-30	-150
Configuration	90	0	0

Table 8.1: Thruster configuration

Results and Discussion

9.1 Testing scenarios

The physical experiments of this thesis were split into two portions. The first consisted of testing only the SA system to determine its accuracy and robustness. Then, after the SA system was validated, the complete control system was tested in several collision avoidance scenarios in the basin. This section will describe the details of each experiment.

9.1.1 Situational Awareness

The accuracy of the SA system was tested in two stages. The initial stage was done on land and consisted of placing the CS Saucer and Cyber ship II in the middle of the operator room of the lab. The goal was to determine any bugs and test the system's stability while stationary. The test compared the hand-measured distance between the two vessels with the estimated distance from the SA system. The measured distance between the two vessels was 2.17 m.

The second set of tests took place in the basin. The CSS was placed in the middle, facing the right wall when observing from the operator room. Then the CSE1 was placed approximately 1 m from the wall, with its bow facing the qualisys cameras. Cords with weights at the end were connected to the vessels to keep them from drifting substantially. Unlike the land test, the exact positional measurements of both vessels were accessible from Qualisys, so the accuracy of the reference transformation was also tested for this test. When the vessels were in position, the SA system on the CSS was activated, and a time series of detections were collected together with Qualisys measurements.

9.1.2 Collision avoidance

Finally, complete system tests were conducted when the controller was tuned and the SA system calibrated. These consisted of two different scenarios, both involving stationary obstacles. During all operations, obstacles were weighted down to avoid drift.

Scenario 1: Single stationary obstacle

For the first scenario, the CSS was manually maneuvered to the edge of Qualisys' range. Then the CSE1 was placed into the basin to interfere with the CSS's path. The length of the path was set to $l_p = 5$ m and the reference speed to $u_{ref} = 0.1$ m/s.

The CSE1 is about 1 m in length and 30 cm in width. To ensure that maneuvers were kept safe, the obstacle region was given a radius of $r_o = 1$ m.

Scenario 2: Multiple stationary obstacles

For the second scenario, a second stationary obstacle was introduced into the basin. The second obstacle took the form of Cyber ship II, which is depicted in Figure 9.1. Cyber ship II was placed further down in the basin and more to the right. For this test, the CSS was given a slightly longer path length of $l_p = 5.5$ m and the same reference speed as in Section 9.3.1.

Cyber ship II has similar dimensions to the CSE1, so the obstacle region was kept to the same radius.

9.2 Situational awareness

9.2.1 Display and performance

The final graphical interface of the sensor fusion module is displayed in Figure 9.1.



Figure 9.1: Detection of the Cybership II

It displays the detection box, the prediction score, the range to the computed cluster center, and the relative angle between the lidar and the centroid. In terms of performance, the average frame rate was 27.3 fps, which is more than enough for real-time purposes. However, there seems to be a slight delay in the image stream coming out of the detection module, which can affect operations. Regardless, considering all communications are done over the local wi-fi in the MC lab, a slight delay in streaming is expected. It is also not too detrimental to this thesis, given the limitations put on obstacles. Since all obstacles are assumed stationary, the position should be the same in every frame. All measurements in the SA module are also time-synchronized using the message filters provided by ROS. This matches the measurements if they have the same time-stamp. However, the delay would have had more significant consequences in a scenario with dynamic obstacles, especially on a small scale of the MC lab, where distances are covered in short periods. On a full-scale vessel, one could expect all computations to be done on the same onboard computer, so the delay caused by wi-fi would not be present.

9.2.2 Range test

Figure 9.2 displays the initial range test results. An average error of half a meter is immediately observable, meaning that the computed cluster centroid must be located behind the vessel.

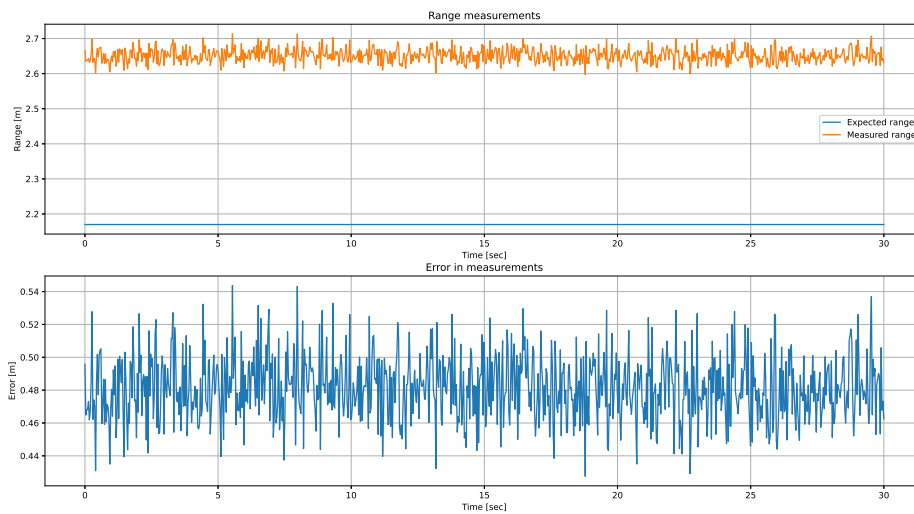


Figure 9.2: Initial range test

The cause of this can lie in the camera and lidar calibration and the way measurements are fused. First, an insufficient calibration means laser points are projected incorrectly onto the image. This results in points corresponding to the wall behind the vessel being projected onto the body. Consequently, the point would be falsely added to the list of valid laser points sent to the k-means algorithm. Since the k-means expects the number of clusters to be equal to the number of detections, all the points in the list will be considered part of a single cluster. Accordingly, the outlier points will influence the centroid placement to be further behind. This is clearly illustrated in Figure 9.4.

Another factor that could cause the same effect is the nature of bounding boxes. The

bounding boxes are constantly shifting their dimensions around the detected object. Naturally, due to the geometry of a traditional ship, some 'empty space' will be found inside the bounding box. Figure 9.1 illustrates this problem, as one can observe significant space between the edges of the box and the vessel's bow and stern. The validity of points is determined by whether their projection lies within the edges of the bounding box. Thus, it is likely that one or more points corresponding to the wall behind are associated with the detection. In the same manner, as a point being projected wrongly, this will cause the centroid to land behind the actual obstacle. In the first case, a poor calibration coupled with the blank space in the bounding box meant several points corresponding to the wall were considered valid. These incorrect projections caused the centroid placement to be almost half a meter beyond the expected point. The distance between the wall and the CSS was also above six meters, which is a large gap. A re-calibration of the camera and lidar was performed to better the performance. For the second test, the range between the two vessels was measured to 2.21 m.

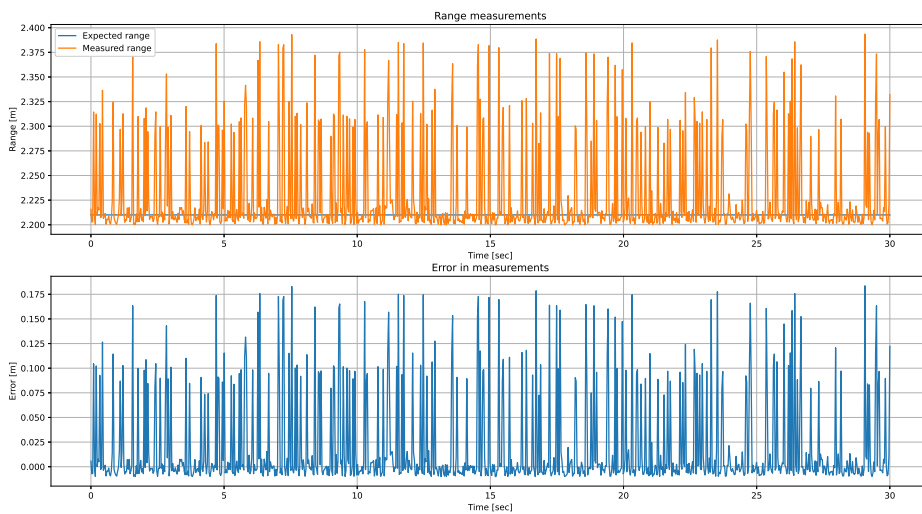


Figure 9.3: Range measurements after calibration

Figure 9.3 displays the range computation for the centroid after the second calibration. A significant improvement in accuracy can be observed, with the error in measurement being close to zero for several samples. The centroid is, however, still being pulled backward, which indicates that outlier points are influencing the measurements, albeit at a significantly lower rate compared to the previous results. In Figure 9.1, several wall points were constantly considered as part of the cluster, while for the second point, this happens occasionally. The centroid only being placed 18 cm behind indicates that fewer false points are projected. This result was deemed satisfactory, as available time in the MC lab was limited, and the accuracy was well within the margin of error. A point to note is that in a perfect measurement, all the lidar points are located along the vessel's hull. Accordingly, the centroid will most likely be located along the hull. The Cyber ship II and CSE1 have a width of about 25 cm, meaning that if the centroid position is 18 cm behind the vessel's outer hull, it will technically still be located on the vessel. Thus, the current calibration was deemed satisfactory for this thesis's purposes. For later operations, the sampling rate of lidar measurements was also reduced not to produce as noisy a measurement as in the land test.

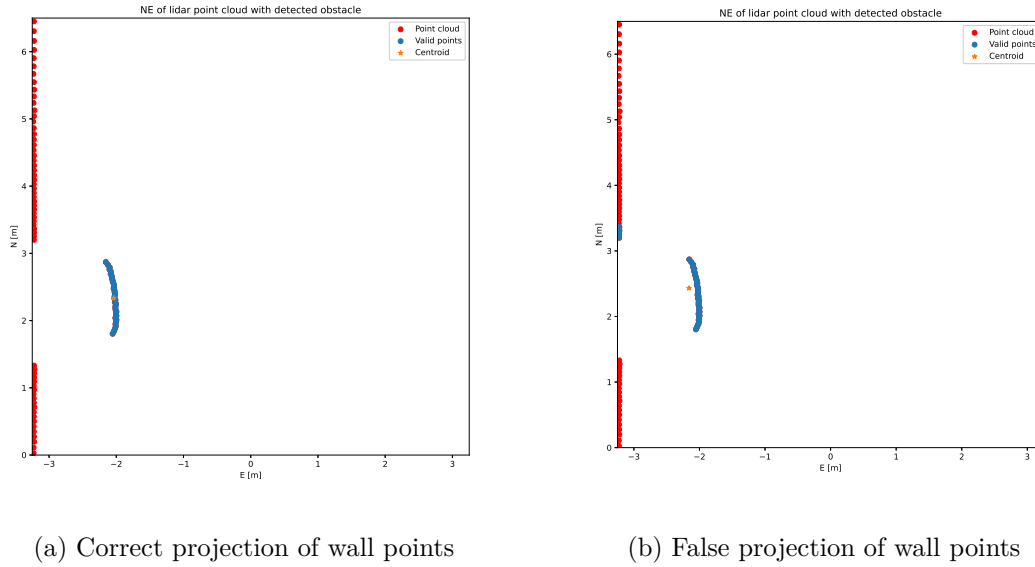


Figure 9.4: Influence of wall points on centroid placement

9.2.3 Basin test

Figure 9.5 displays a sample series of cluster centroids computed during the basin test. The black and red dashed squares are meant to illustrate the initial and terminal positions of the CSE1, measured by Qualisys. This illustrates the vessel drift, which constitutes a couple of centimeters in each direction. The same is done for the CSS, as illustrated by the gray and black circles. The orientation of the vessel and lidar is also illustrated by the arrows pointing from the vessel CO. Figure 9.6 illustrated the North and East position, along with the relative error in each direction.

As can be observed from both Figure 9.5 and Figure 9.6 estimation, the centroids are still pulled back towards the wall in most cases, indicating that outlier points are being projected onto the detection. However, the distance to the wall is much smaller than in the previous experiment, causing the outlier point to interfere less. Furthermore, most centroids are located inside the area encompassed by the CSE1, meaning that the results are more than sufficient, despite the influence of the wall points.

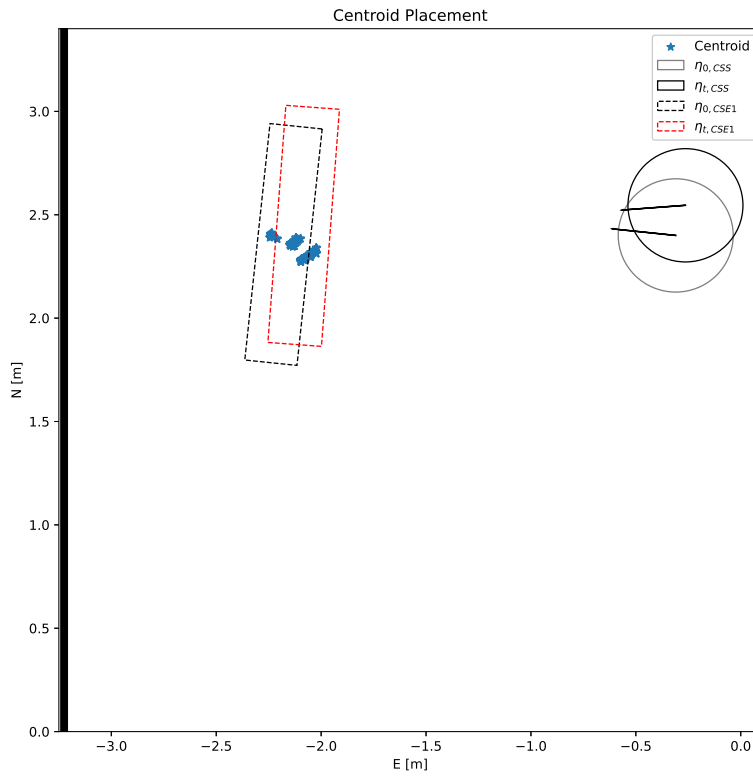


Figure 9.5: Estimated obstacle positions from object detection

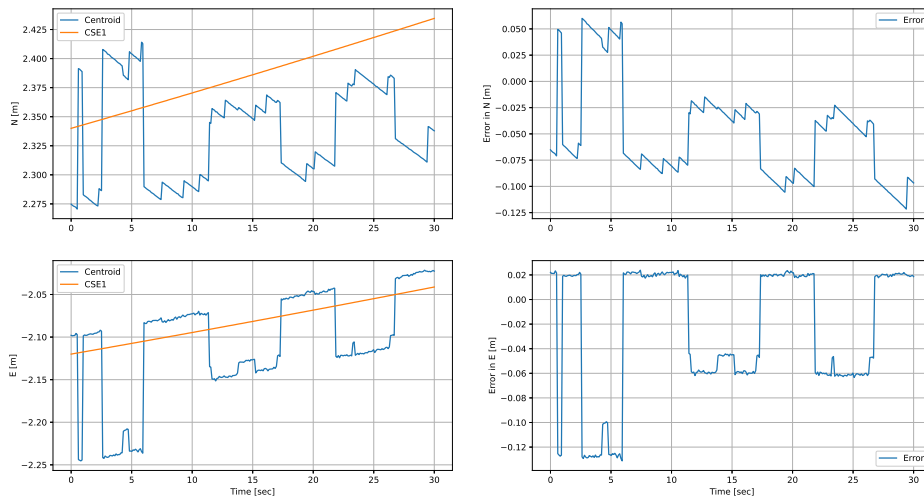


Figure 9.6: Centroid computations compared to qualisys measurements

The error in each direction is relatively small, but it is important to note that, ideally, the centroid should lie somewhere in the center of the hull. An example of this is illustrated

in Figure 9.4. The 'error' should have an absolute value of about 12 cm if you consider the target vessel's width.

9.3 Complete system tests

9.3.1 Scenario 1: Single stationary obstacle

The results for scenario 1 are shown in Figure 9.7, Figure 9.8, Figure 9.9 and Figure 9.10. First and foremost, one can observe from Figure 9.7 that the CSS successfully detects the obstacle and manages to maneuver from the initial point to the endpoint while avoiding a collision. The position of the detected obstacle was found to be $p_o = [0.51 \quad -0.40 \text{ big}]^T$, and was successfully detected immediately after the guidance module was initiated.

The vessel can follow the desired path to a certain degree but overshoots in both sway and heading when the evasive maneuver is performed. A delay can also be observed if one looks at Figure 9.8. The cause of this is how the guidance and control systems are activated. The Saucer is maneuvered into its start position using manual joystick control, with the guidance only being activated after the vessel is in position. The switch from manual to automatic control takes a little bit, causing the reference position to get a jump-start on the control system. It is possible that allowing the reference to large of a head-start on the controller can cause it to overshoot somewhat. A slight overshoot in heading is somewhat expected due to the geometry of the CSS. It is, however, much less than expected, meaning that the measures taken to stabilize the heading of the CS Saucer seem to be performing well. Figure 9.10 shows that the system largely relies on the two rear thrusters when maneuvering and only allocates forces to the front thruster when sharper turns are required. Considering the system is mainly moving forward, this seems reasonable.

The overshoot in sway is much more severe, as the vessel initially drifts in the wrong direction while turning, which causes the system to overcompensate and overshoot. However, the control system can correct itself after the evasive maneuver ends and reach the end position. Another factor that can contribute to the overshoot may also be the tuning of the controller. Given more time in the laboratory to run experiments, better tuning could likely be achieved. Unfortunately, time was limited, so the focus was put on performing the experiments rather than turning further. The system also overshoots the final way-point by some margin and cannot correct itself as the vessel drifts outside the range of the Qualisys system. Thus the time series is cut off when the vessel passes the endpoint.

The whole maneuver takes slightly around 75 seconds. The given reference speed was 0.1 m/S, so this seems reasonable if one takes the extra distance the evasive maneuver requires. As can be observed from Figure 9.8, the velocity in surge quickly accelerates towards the reference speed before the evasive maneuver is initiated. After the maneuver is executed, the speed converges on the reference again. The low reference speed also questions whether the saturation in force was necessary. As shown in Figure 9.9, the commanded forces are nowhere near the saturation point.

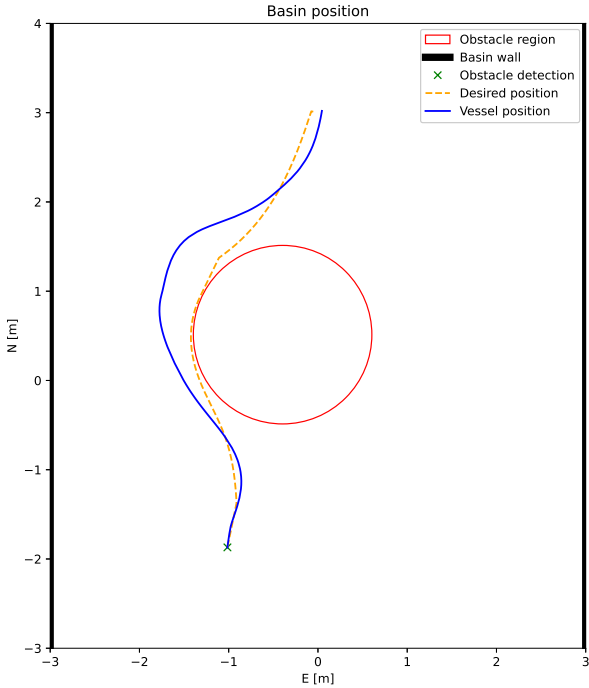


Figure 9.7: North-East plot for Scenario 1

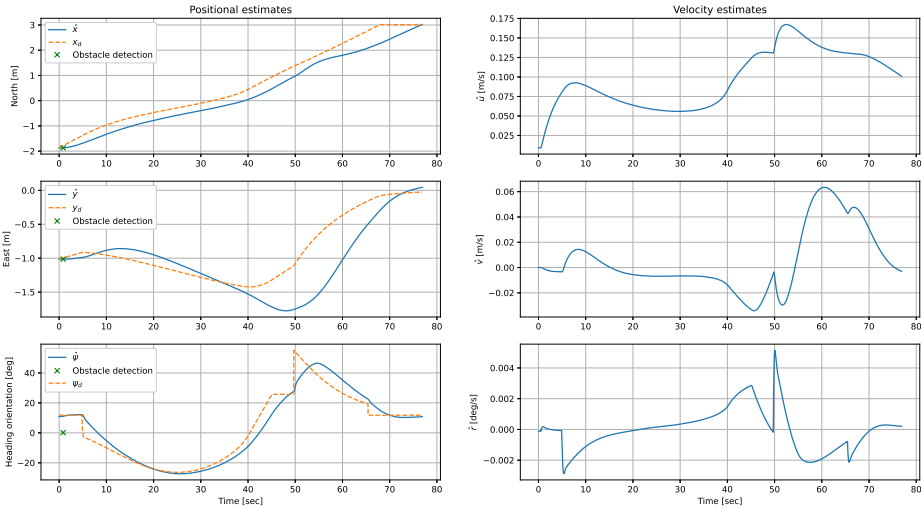


Figure 9.8: Position and velocity estimates for Scenario 1

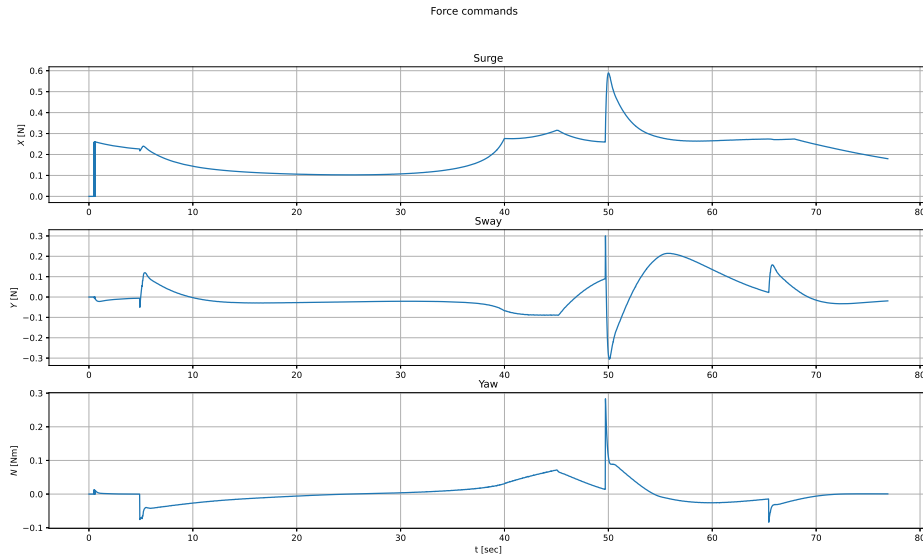


Figure 9.9: Force commands for Scenario 1

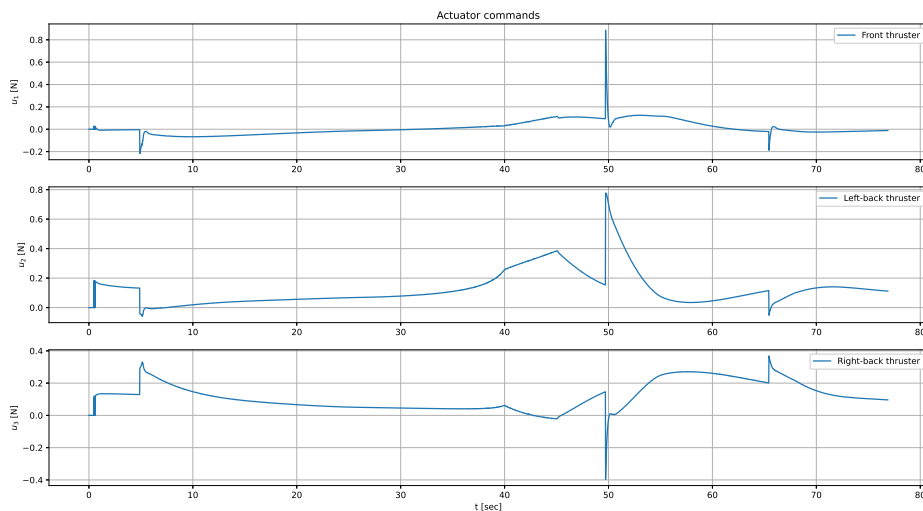


Figure 9.10: Actuator commands for Scenario 1

9.3.2 Scenario 2: Multiple stationary obstacles

The results for scenario 1 are shown in Figure 9.11, Figure 9.12, Figure 9.13 and Figure 9.14. They tend to follow many of the same patterns observed in Section 9.3.1. As is clear in Figure 9.11, the system can perform collision avoidance with two obstacles. The first obstacle is detected almost instantly, causing its associated barrier function to mark the path as unsafe early in the voyage. While the system performs the evasive maneuver for the first obstacle, the second ship is detected at $t = 24$ seconds. Fortunately, the second obstacle does not seem to interfere too much with the generated path, and the vessel can maneuver safely between the obstacles. The detection coordinates where

$p_{o,1} = [-0.53 \ 0.13]^\top$ and $p_{o,2} = [-1.53 \ 1.54]^\top$ for the first and second obstacle, respectively.

Similar to the first scenario, the vessel can follow the path to a certain degree, still overshooting in sway direction when performing the evasive maneuver. There is also an immediate sideslip when the controller initiates, caused by the vessel drifting in yaw between the activation of the guidance module and the controller module. The causes discussed in Section 9.3.1 likely apply in this scenario as well, as little to no tuning was performed between the scenario.

Another thing to note is that the vessel completes this maneuver quicker than the previous one, despite the length of the specified path being slightly longer. The reason for this is likely the initial conditions of the vessel and the evasive maneuver being longer for the first scenario.

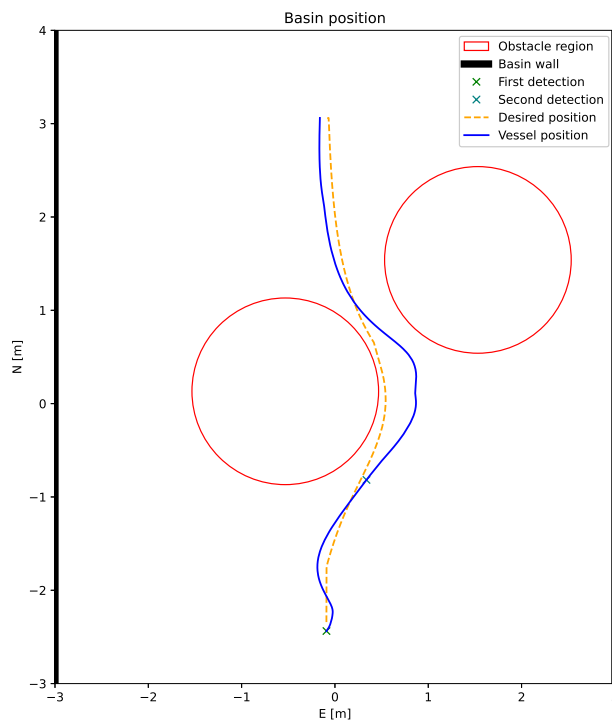


Figure 9.11: North-East plot for Scenario 2

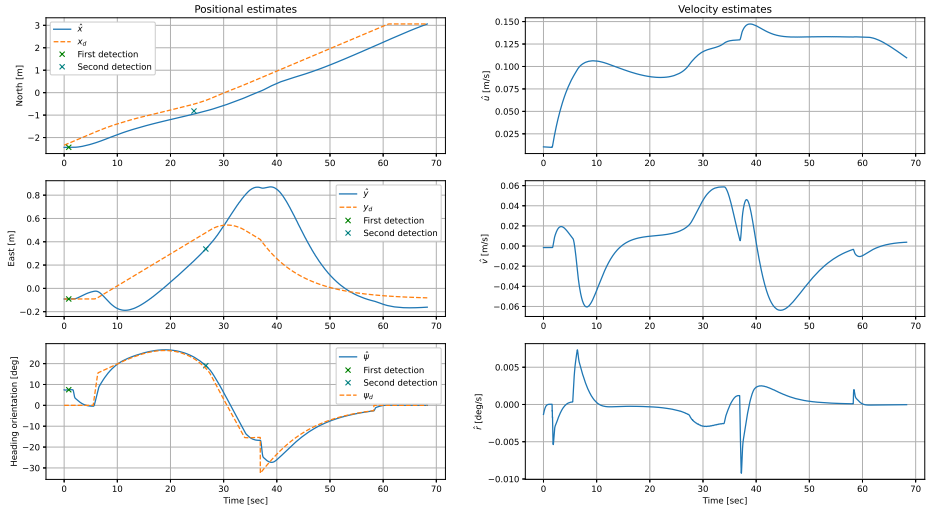


Figure 9.12: Position and velocity estimates for Scenario 2

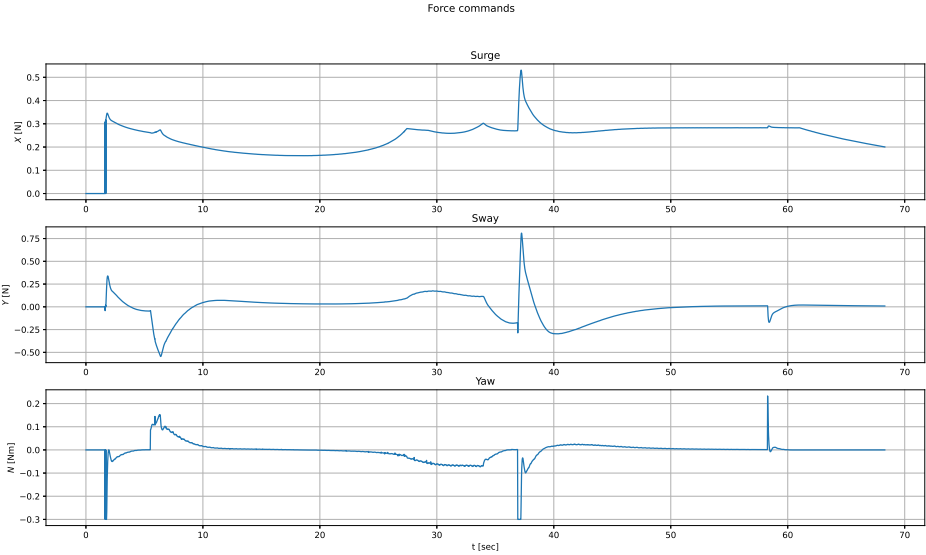


Figure 9.13: Force commands for Scenario 2

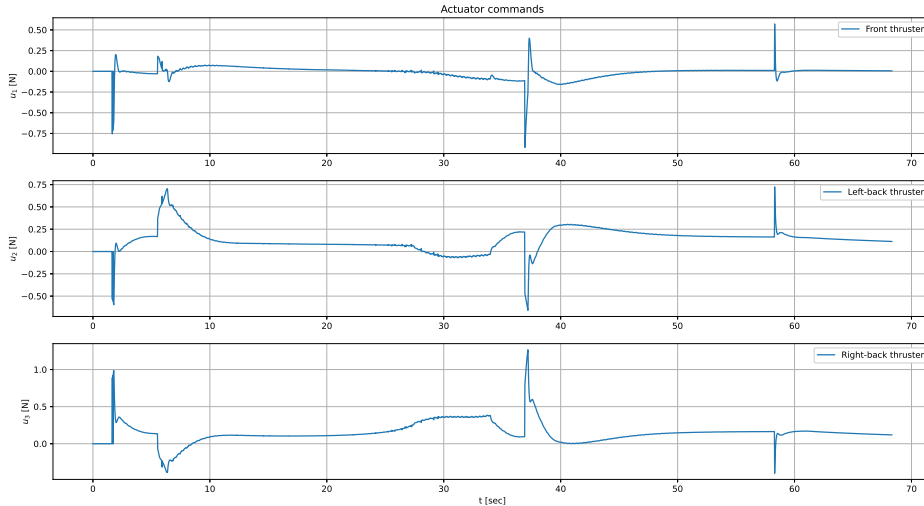


Figure 9.14: Actuator commands for Scenario 2

9.4 Final discussion

While the results presented so far show that the system is capable of object detection and subsequent evasive maneuvers, these are a sample of the best-performing experiments. At the end of experiments, the system failed one out of every three tries. Generally, the failures were caused by the object detection not consistently identifying the CSE1 and Cyber ship II as ships, meaning that they were not classified as obstacles. Typically, the detection failed when the vessels had a particular orientation. Examples are the bow or stern facing directly towards the camera, making the ship's sides harder to see. The detection would generally fail when the obstacles were oriented in such a way. Contrary, when the vessel was viewed from the side, the detection would usually succeed, albeit the certainty threshold for detections had to be lowered by 10% in the detection model to better the rate. If the SA system failed to detect the obstacles, the vessel would perform nominal straight-line maneuvering and enter the obstacle regions, resulting in collisions. Sometimes it would detect the first obstacle but not the second causing the same result. The visual detection range was also somewhat limited, with it rarely detecting anything more than 3 meters ahead. Factors in this are likely the low resolution the camera was recording and the lens utilized. As a performance of 30 fps is more than enough, more consideration should likely have been put into increasing the resolution to achieve better detections.

A second factor is likely that the CNN is not trained sufficiently to recognize the vessels in the MC Lab. As mentioned in Section 5.4, the model was primarily trained on generic data of full-scale ships operating at sea. While the training data from COCO is of high quality, it only presents vessels in an outside context. The MC-lab basin is an indoor pool surrounded by concrete walls and artificial lighting. Cyber ship II and CSE1 may also be more challenging to detect than conventional ships due to their smaller size and a greater number of modifications, i.e., Qualisys markers. Ideally, an additional custom dataset of the cyber fleet in the MC-lab should have been generated to provide better detection accuracy. This was outside the scope of this thesis, however.

The space between the two obstacle regions is also relatively tight. If the CSS's hull radius of 274 cm is taken into account, the vessel passes inside the obstacle region of the CSE1 in Figure 9.11. However, no crash was observed, as the radius for the obstacle region was large enough to provide enough buffer. This also brings into question the shape of the obstacle region. Ships tend to have an oblong geometry, causing the obstacle region to have much space next to the vessel's sides, while the distance from the stern or bow to the boundary is much smaller. Since detection points typically correspond to a point along the hull, the obstacle region may not encompass the entire obstacle. This means the radius for the obstacles had to be increased to ensure that the region encompassed the entire obstacle. A radius of one meter meant that for two obstacles, almost four meters of width would be invalid space. Considering the functional space in the tank was no more than 6 meters in width, the maximum number of obstacles one could use without causing problems was two. Anything more would typically cause overlapping in the regions or too small margins to pass through. An example is illustrated in Figure 9.15, where the vessel detected both obstacles failed the maneuver partly due to the overlapping regions.

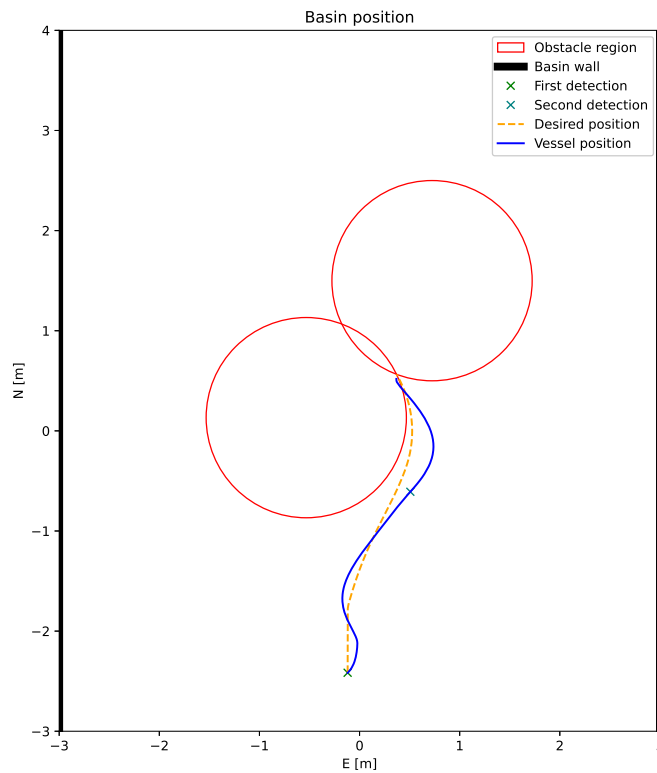


Figure 9.15: Failure in evasive maneuver due to overlapping obstacle regions

Chapter 10

Conclusion

This thesis presents the implementation and integration of a Situational Awareness system with control barrier functions for the autonomous surface vessel CS Saucer. To this end, the vessel was first extensively upgraded with a new embedded computer in the form of a Raspberry Pi 4b, qualisys markers, and a digital camera for computer vision purposes. The software architecture was also completely revamped, using ROS and Python as the framework for all hardware and control system drivers. The new vessel control system standardizes the cyber ship model in the MC-lab, providing more flexibility and continuity for students. This will likely be an academic advantage for future projects on the CSS.

The SA system is based on a 2D lidar scanner and a monocular camera. Methods for data reception and calibration of each sensor to give an accurate transformation between each coordinate system and the NED-frame have been presented and implemented. Additionally, a CNN for detecting ships was implemented along with the k-means clustering algorithm. Then a method for fusing the measurements was implemented by projecting lidar points onto the detection from the CNN. Valid points were sent to the clustering technique, which identified the cluster's center and the obstacle. A series of tests were planned, organized, and executed to test the SA system's robustness on land and in the MC-lab basin. The experimental results show that the SA system functions adequately and correctly estimate the detected vessels' position within a margin of error. The system does have some weaknesses, mainly its unreliable calibration method for the extrinsic parameters of the lidar and camera, which can cause significant inaccuracies due to incorrect point projections. The range of the detections is also quite limited, and the CNN fails to detect the obstacle ships in specific orientations. It was, however, sufficient for the scope of this thesis.

A complete maneuvering control system was also presented and implemented for the CSS. In addition to the situational awareness system, it consists of a guidance module, an observer, a controller module, and a thrust allocation module. The guidance module utilizes a two-parameter path parameterization that incorporates the use of a CBF to avoid detected obstacles detected by the SA system. The controller used in this thesis is a cascade backstepping controller that considers heading and positional control separately to stabilize the CSS heading. For this purpose, a fixed thruster allocation scheme was also implemented. The controller was first tested in a simulated environment and later in physical experiments. The experiments proved that the CBF architecture of the guidance module enabled the vessel to successfully generate safe paths around the obstacles that

the SA system detected and quantified.

Furthermore, the controller and thrust allocation can also be considered a success, as the vessel is capable of following the desired paths to a satisfying degree. Especially the heading is stable when following the reference values generated by the guidance module. However, it overshoots the desired positions, necessitating more system tuning.

10.1 Further work

Designing a complete control system for an ASV, including working situational awareness and maneuvering control, is no small feat. Throughout this thesis, several ideas for improvements or more sophisticated solutions were considered but not necessarily implemented due to time constraints. This section is meant to shed light on areas where improvements should be made if continued research is conducted using the CSS and its new control system.

First, the author believes that the current 2D lidar should be replaced with a more sophisticated 3D laser scanner. While working in a 2D plane initially seemed more straightforward and less complex, this thesis discovered that an extra dimension provides a more robust and accurate depiction of the environment in terms of computer vision and situational awareness. Furthermore, 3D lidars are considered state-of-the-art and are widely used in scientific research and industry projects. Thus ready, open-source software will be more available for 3D scanners. Additionally, extrinsic calibration between a 3D lidar and a monocular camera is more straightforward and robust, as the lidar point cloud can show spatial features in more detail, making it easier to relate pixel coordinates to laser points. Better feature extraction yields more accurate and robust calibrations. All of this will cascade into an overall more robust SA system.

To further improve the performance of the SA system, one could upgrade the operator computer to utilize GPU computation. For the current system, a modern state-of-the-art GPU will likely yield a frame rate in the hundreds. More computational power gives the user the room to increase the resolution of the captured images. In addition, it allows for more robust detection at further distances. *SSD Mobilenet* is also a relatively 'simple' CNN, and more computational power means it can be substituted for a more complex network such as *Faster R-CNN* to achieve better detections.

Another part of this thesis that can be improved upon is obstacles. The obstacles used in experiments have been restricted to stationary boats with a set size of the obstacle region. In a real-world scenario, it is reasonable to assume that some obstacles will be moving and more diverse than just a 'boat.' Therefore, for a system ready for real life, obstacles that move must be taken into account. The CNN should also consider more obstacles than just ships. Other relevant marine objects such as people, debris, rocks, or ice are examples of relevant classes for the model. This ties in with the previous points, as more computational power and better sensor readings, will allow for more complex tracking algorithms to be implemented. Ideally, a unique model should be trained from the ground up to work in the MC lab with custom labels and classes. The model should be trained on data accumulated inside the MC lab and generically available data like COCO.

If more dynamic obstacles were to be introduced, tracking procedures should also accompany them. Typically such tracking is done using Probability Density Association (PDA) filters or its derivatives integrated PDA (IPDA) and joint IPDA (JIPDA).

Finally, the system should be tuned to a more significant degree to counteract the displacement between the desired and actual positions in sway.

References

- Ames, Aaron D. et al. (2019). “Control Barrier Functions: Theory and Applications.” In: *2019 18th European Control Conference (ECC)*, pp. 3420–3431. DOI: 10.23919/ECC.2019.8796030.
- Åsheim, Nora (2021). “Autonomous ship maneuvering with guaranteed safety.” MA thesis. NTNU.
- Bishop, Christopher M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st ed. Springer. ISBN: 0387310738.
- Debeunne, César and Damien Vivet (2020). “A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping.” In: *Sensors* 20.7. ISSN: 1424-8220. DOI: 10.3390/s20072068. URL: <https://www.mdpi.com/1424-8220/20/7/2068>.
- ehong-tl (2019). URL: https://github.com/ehong-tl/camera_2d_lidar_calibration.
- Endsley, Mica R. (1988). “Design and Evaluation for Situation Awareness Enhancement.” In: *Proceedings of the Human Factors Society Annual Meeting* 32.2, pp. 97–101. DOI: 10.1177/154193128803200221. URL: <https://doi.org/10.1177/154193128803200221>.
- Everitt, Brian S et al. (2011). *Cluster Analysis, 5th Edition*. John Wiley and Sons Ltd.
- Ferguson, Max et al. (2017). “Automatic localization of casting defects with convolutional neural networks.” In: *2017 IEEE international conference on big data (big data)*. IEEE, pp. 1726–1735.
- Forgy, Edward W (1965). “Cluster analysis of multivariate data: efficiency versus interpretability of classifications.” In: *biometrics* 21, pp. 768–769.
- Fossen, Thor I. (2021). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley and Sons Ltd.
- Gao, Xiao-Shan et al. (2003). “Complete solution classification for the perspective-three-point problem.” In: *IEEE transactions on pattern analysis and machine intelligence* 25.8, pp. 930–943.
- Gu, Jiuxiang et al. (2018). “Recent advances in convolutional neural networks.” In: *Pattern Recognition* 77, pp. 354–377.
- Hamdan, Muhammad KA (2018). “VHDL auto-generation tool for optimized hardware acceleration of convolutional neural networks on FPGA (VGT).” PhD thesis. Iowa State University.
- Hartley, Richard and Andrew Zisserman (2015). *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press.
- HediVision (2021). *Pinhole Camera Model*. URL: <https://hedivision.github.io/Pinhole.html> (visited on Dec. 15, 2021).
- Heikkila, Janne and Olli Silvén (1997). “A four-step camera calibration procedure with implicit image correction.” In: *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE, pp. 1106–1112.

- Idland, Tor Kvestad (2015). “Marine cybernetics vessel cs saucer:-design, construction and control.” MA thesis. NTNU.
- Kim, Dongnam (2018). *Human detection ros node using 2d-lidar and tx2 onboard camera fusion*. URL: https://github.com/4artit/human_detection_ros_node_using_2d-lidar_TX2-onboard-cam_fusion.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems* 25, pp. 1097–1105.
- LeCun, Yann A et al. (2012). “Efficient backprop.” In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.
- Lloyd, Stuart (1982). “Least squares quantization in PCM.” In: *IEEE transactions on information theory* 28.2, pp. 129–137.
- Marley, Mathias (2021). *Technical Note: Maneuvering control design using two path variable*, Rev B. Tech. rep. NTNU.
- Marley, Mathias, Roger Skjetne, Morten Breivik, et al. (2020). “A hybrid kinematic controller for resilient obstacle avoidance of autonomous ships.” In: *IOP Conference Series: Materials Science and Engineering*. Vol. 929. 1. IOP Publishing, p. 012022. DOI: 10.1088/1757-899x/929/1/012022. URL: <https://doi.org/10.1088/1757-899x/929/1/012022>.
- Marley, Mathias, Roger Skjetne, and Andrew R. Teel (2021). “Synergistic control barrier functions with application to obstacle avoidance for nonholonomic vehicles.” In: *2021 American Control Conference (ACC)*, pp. 243–249. DOI: 10.23919/ACC50511.2021.9482979.
- Millan, James (2008). “Thrust allocation techniques for dynamically positioned vessels.” In: *Laboratory Memorandum LM-2008-04, National Research Council, St. John’s, Newfoundland, Canada*.
- Moen, Jon Magnus (2021). “Automatic control with risk contingencies for autonomous passenger ferry.” MA thesis. NTNU.
- NTNU (2022). *Marine cybernetics laboratory*. URL: <https://www.ntnu.edu/imt/lab/cybernetics>.
- (2021). *Autonomous all-electric passenger ferries for ruban water transport (Autoferry)*. URL: <https://www.ntnu.edu/autoferry> (visited on Dec. 14, 2021).
- Ottesen, Are E (2014). “Situation Awareness in Remote Operation of Autonomous Ships.” In: *Shore Control Center Guidelines Norway*.
- Podareanu, Damian et al. (2019). *Best Practice Guide-Deep Learning*.
- Schöller, Frederik Emil Thorsson et al. (2020). “Vision-based object tracking in marine environments using features from neural network detections.” In: *IFAC-PapersOnLine* 53.2, pp. 14517–14523.
- Sharoni, Rotem (2016). “Marine Inverted Pendulum.” MA thesis. NTNU.
- Simonyan, Karen and Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition.” In: *arXiv preprint arXiv:1409.1556*.
- Singh, Archana, Avantika Yadav, and Ajay Rana (2013). “K-means with Three different Distance Metrics.” In: *International Journal of Computer Applications* 67.10.
- Skjetne, Roger (2005). “The maneuvering problem.” In: *NTNU, PhD-thesis* 1, pp. 95–98.
- (2021). *Cascade backstepping-based maneuvering control design for a low-speed fully-actuated ship*. Tech. rep. NTNU.
- Smogeli, Øyvind (2021). ‘Module 1: Introduction to autonomy in marine applications’, *Lecture Notes, TMR06 - Autonomous Marine Systems*.
- Solheim, Mathias (2021). “Sensor Fusion between camera and lidar for C/S Saucer, Specialization project.” MA thesis. NTNU.
- Steinhaus, Hugo et al. (1956). “Sur la division des corps matériels en parties.” In: *Bull. Acad. Polon. Sci* 1.804, p. 801.

- Ueland, Einar Skiftestad (2016). “Marine autonomous exploration using a lidar.” MA thesis. NTNU.
- Værnø, Sverre Arne and Roger Skjetne (2017). *Observer for simplified DP model: Design and proof*.
- Wang, Sun-Chong (2003). “Artificial neural network.” In: *Interdisciplinary computing in java programming*. Springer, pp. 81–100.
- Zeabus (2021). *NTNU and the 'milliampere ferries*. URL: <https://zeabus.com/miliampere/> (visited on Dec. 14, 2021).
- Zhang, Zhengyou (Dec. 2000). “A Flexible New Technique for Camera Calibration.” In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22, pp. 1330–1334. DOI: 10.1109/34.888718.
- Zitzewitz, Gustav von (2018). *Realtime Object Detection*. URL: https://github.com/gustavz/realtime_object_detection.

Getting started with ROS

This manual is intended to provide an introduction to the basic ROS procedures necessary to operate the CS Saucer in the Marine Cybernetics Library. It is intended for the use of future students at NTNU, whom to utilize the vessel for projects. The procedures presented in this manual are also generic enough that they apply to the other vessels of the cybership fleet.

The author of this manual has spent countless hours troubleshooting software, and setting up an environment suitable for operation on the Raspberry Pi. The hope is that the provided procedures will shorten the time future students will have to use by providing better documentation.

A.1 Raspberry Pi Image

Currently, all the vessels in the MC Lab run a Raspberry Pi 4B as their embedded computer. The CSE1 and CSE1 run the same raspberry pi image with Raspbian Buster as the OS and Melodic as their ROS-distribution.

For this thesis, the Raspbian Buster installation was insufficient, so a new image was created for this thesis. The main problem was that Raspbian Buster contained a fatal bug that prevents it from connecting to enterprise networks such as eduroam. Considering that much of the work is done on campus ground, and the Saucer required downloading a lot of software, a new operating system was installed for the Saucer. At the time of writing it runs Ubuntu 20.04 LTS Server edition, and ROS Noetic. Everything needed to run basic ros is installed, along with the computer vision library Open CV, Tensorflow and scipy as well.

The student can choose which OS suits their needs and make an image of the memory card on either the CS Saucer or CS Enterprise for their own use. More information about this can be found here. It is recommended to copy one of the memory cards rather than attempting a install from scratch.

A.2 Communicating with the Raspberry Pi

To communicate with the Raspberry Pi on the vessel, one needs to be connected to the 'MC lab' local network. After connecting to the network, open up a terminal and see if the vessel is also connected by running the command:

```
1 $ ping 192.168.0.108
```

Here the number-sequence is the IP of the Saucer. If the RPi responds you can SSH onto it and run files via the terminal. Simply run the command

```
1 $ ssh ubuntu@192.168.0.108
```

Most likely you will then be prompted for a password. All the RPi's have *marin33* as the designated password. You are now ready to run commands on the raspberry pi remotely.

A.3 How to ROS: A step by step guide

This section covers the basic ROS-commands you need to properly run scripts on either the RPi or your Ubuntu PC.

A.3.1 Sourcing ROS

ROS-programs are generally run using command-line tools. Thus, we have to source our installation when we open a terminal. To source the installation run the command

```
1 $ source /opt/ros/$ROS_DISTRIBUTION$/setup.bash
```

The ROS-distribution will either be called *melodic* or *noetic*, depending on which vessel you use. On the provided hardware ROS is sourced automatically via a bash-script every time you open the terminal, but if you are using your own computer you will have to do this for ROS commands to function.

A.3.2 Creating a workspace

Next, we need to create a designated workspace for our project. Navigate to the desired location on your computer (/documents, for example) and create a new folder for your project by running the following command

```
1 $ mkdir -p my_folder/src
```

This creates a folder with a sub-directory called *src*. You can replace *my_folder* with any name you want, but *src* must be kept the same. Standard ROS-convention is prefixing your workspace folder with *ws*, e.g *ws_dplab*.

Next, navigate to the workspace folder, and run the command

```
1 $ catkin_make
```

This will build your workspace environment with the proper ROS dependencies. After running this command, you should be left with a directory structure that looks like this:

```
my_folder
├── build
├── devel
└── src
```

You are now ready to run ROS-files. As you add more packages to your project, it is good to rebuild your project. This is done by running the *catkin.make* again.

A.3.3 The src-directory

This is the folder you will be working the most in, and were you will be putting all your ROS-packages. Each of these packages contain its own *src*-directory where the scripts you will be using are located. A typical project folder can look something like this:

```
my_folder
├── build
├── devel
└── src
    ├── simulator
    │   ├── launch
    │   └── src
    │       └── CSS.py
    ├── feedback_controller
    │   ├── launch
    │   └── src
    │       └── ctrl_joy.py
```

A.3.4 Running ROS nodes

After your code is written it is time two activate your ROS-nodes.

A.3.4.1 The ROS-master

First we need to make sure the ROS-master is running. On the raspberry-pi, this should be activated automatically. If you are running on the Ubuntu-computer you will have to enable it manually. Open a new command-line window and run the command:

```
1 $ roscore
```

Now the ROS-master is running, and you can start activating nodes.

If you are running nodes on separate computers, you need to make sure the computer not running the master knows were to find it. This is done by exporting the url for the ROS master. Before running any nodes, use the following command:

```
1 $ export ROS_MASTER_URI=http://192.168.0.108:11311
```

It is also smart to export the ROS.IP. This tells the master were the signals are coming from. You can determine your IP by running the command *ifconfig* and then

```
1 $ export ROS_IP=YOUR_IP
```

A.3.4.2 Activating nodes

In a separate command-line window, navigate to your project folder. Then source the `setup.bash` file in `devel`.

```
1 $ source devel/setup.bash
```

To activate a node, we first need to make sure that the script is executable. To make a file executable, navigate to the relevant directory and run the following command:

```
1 $ chmod +x <node-script>.py
```

If the script has become executable, the file-name should be green the next time you run `ls` inside the directory. Then, to activate a node run navigate to the base directory of your workspace and run

```
1 $ rosrun <package-name> <node-script>.py
```

where `<package-name>` is the name of the package you want to run, e.g "controller", and the `<node-script>.py` is the python script that initializes the rosnode, e.g "ctrl_joy.py".

A.3.5 Launch files

As a project progresses, more and more nodes are typically added. The process of activating nodes can therefore become more tedious as the complexity increases. To avoid having to open new command-line windows for every node, we can instead use *launch-files* to activate multiple nodes simultaneously. A launch file is easy to create, and is placed in the `launch` directory of a package, as illustrated in the directory-tree in section A.3.3.

Listing 1 Example of a launch-file

```
<launch>
  <node name="simulator" pkg = "simulator" type="CSS.py" />
  <node name="observer" pkg = "observer" type="observer.py" />
  <node name="guidance" pkg = "guidance" type="guidance_CBF.py" />
  <node name="controller" pkg="feedback_controller" type="cascade_backstepping.py" />
</launch>
```

Listing 1 shows the basic setup of a launch file. This example launches four separate nodes, the simulator, an observer, the guidance module and the controller module.

To run a launch file we use the following command:

```
1 $ roslaunch <package_name> my_launchfile.launch
```

Replace `package_name` with the name of the package you placed your launch file in. All launch files have the `.launch` ending (e.g `DP-system.launch`).

A.3.6 Topics

When nodes are activated, they will start to either *publish* or *subscribe* to *topics*. Topics are named buses over which nodes exchange messages. We can any topics that are being published/subscribed to in a ROS-system by using the command:

```
1 $ rostopic list
```

This will display all the active topics in a list in your terminal. In a DP-system we might, for example, have a thrust allocation algorithm that publishes actuator commands u to a topic that a the driver of the thrusters subscribes too. This topic is called something like **CSS/u**. If we wish to display the signals from this topic in real time we can run the command

```
1 $ rostopic echo CSEI/u
```

This will print every message that is sent to the given topic, and can be a useful tool when debugging.

A.3.7 Storing data

It is desirable to store the message-signals in the system in some format so that we can later analyze and plot the data. ROS provides its own tools and file-format for this, called a *bag* file. After we have launched some nodes we run the command

```
1 $ rosbag record <topic>
```

<Topic> is replaced with the names of the topics we wish to save. Multiple topics can be recorded at the same time, in the same bag file. You just have to list the topic names with a space. If i for example wanted to record the commanded force signal from our control law and the resulting actuator commands form the thrust allocation i would use

```
1 $ rosbag record CSS/u CSS/tau
```

When you have collected sufficient data, simply use CTRL+C to abort the operation. The data will be saved in a bag-file in your workspace.

A.3.8 Other usefull ROS-commands

```
1 $ rosnode list
```

Shows a list of all active ROS-nodes

Appendix **B**

Control system manual

B.1 System requirements

The following is a list of all software required to run the control system implemented in this thesis. The are split into to parts, general software, python libraries and ROS-packages.

Requirements	
General	
Operating System	Ubuntu 20.04 LTS (Server version)
ROS distrubution	Noetic
Python	3.8+
GCC	8+
OpenCV	
Python libraries	
Numpy	1.19.5+
tensorflow	
openCV	
scipy	
qp_solvers	
cvxopt	
ds4drv	
Ros packages	
CV-camera	
CV-bridge	
rplidar	
rosserial	
ds4_drivers	
dynamic_reconfigure	

Table B.1: Required software

B.2 Running the control system nodes

This section will describe the procedure for running the available control system. First, one must SSH onto the raspberry pi as described in Appendix A.2. Then navigate to the workspace. It should be called `ws_saucer`. Source the `setup.bash` file, and we are ready to start.

B.2.1 Dualshock 4 driver

The system relies on the playstation `ds4-driver`, so we start by launching this node. Its package should be in the directory. To activate the driver, run the command

```
1 $ roslaunch ds4_driver ds4_driver.launch
   use_standard_messages:=True
```

Running a `roslaunch` command will automatically start the ROS-master if you do not have one running. To connect the bluetooth controller, simply press the button on the middle. The controller should light blue when connected and an output should be printed in the terminal window. If the controller is not paired, you can follow this [tutorial](#).

B.2.2 Camera

Next up is the camera node. Open a new terminal. For this driver you do not have to be in the workspace.

```
1 $ rosruncv_camera cv_camera_node
```

You should see an output that the calibration file was loaded if successfully activated.

B.2.3 Lidar

To activate the lidar, open a new terminal and navigate to the workspace folder again. Then run the command

```
1 $ roslaunch rplidar rplidar.launch
```

Again, if successful you should see an output in the terminal. If it tells you the firmware is deprecated, just ignore the warning.

B.2.4 Arduino

Finally, before we activate the motion control system we must enable one final node. This is the node on the Arduino that subscribes to the pwm signals. To enable it, run

```
1 $ rosruncv_serial_python serial_node.py _port:=dev/ttyACM0
```

Here the final argument specifies which usb port the arduino is connected to on the raspberry pi. It may vary, so to check run

```
1 $ ls /dev
```

to see which devices are connected.

B.2.5 Motion Control System

Finally, we can activate the motion control system. In the workspace folder run

```
1 $ roslaunch feedback_controller collision_avoidance.launch
```

This should activate necessary nodes. The system defaults to manual control with the joystick controller, but can switch to automatic by pressing the triangle button on the ds4 controller. The guidance is activated by pressing square. Note that this should not be done unless the vessel is in the basin, and the object detection node is active.

If one wishes to for example test one node, each package can be launched individually using the guidelines from Appendix A.3.4.

B.2.6 Object detection

On your operator computer, navigate to the workspace folder of your choice. Remember to export the ROS master url to establish communication between the nodes. To run the SA system use the following command

```
1 $ rosrun objdetection detection_node
```

This will open a window with video-feed from the saucer on your operator computer.

B.3 Dynamic Reconfigure

To make tuning your controller and observer more effective, tools for dynamic tuning are provided in the form of the ROS-package `gain_server`. This allows you to change the gains in real time, rather than having to hard-code the gains and then stopping and starting your system every time you want to change them. To activate the node, and GUI use the following steps

Run the node use the command:

```
1 $ rosrun gain_server server.py
```

NOTE! If you use the dynamic tuning, you should always activate this node **first** or you may experience errors. This is because the subscribers in guidance, observer and controller nodes will expect values to be there when they are not. When the node is activated, the initial gains should be printed to your terminal. If you use the `collision_avoidance.launch` file the `gain_server` node will be started as part of this, and you can disregard the first step.

In another terminal, activate the GUI:

```
1 $ rosrun rqt_gui rqt_gui -s reconfigure
```

This should open a program that looks like this:

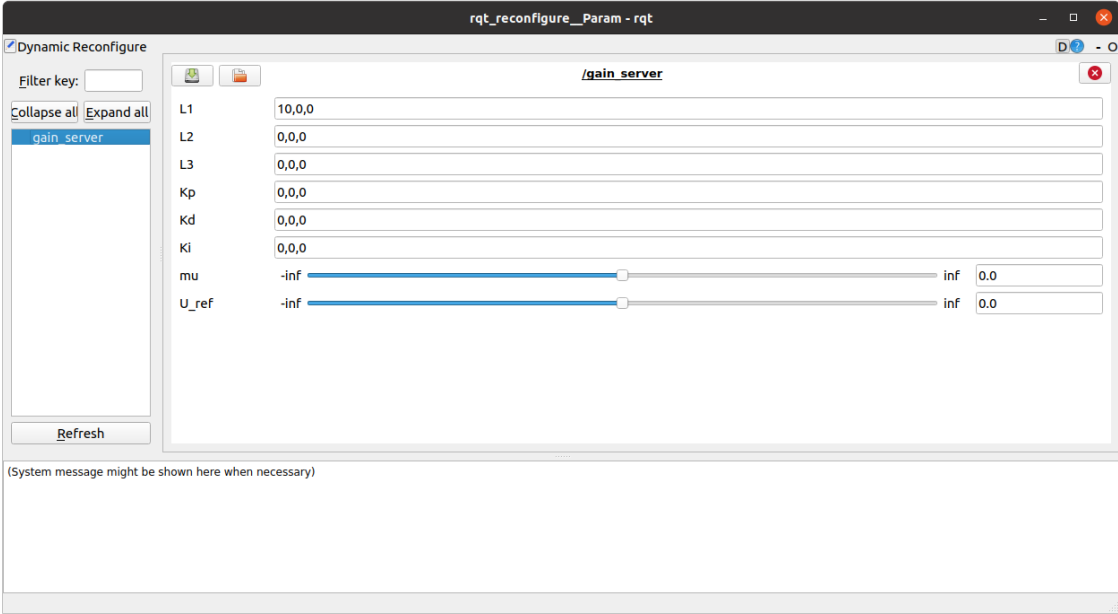


Figure B.1: Tuning GUI

Each field or slider here corresponds to a gain in either the observer or controller. Not all the variables are relevant either. The last two are single float values, while the first six are meant to represent the elements on the diagonal of a 3×3 -matrix. Change these to tune you controller.

Calibration Procedures

This manual is meant to walk the user through calibrating the situational awareness system of the CS Saucer. It encompasses the camera calibration and then the lidar and camera calibration procedures.

C.1 Camera calibration

First, make sure that the camera is correctly mounted, and that the ribbon cable is connected to the Raspberry Pi. For this calibration, you will need a checkerboard pattern. You can find some in the MC Lab, or print out your own on an A3 paper. It is recommended that you laminate the checkerboard so you can use it multiple times. The pattern used in this thesis was 9x7 squares with the length of 40 mm. The exact number and size is arbitrary as long as it is specified to the software. Place the Saucer somewhere high enough that you can move quite freely in the camera frame.

SSH on to the Raspberry pi via your operator computer, and run the camera driver as outlined in Appendix B.2.2. Then in a separate terminal window on the operator computer (not the RPi!) run the following comand:

```
1 $ rosrunc camera_calibration cameracalibrator.py --size 8x6 --square 0.40 image:=/cv_camera/raw_image camera:=/cv_camera
```

Here `-size` is the argument for number of inner corners in the pattern and `-square` the length of the sides. If the node is successfully activated, you will be met with the following window:

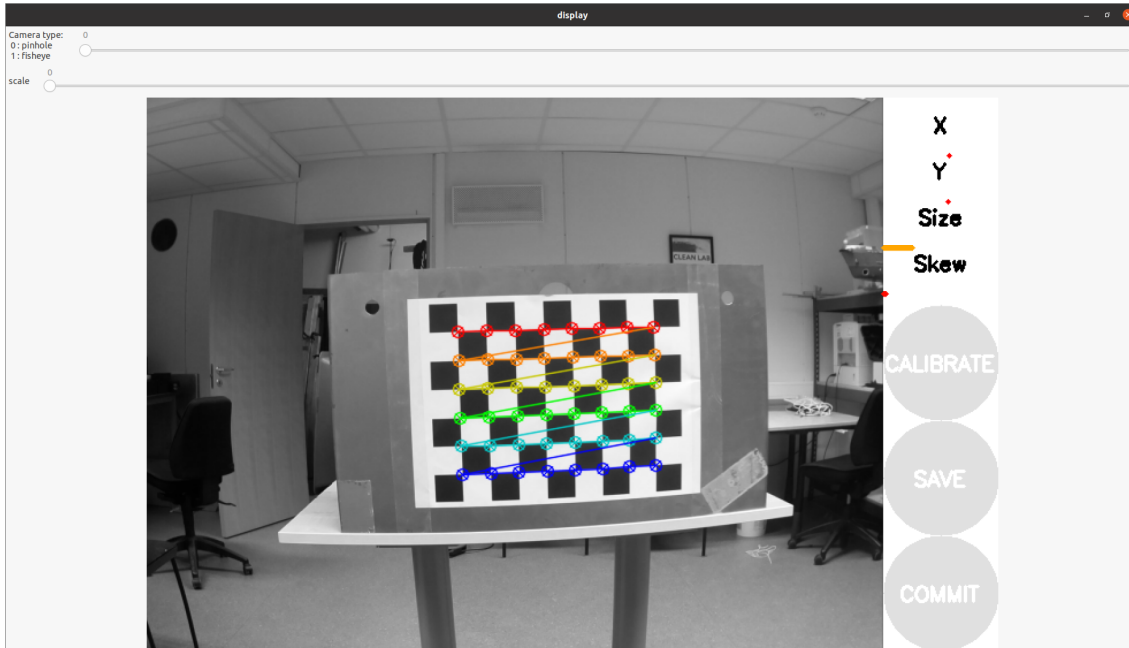


Figure C.1: Calibration window

Move the checkerboard around in different orientations, at different distances in the camera frame. The program will grab data-samples as you go. When the bars on the right of Figure C.1 are full enough, the calibrate button will turn a darker shade of grey. This means that enough data has been collected to produce a calibration. You can keep going to produce a more accurate calibration. When you are finished, simply press the calibrate button and the procedure will start. After it is over, the intrinsic parameters will be printed to the terminal window. To save these, press commit. This will make sure that the camera driver on the raspberry pi, always launches with these intrinsic parameters. You can also save the parameters to a file using the 'Save' button. Now you can exit the window.

C.2 Camera-Lidar calibration

The software for the lidar-camera calibration is based on ehong-tl, 2019. However, some changes have been made, so it is recommended that you use the files provided electronically with the thesis. Copy the calibration workspace over to your operator computer, and build it. Make sure you clean the files in the directory *data*. In the directory called *config*, edit the *config.yaml* file to contain the intrinsic camera parameters you obtained from the camera calibration.

Next you should identify the geometry you wish to use as your anchor points, and measure the height up to the lidar from the floor. Then, mark that height with tape on the corners inside the cameras point of view.

Then you can SSH onto the RPi and start the lidar and camera according to Appendix B.2.3 and Appendix B.2.2. On the operator computer, in the calibration workspace, run the command

```
1 $ roslaunch camera_2d_lidar_calibration
   collect_camera_calibration_data.launch
```

This will print you camera parameters in the terminal and activate the program RViz, as seen in Figure C.2

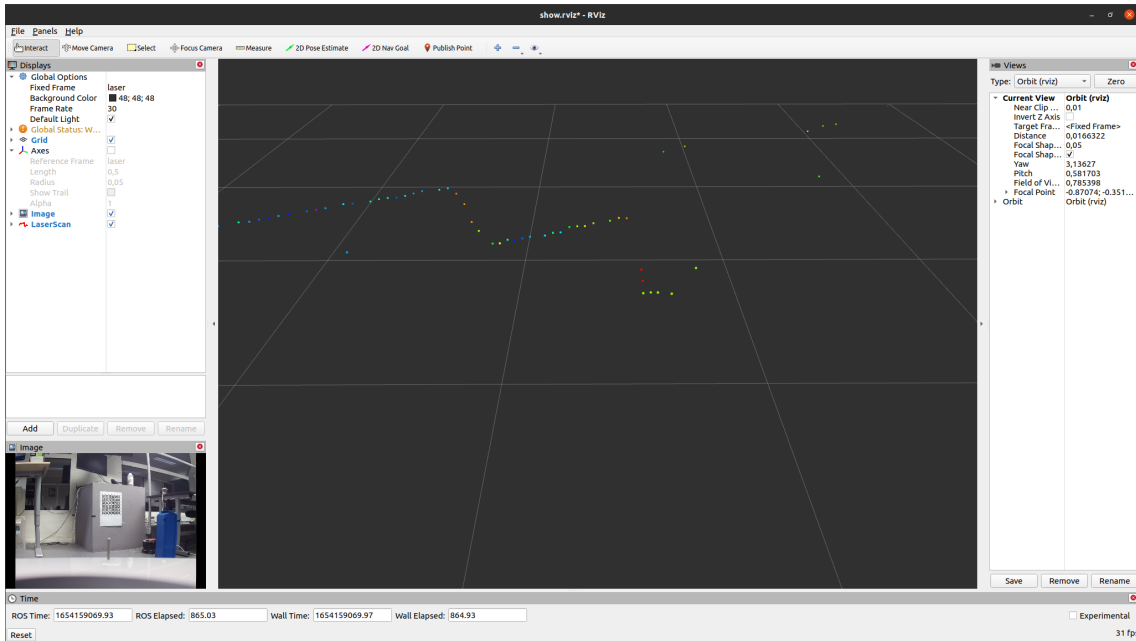


Figure C.2: RViz window for calibration

Then using the **2D Nav Goal** tool from the top bar, pick a point from the laser point cloud that corresponds to a corner. This will prompt the following window in Figure C.3



Figure C.3: Choosing corresponding pixel value

Mark the area in the image that corresponds to the lidar point you chose. This is not an exact science, but try to be as accurate as you can. When satisfied with the point placement, press space and the window will close and the point be saved. Repeat the

process several times for all the corners until you get enough data points. 10-20 should suffice, try to reduce the RMSE as much as possible. After you have enough data, close RViz and in the terminal run the second command

```
1 $ roslaunch camera_2d_lidar_calibration calibration.launch
```

This will run the camera and lidar calibration procedure and spit out the rigid body transformation along with the root mean square error. You can check the translation with hand-measurements, if they are in the same ballpark the calibration is ok. You can now try to reproject the laser point onto the live image. This is done by running

```
1 $ roslaunch camera_2d_lidar_calibration reprojection.launch
```

Here you can see how the lidar points line up with the tape.

Heading priority allocation

As part of this thesis, a second thrust allocation method was developed in an effort to stabilize the heading measurements. The method is proposed by Millan, 2008 and consists of a priority system for allocation, where heading is considered the most critical mode of control. This method requires that the vessel's azimuth angles vary, so the thruster configuration matrix must be augmented so angles can be determined from the allocated forces rather than manually set. To this end, each thruster is decomposed into Cartesian coordinates:

$$u_{i,x} = [1 \ 0 \ l_{1,y}]^T \text{ and } u_{i,y} = [0 \ 1 \ l_{1,x}]^T. \quad (\text{D.1})$$

This means the control input vector is extended to $\mathbf{u}_{ext} \in \mathbb{R}^6$

$$\mathbf{u}_{ext} = [u_{1,x} \ u_{1,y} \ u_{2,x} \ u_{2,y} \ u_{3,x} \ u_{3,y}]^T. \quad (\text{D.2})$$

Accordingly the *extended thruster configuration* matrix then becomes:

$$\mathbf{B}_{ext} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & r & r \sin -\frac{2\pi}{3} & -r \cos -\frac{2\pi}{3} & r \sin \frac{2\pi}{3} & -r \cos \frac{2\pi}{3} \end{bmatrix}. \quad (\text{D.3})$$

After solving (8.38), but substituting for \mathbf{B}_{ext} and \mathbf{u}_{ext} , the azimuth angle and thruster force of each respective actuator can be computed by simple trigonometry:

$$u_i = \sqrt{u_{i,x}^2 + u_{i,y}^2} \quad (\text{D.4a})$$

$$\alpha_i = \text{atan2} \frac{u_{i,y}}{u_{i,x}}. \quad (\text{D.4b})$$

For the heading priority scheme, (8.38) is solved with respect to only the yaw moment, i.e.,

$$\boldsymbol{\tau} = [0 \ 0 \ N]^T. \quad (\text{D.5})$$

In this way, an unconstrained optimization problem is solved to satisfy yaw demand. The resulting force vector and azimuth angles are computed using (D.4a) and then locked. The actuators are then checked for saturation. If none of the thrusters are saturated, the reserve thrust capacity can be used to satisfy the demand in surge and sway. A new configuration matrix must be defined for this purpose, using the optimal azimuth angles computed for heading.

Let

$$\mathbf{B}_{XY}(\boldsymbol{\alpha}) = \begin{bmatrix} \cos \alpha_1 & \cos \alpha_2 & \cos \alpha_3 \\ \sin \alpha_1 & \sin \alpha_2 & \sin \alpha_3 \end{bmatrix} \quad (\text{D.6})$$

be the thruster configuration matrix for allocating forces in surge and sway. To allocate, a *median search approach* is utilized. This basic procedure will attempt to distribute some percentage of the demand in surge and sway given by

$$\boldsymbol{\tau}_{XY} p_{ss} = \mathbf{B}_{XY}(\boldsymbol{\alpha}) \hat{\mathbf{u}}, \quad (\text{D.7})$$

where $\hat{\mathbf{u}}$ is the component of thrust for each thruster that will satisfy the remaining surge-sway demand, given the existing azimuth angles for heading priority. The variable $0 \leq p_{ss} \leq 1$ is the percentage of surge-sway demand to be allocated. The computed thrust components for *sda* are then summed with the ones for satisfying yaw, and thrusters are again checked for saturation. If one of the thrusters is saturated, the thrust components for surge-sway are rejected, and p_{ss} is decreased by 50%. Then the thrust components for satisfying surge-sway are recomputed with the new p_{ss} . This process is repeated until none of the thrusters are saturated.

DS4 Controller Mapping

Figure E.1 shows the current mapping of buttons and joysticks utilized in this thesis. As can be seen, there are plenty of free buttons that can be utilized for other functionality in future projects.

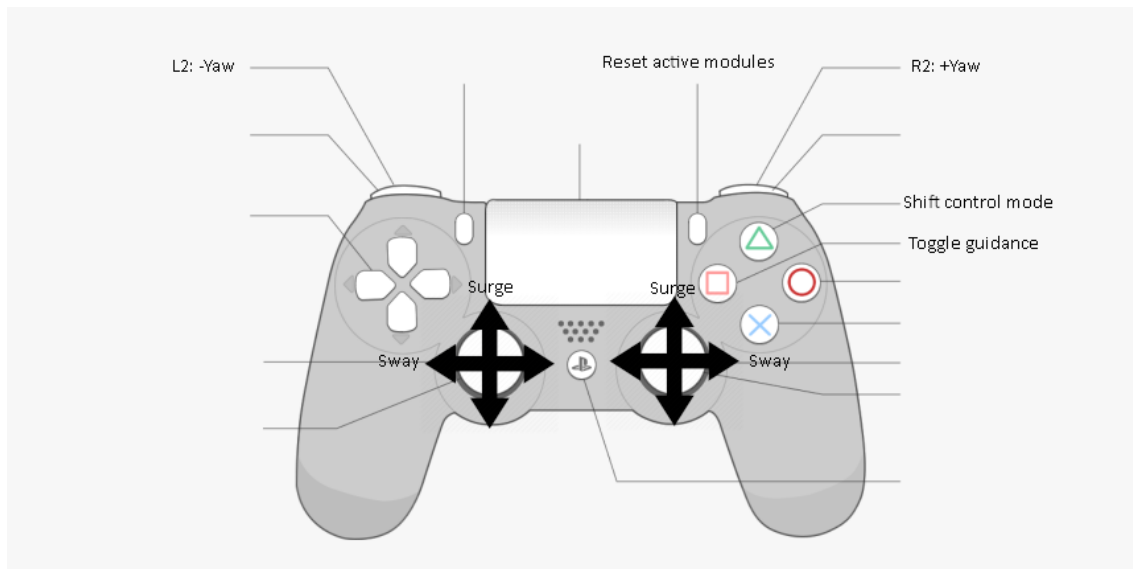


Figure E.1: Mapping of DS4 controller for manual and automatic control

