



Norwegian University of  
Science and Technology

# Autonomous Docking for Marine Vessels Using a Lidar and Proximity Sensors.

**Joachim Spange**

Marine Technology  
Submission date: December 2016  
Supervisor: Roger Skjetne, IMT

Norwegian University of Science and Technology  
Department of Marine Technology



If I have seen further, it is by standing on the shoulders of giants.

Isaac Newton, 1676.





## MSC Thesis DESCRIPTION SHEET

<b>Name of the candidate:</b>	Joachim Spange
<b>Field of study:</b>	Marine control engineering
<b>Thesis title (Norwegian):</b>	Autonom dokking av marine fartøy ved bruk av en Lidar og avstandsfølere.
<b>Thesis title (English):</b>	Autonomous docking for marine vessels using a Lidar and proximity sensors.

### Background

A highly maneuverable and robust multi-purpose marine model platform, called the “C/S Saucer”, has been developed for laboratory experiments the NTNU Marine Cybernetics Laboratory (MC-Lab). The intended use of this multi-purpose vehicle is for students to design, implement, and test a variety of nonlinear guidance, control, and estimation algorithms for specified experimental case studies.

This thesis will continue the work on autonomous exploration using a Lidar, by Einar Ueland, augmenting the platform with sensors detecting obstacles outside the scanned plane, to perform autonomous docking and take-off. Also updating the thrust allocation (TA) to account for rotatable azimuth angles, with constrained, in a fashion that is similar to a leisure boat with one or two stern thruster and the possibility of a bow thruster.

### Work description

1. Perform a background and literature review to provide information and relevant references on:
  - Unmanned surface vessels and autonomous functions.
  - Autonomous marine control systems.
  - Optimal thrust allocation (TA) methods applied on a leisure boat.
  - Lidar and proximity sensors, and SLAM algorithm.
  - MC-Lab and the C/S Saucer model.

Write a list with abbreviations and definitions of terms and concepts, explaining relevant concepts related to the literature and the project assignment.
2. Modify the TA on the C/S Saucer to perform optimally based on rotatable azimuth angles in the stern and a fixed azimuth angle in the bow. Study four different configurations:
  - 2 rotatable azimuth in the stern outputting equal thrust and angle, thus emulating one thruster/rudder.
  - Same as above but with a bow thruster outputting a limited thrust at a fixed angle.
  - 2 rotatable, independent, azimuth in the stern.
  - Same as above but with a bow thruster outputting a limited thrust at a fixed angle.

Simulate the behavior of the vessel with the four different configurations. Present all the four configurations as different “maneuverability modes” for C/S Saucer.
3. Modify the online visualization of the exploration process to include vessel heading. Include a function for clicking on a docking spot, to enable autonomous docking at that spot.
4. Integrate ultrasonic range detection, enabling obstacle detection outside the scanned plane by the Lidar, when performing docking and take-off.
5. Develop control algorithms for the automatic docking and take-off function. Simulate system responses and present and discuss results.
6. Test the implemented system in the MC-Lab to verify that the system is working. Present and discuss the results.



### Guidelines

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 various steps in the deduction.

The report shall be organized in a rational manner to give a clear exposition of results, assessments, and conclusions. The text should be brief and to the point, with a clear language. The report shall be written in English (preferably US) and contain the following elements: Abstract, acknowledgements, table of contents, main body, conclusions with recommendations for further work, list of symbols and acronyms, references, and optional appendices. All figures, tables, and equations shall be numerated. The original contribution of the candidate 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. *nativib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct. Such practice is taken very seriously by the university and will have consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with a printed and electronic copy to the main supervisor, each copy signed by the candidate. The final revised version of this thesis description must be included. The report must be submitted according to NTNU procedures. Computer code, pictures, videos, data series, and a PDF version of the report shall be included electronically with all submitted versions.

**Start date:** 5<sup>th</sup> August, 2016

**Due date:** 22<sup>th</sup> December, 2016

**Supervisor:** Roger Skjetne

**Co-advisor(s):** Einar Ueland, Dong T. Nguyen

Trondheim, \_\_\_\_\_



Roger Skjetne

Digitally signed by Roger Skjetne

Date: 2016.12.14 15:33:24 +01'00'

Roger Skjetne  
Supervisor

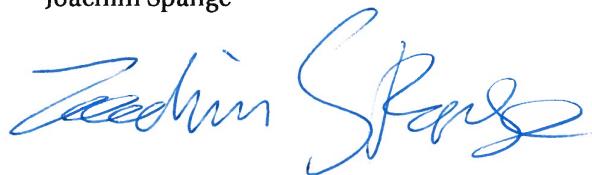
# Preface

This thesis concludes my five and a half years in Trondheim, studying at the Norwegian University of Science and Technology. As an experiment I have logged hours working on my thesis, the result became 990 hours of work, or just below 50 hours per week. I wish to thank the people I have met and befriended during my years in Trondheim; it truly is a great student city. I would also like to thank my student orchestra, Studentorchesteret Dei Taktlause. It has been necessary to have a group of people where I was the only marine cybernetician, such that we could talk about normal stuff.

In the spring of 2015, an idea came to mind: there are not much automatic control systems for leisure boats. Cars have become computers on wheels; the same thing can hardly be said about the consumer end of the naval fleet. In spring 2016, I wrote my project thesis on dynamic positioning functionalities for leisure boats, and this thesis follows the idea developing autonomous docking. This thesis has motivated me even further, and I am confident that autonomous features for leisure boats are possible, though the road towards a full-scale prototype is long. I believe this has a commercial opportunity, and maybe the next step would be to ensemble a team and create a spin-off.

The reader should preferably have knowledge of mathematical modeling of marine vessels, marine cybernetic and control theory.

Joachim Spange

A handwritten signature in blue ink, appearing to read "Joachim Spange".

Trondheim, December 21, 2016

# Acknowledgment

I would like to thank Professor Roger Skjetne for agreeing to supervise my own proposal for a Master's Thesis and being helpful along the way. I have been allowed to pursue my idea with feedback once I had any questions.

I would like to thank Einar Ueland, my first co-supervisor, for continuous supervision during this semester. He has been available all the time, on a short time notice, helping me resolve numerous problems. Since I am continuing his work on the CS Saucer, it was natural that he became my go-to guy.

I would like to thank Jon Bjørnø and Hans-Marin Heyn which whom I have shared an office with; there has allays been an atmosphere for asking questions.

Thanks to Dong T. Nguyen, my second co-supervisor, for help with finalizing my thesis. Together with Einar Ueland and Håkon Mork you have helped me produce my Master's thesis.

J.S.



# Abstract

This thesis reviews the development of an autonomous docking feature for a marine vessel in a small-scale, sheltered, environment. Proximity sensors are added to aid the autonomous exploration by detecting obstacles outside the lidars scan-plane, an assessment of different thrust allocations to emulate leisure boats are addressed.

The Cyber Ship Saucer, a model-scale vessel built for testing in the Marine Cybernetics Laboratory at NTNU, have been serving the platform for testing the docking scheme. The model and the lab, together with autonomous systems, optimal thrust allocation, sensors and SLAM is presented as a background for the experimental work performed in this thesis.

In addition to the fixed thrusters, four new configurations which considering rotation, emulating a leisure boat, have been studied. The "constrained control allocation for azimuth thruster" were applied to find an optimal solution for each of the modes.

The online visualization process has been altered such that obstacles detected by the proximity sensors become apparent for the operator, along with the heading of the vessel.

Through modifying the existing system, influencing both the path planning strategy and operator interaction, autonomous docking and take-off arose.

Simulations, and later experiments, verified the autonomous docking by utilizing the proximity sensors using a fixed thruster configuration. Through the user interface, the user is able to select a spot and then inputting the desired heading there. The results are presented, and discussed, in a separate chapter. The electronic attachment contains, among other, photos and videos from the experimental trials.



# Sammendrag

Denne avhandlingen gjennomgår utviklingen av autonom docking for et marint fartøy, i et avgrenset, skjermet, miljø. Avstandsfølere har blitt lagt til for å bistå med å søke der lidaren ikke ser. En vurdering av ulike trøstallokeringer for å emulere en fritidsbåt blir presentert.

Cyber Ship Saucer, et modellskala fartøy bygd for testing i barin kybernetikk-laboratoriet ved NTNU, har blitt brukt som plattform til å teste docking-algoritmen. Modellen og laben, sammen med autonome systemer, optimal trøstallokering, sensorer og SLAM blir presentert som bakgrunnsstoff for eksperimentene sol ble utført.

I tillegg til et oppsett av fikserte trøstere har fire nye konfigurasjoner, basert på roterende trøstere, blitt vurdert, dette fordi oppsettet bedre emulerer en fritidsbåt. En optimal løsning, for hver av de fire modusene, har blitt funnet ved hjelp av metoden "constrained control allocation for azimuth thruster".

Sanntidsprosessen som visualiserer prosessen har blitt endret slik at hindringer sett av avstandsfølerne dukker opp, sammen med retningen til fartøyet.

Det eksisterende systemet ble endret ved å modifisere strategien for å planlegge banen, samt brukergrensesnittet for å utvikle autonom docking og avgang.

Simuleringer, og senere eksperimenter, verifiserte autonom docking ved å bruke avstandsfølere og et fiksert trøsteroppsett. Via brukergrensesnittet vil brukeren kunne velge en plass, også angi en ønsket retning. Resultatene er presentert, og diskutert, i et eget kapittel. Det elektroniske vedlegget inneholder, blant annet, bilder og videoer fra de eksperimentelle forsøkene.



# Contents

MSc thesis description sheet . . . . .	iii
Preface . . . . .	v
Acknowledgment . . . . .	v
Abstract . . . . .	vii
Sammendrag . . . . .	ix
Nomenclature . . . . .	xix
Abbreviations . . . . .	xxi
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Background . . . . .	2
1.2.1 Optimal thrust allocation . . . . .	2
1.2.2 Proximity sensors . . . . .	3
1.2.3 Lidars . . . . .	4
1.2.4 Autonomous systems . . . . .	7
1.2.5 Unmanned Surface Vessels . . . . .	9
1.3 Thesis contribution . . . . .	12
1.4 Outline of the thesis . . . . .	13
1.4.1 Scope and delimitation . . . . .	13
1.4.2 Notation . . . . .	14
<b>2 Experimental setup</b>	<b>15</b>
2.1 CS Saucer . . . . .	15
2.1.1 Previous works . . . . .	15
2.1.2 The Robot Operating System . . . . .	16
2.1.3 Software architecture . . . . .	17
2.1.4 Hardware architecture . . . . .	18
2.2 Marine Cybernetic Laboratory . . . . .	23
2.2.1 Obstacles . . . . .	23
<b>3 Mathematical model</b>	<b>25</b>

3.1	Kinematics . . . . .	25
3.1.1	The Basin-fixed and Basin-relative reference frames . . . . .	25
3.1.2	Body-fixed reference frame . . . . .	26
3.1.3	Transformation between reference frames . . . . .	27
3.2	Kinetics . . . . .	28
<b>4</b>	<b>Control allocation</b>	<b>31</b>
4.1	Fixed angles . . . . .	33
4.2	Constrained rotatable thrusters . . . . .	34
4.2.1	Configuration 1 . . . . .	35
4.2.2	Configuration 2 . . . . .	36
4.2.3	Configuration 3 . . . . .	36
4.2.4	Configuration 4 . . . . .	37
4.2.5	Simulink implementation . . . . .	37
4.3	Simulations . . . . .	38
<b>5</b>	<b>Autonomous docking</b>	<b>41</b>
5.1	Problem statement . . . . .	41
5.2	Motion control system . . . . .	42
5.2.1	Steering law . . . . .	42
5.2.2	Docking and take-off . . . . .	43
5.2.3	PD-controller . . . . .	43
5.3	Map processing . . . . .	44
5.3.1	Draw test environment . . . . .	44
5.3.2	Including the proximity sensors . . . . .	45
5.3.3	Inflating the map . . . . .	46
5.3.4	Feasibility correction of the cost-map . . . . .	46
5.3.5	Path planner . . . . .	47
5.3.6	Map layers . . . . .	47
5.3.7	Operator interaction . . . . .	48
5.4	Sequence of docking and take-off . . . . .	48
<b>6</b>	<b>Results</b>	<b>51</b>
6.1	Simulations . . . . .	51
6.1.1	Adding proximity sensors to the mapping environment . . . . .	51
6.1.2	Autonomous docking . . . . .	52
6.2	Experiments . . . . .	53
6.2.1	Testing of modules . . . . .	53
6.2.2	Proximity sensors . . . . .	54
6.2.3	Autonomous docking . . . . .	56

6.3 Discussion . . . . .	57
<b>7 Concluding remarks</b>	<b>59</b>
7.1 Conclusion . . . . .	59
7.2 Further work . . . . .	61
<b>A MATLAB scripts</b>	<b>A1</b>
A.1 Create map for simulation . . . . .	A1
A.2 Angular dependent cost-map . . . . .	A2
<b>B Electronic attachments</b>	<b>B1</b>
B.1 Parameter generations files . . . . .	B1
B.2 Path_exploration node . . . . .	B2
B.2.1 Exploration_Pathplanner node . . . . .	B2
B.2.2 Path2SetPoint node . . . . .	B3
B.2.3 Scan2SetPointDist . . . . .	B3
B.2.4 Hector2VesselPos . . . . .	B3
B.2.5 Motion_Controller . . . . .	B3
B.2.6 Arduino code . . . . .	B4
B.2.7 Hector-Slam nodes . . . . .	B4
B.2.8 RPLidar node . . . . .	B4
B.2.9 ROS serial node . . . . .	B4
B.3 Simulator nodes . . . . .	B5
B.3.1 Vessel simulator node . . . . .	B5
B.3.2 Mapping fimulator node . . . . .	B5
B.4 Launch files . . . . .	B5
B.5 Other . . . . .	B6
B.5.1 Real-Time Pacer . . . . .	B6
B.5.2 Marine Systems Simulator . . . . .	B6
B.5.3 Photo and video documentation . . . . .	B6
B.5.4 Thesis files . . . . .	B6
<b>C Software set up and installation</b>	<b>C1</b>
C.1 Installing ROS and UBUNTU . . . . .	C3
C.1.1 Ubuntu and ROS on your personal computer . . . . .	C3
C.1.2 Ubuntu and ROS on your single board computer (RP2) . . . . .	C3
C.2 Getting started with ROS . . . . .	C4
C.3 Communicating between Raspberry Pi 2 and computer . . . . .	C5
C.3.1 Arduino on ROS . . . . .	C6
C.4 RP lidar and Hector-SLAM in ROS . . . . .	C7

<b>D Tutorials: ROS</b>	<b>D1</b>
<b>E Launch manual</b>	<b>E1</b>
E.1 Deploy vessel for autonomous exploration . . . . .	E1
E.2 Perform simulations . . . . .	E4
E.2.1 Launching . . . . .	E4
E.2.2 Run Simulink nodes independently . . . . .	E4
<b>F ROS architecture overview</b>	<b>F1</b>

# Figures

1.1 Constol system structure.	2
1.2 Concept of an ultrasonic proximity sensor.	3
1.3 2D lidar, illustrated with the RPLidar.	4
1.4 Vacuum cleaner XV-11 from Neato Robotics.	5
1.5 The driverless shuttle from EasyMile	6
1.6 10 levels autonomous control and how autonomy will improve over time.	7
1.7 Control architecture for unmanned underwater vehicles.	8
1.8 Mariner USV.	9
1.9 An unmanned ferry is planned to cross the a canal in Trondheim.	10
1.10 The autonomous vessel Hrönn will be tested in the Trondheimsfjord.	11
2.1 ROS publisher/subscriber architecture.	16
2.2 Signal flow between components.	17
2.3 A 170 point breadboard.	18
2.4 The HC-SR04 acoustic sensor.	20
2.5 Modules for increasing sway and yaw resistance.	22
2.6 Marine Cybernetics Laboratory, along with obstacles used in this thesis.	24
3.1 Basin-relative and Body-fixed reference frame.	26
4.1 CS Saucer thruster setup	32
4.2 The four modes, showing dead zones for the stern azimuth thrusters.	34
4.3 Top level of simulink model.	38
4.4 Mode 1-4 trying to produce the desired thrust.	39
4.5 The "MotionControl" Node	40
5.1 LOS steering law	43
5.2 Test environment for the simulator	45
5.3 Heading dependent weighting introduced to the system	47
5.4 Heading dependent weighting introduced to the system	48
6.1 The reference map, drawn in Paint, and the result after an autonomous exploration.	52

6.2	Docking of the CS Saucer. . . . .	53
6.3	Two modules tested on the CS Saucer, in the basin. . . . .	54
6.4	Test environment in the basin along with a figure illustrating the setup. . . . .	55
6.5	Scan of the basin showing obstacles detected by the lidar and the acoustic sensors.	56
6.6	Vessel behaviour before docking. See the jump in $\psi_{rel}$ around sample 500. . . . .	57
7.1	Suggestions for future work. . . . .	62
E.1	Overview over topics and nodes in the system, when running in the MCLab. . . . .	F3
E.2	Overview over topics and nodes in the system, when running as a simulation. . . . .	F4

# Tables

2.1	Pin connection overview . . . . .	19
2.2	Datasheet, HC-SR04 . . . . .	20
3.1	Parameters used in the maneuvering equation . . . . .	29
7.1	TRL in the European Commission ( <i>European Commision, 2014</i> ) . . . . .	60
F.1	Overview over nodes in the system in MCLab, see Figure F.1 . . . . .	F1
F.2	Topics in the system explained . . . . .	F2



# Nomenclature

$C_A(v_r)$	Hydrodynamic Coriolis matrix.	28
$C_{RB}(v)$	Rigid body Coriolis matrix.	28
$D$	Linear damping matrix.	28
$D_n(v_r)$	Non-linear damping matrix (includes linear components).	28
$I_z$	Moment of inertia about the z-axis.	29
$M_A$	Added mass matrix.	28
$M_{RB}$	Rigid body mass and inertia matrix.	28
$N_{\dot{r}}$	Added mass in yaw.	29
$N_{r v }$	Nonlinear damping in yaw.	29
$N_r$	Linear damping in yaw.	29
$T(\alpha)$	Thrust configuration matrix.	31, 35
$T_e$	Extended thrust configuration matrix.	35, 36
$X_{\dot{u}}$	Added mass in surge.	29
$X_{u u }$	Nonlinear damping in surge.	29
$X_u$	Linear damping in surge.	29
$Y_{\dot{v}}$	Added mass in sway.	29
$Y_{v v }$	Nonlinear damping in sway.	29
$Y_v$	Linear damping in sway.	29
$\Delta\psi_{\text{rel}}$	Angle between Basin-relative and Basin-fixed reference frame.	25
$\alpha_1$	Fixed angle of bow thruster.	33, 38

$\alpha_2$	Angle of port stern thruster. 33, 37, 38
$\alpha_3$	Angle of starboard stern thruster. 33, 37, 38
$\alpha_{23}$	The equal angle of port and starboard stern thruster. 37, 38
$\eta$	Position and attitude in the Basin-relative reference frame. 52
$\eta_d$	Desired position and attitude in the Basin-relative reference frame. 43, 52
$\hat{\eta}$	Observer estimated position and attitude in the Basin-relative reference frame. 43
$\hat{v}$	Linear velocities. 43
$v$	Linear velocities. 27, 28
$v_r$	Linear velocities relative to local current. 28
$\psi_{\text{docking}}$	Desired docking angle. 43
$\tau$	Force [N] is the Body-frame. 28, 31, 33, 35, 43
$\tau_c$	Commanded force [N] from the thrust allocation. 38
$\tau_d$	Desired force [N] is the Body-frame. 38
$f$	Force vector [N] for each thruster. 31, 33, 38
$f_e$	Extended force vector [N], decomposed forces for each thruster. 35, 36
$g(\eta)$	Vector of gravitational/buoyancy forces and moments. 28
$g_0$	Vector used for pretrimming. 28
$m$	Total mass of the system. 29
$m_e$	Mass of the module attached to the vessel. 29
$m_v$	Mass of the vessel. 29
$r_t$	Radius from center of origin to thrusters. 31
R	Rotation matrix. 27, 43

# Abbreviations

AUV	Autonomous Underwater Vessel.	11
BPA	Boat Parking Assistance.	11
DOF	Degree of Freedom.	2, 27, 28
HMI	Human Machine Interface.	21
IMU	Inertial Measurement Unit.	62
LOS	Line-of-Sight.	42, 62
MCLab	Marine Cybernetics Laboratory.	15, 19, 23, 39, 44, 51, 53, 55, 57, E2
PWM	Pulse Width Modulation.	18, 20
ROS	Robot Operating System.	14–17, 21, 26, 39, 58, 59, 62, E5
RP2	Raspberry Pi 2.	15, 17, 18, 21, E1
SLAM	Simultaneous Localization and Mapping.	2, 5, 15, 25, 26, 47, 48, 52, 61, 62
TA	Thrust Allocation.	2, 34, 38, 39, 61
TRL	Technology Readiness Level.	60–62
USV	Unmanned Surface Vehicle.	9



# Chapter 1

## Introduction

### 1.1 Motivation

In October 2016 Tesla Motors wrote the blog post "All Tesla Cars Being Produced Now Have Full Self-Driving Hardware". The hardware package is described as:

*Eight surround cameras provide 360 degree visibility around the car at up to 250 meters of range. Twelve updated ultrasonic sensors complement this vision, allowing for detection of both hard and soft objects at nearly twice the distance of the prior system. A forward-facing radar with enhanced processing provides additional data about the world on a redundant wavelength, capable of seeing through heavy rain, fog, dust and even the car ahead.*(Tesla Motors, 2016)

In the blog post, there is a video of a man entering his car and being driven to work without any interaction with the car. He exits the car in front of the main entrance leaving the car to find a valid spot to park. Technology on land have always been leading on marine technology, for the consumer market, thus in a few years the same advanced "Self-Driving" technology might be available on leisure boats.

In this thesis a lidar is used instead of a radar, while four acoustic sensors complement the vision by detecting obstacles outside the plane scanned by the lidar. The lidar is often placed high up to get an overview. Due to its placement, it will have a blind spot as a perimeter around the vessel. Adding sensors to look in the blind spot is essential, a low-cost acoustic sensor is good at close range detection and thus well suited. The establishment of using a marine vessel with inexpensive sensors to operate and dock autonomously in a controlled laboratory environment is a step automation of smaller, recreational, boats. Other indications that leisure boats will become more autonomous is the industrial initiative toward creating autonomous vessels. Several examples follow in the next section.

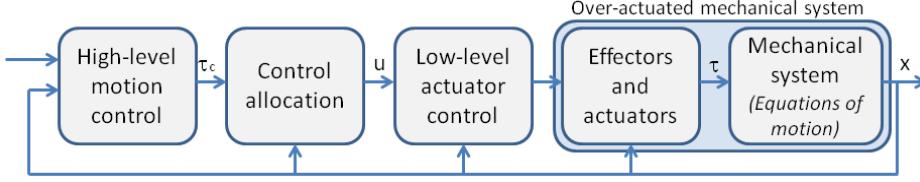


Figure 1.1: Constrol system structure, the first three systems referred to as the tree levels of control algorithm hierarchy.  $\tau_c$  denotes the commanded, virtual, control effort, and  $\tau$  is the actual, allocated, control effort. Courtesy Johansen and Fossen (2013)

## 1.2 Background

This section aims at providing a basic understanding on the topics of thrust allocation, autonomous systems, unmanned surface vessels and the sensors used; lidar; proximity sensor. The background of this thesis overlaps with some of the works presented by Ueland (2016). Hence an introduction to the topics autonomous mapping, map representation, Simultaneous Localization and Mapping (SLAM), path planning and exploration strategies, this thesis refers to Ueland (2016, Sec 1.2.1-1.2.5). The background topics presented here, which may still be overlapping, are aimed to be explained through other examples.

### 1.2.1 Optimal thrust allocation

Johansen and Fossen (2013) divided the control hierarchy of over-actuated systems commonly includes three levels, see Figure 1.2.1. First, a high-level motion controller, such as a PD-controller, is responsible for generating a vector of virtual control efforts,  $\tau_c$ , to meet the overall motion control objective. Second, a control allocation coordinates the effectors such that they combined produce  $\tau_c$ . Third, a low-level control algorithm may be used to control each individual effector via its actuators. This section focuses on the second layer.

When discussing Thrust Allocation (TA) it is important to distinguish between an underactuated and a fully actuated marine craft. According to Fossen (2011, Sec. 1.2.2), marine crafts with less control inputs than generalized coordinates ( $r < n$ ), are underactuated. Equal or more control inputs results in a fully actuated marine craft. If  $r > n$ , it will be referred to as over-actuated. Throughout this thesis only surge, sway and yaw are studied, hence  $n = 3$  which means that the system is 3 Degree of Freedom (DOF), where DOF is defined by Fossen (2011) as:

*For a marine craft, DOF is the set of independent displacements and rotations that completely specify the displaced position and orientation of the craft. A craft that can move freely in the 3-D space has a maximum of 6 DOFs, three translational and three rotational components.*

In the case of an over-actuated system, there are often many, or infinitely many, solutions to allocate the desired thrust. A solution, as suggested by Sørdalen (1997), is to use the Moore-Penrose

pseudoinverse, which minimizes the square of the control input. If the problem is augmented, including properties such as input saturation, rate constraints, and forbidden zones an optimal solution can be found relative to the importance of each constraint.

One such solution is the "constrained control allocation for Azimuth thruster" presented by Fossen (2011, Sec. 12.3.4), based on the works by Johansen et al. (2004), which is an iterative solution using quadratic programming.

For further background information, see Johansen and Fossen (2013), and Frederich (2016, Sec. 1.2).

## 1.2.2 Proximity sensors

A proximity sensor is a sensor able to detect the presence of nearby objects without any physical contact. The methods to accomplish this span over a variety of concepts. In this thesis the proximity sensor uses ultrasonic waves measuring the time,  $t$ , it takes from the signal is sent to received. Usually, the frequency is around  $40\text{kHz}$ , well above human hearing. The speed of sound is around  $v_s = 340 \text{ m/s}$ , hence the distance to the object becomes

$$d = \frac{t * v_s}{2}. \quad (1.1)$$

It is important to divide by 2 since the sound travels the distance twice, see Figure 1.2. If the sensor had a time step is  $1\mu\text{s}$  the resolution becomes  $0.3\text{mm}$ . The biggest advantage using an ultrasonic sensor is that it detects many materials, and can be very cheap. A drawback is how the speed of sound varies with, among other, temperature, leading to a error. Another drawback is that if the target is inclined the sound might reflect in another direction and may not hit the receiver, hence the object is not detected.

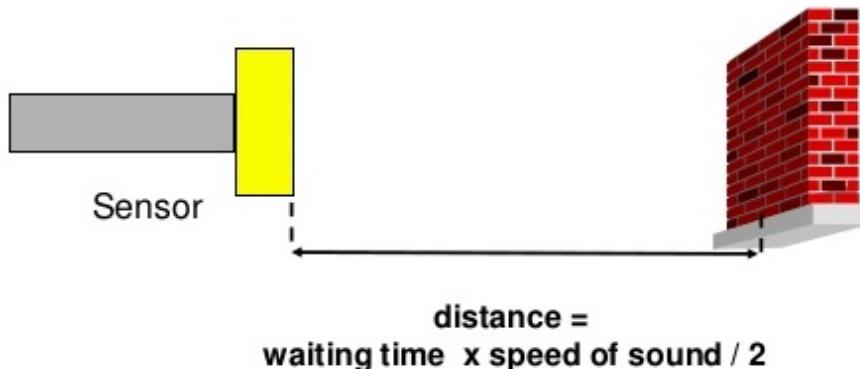


Figure 1.2: Concept of an ultrasonic proximity sensor. Courtesy S. Vyasarasamudra

### 1.2.3 Lidars

A lidar is a remote sensing device that measures the distance to nearby targets by illuminating its environment with a laser, and analyzing the reflected light. The lidar emits a laser pulse that is reflected by the object it reaches. The returning signal is sampled by vision acquisition embedded in the lidar. By measuring the time that the light uses to return, a point cloud is generated, which can be utilized for mapping and localization.

Lidars are recognized for high accuracy, allowing for fast data acquisition and for being independent of ambient light. Figure 1.3 illustrates how the 2D lidar installed on the vessel emits a laser pulse that is reflected by a wall and sampled by vision acquisition in the lidar. This allows the system to sense its environment in the 2D horizontal plane. Lidars have a more detailed description by Ueland (2016).

In the following two examples are presented, one using a similar 2D lidar, the other a 3D lidar with a narrow vertical field of view.

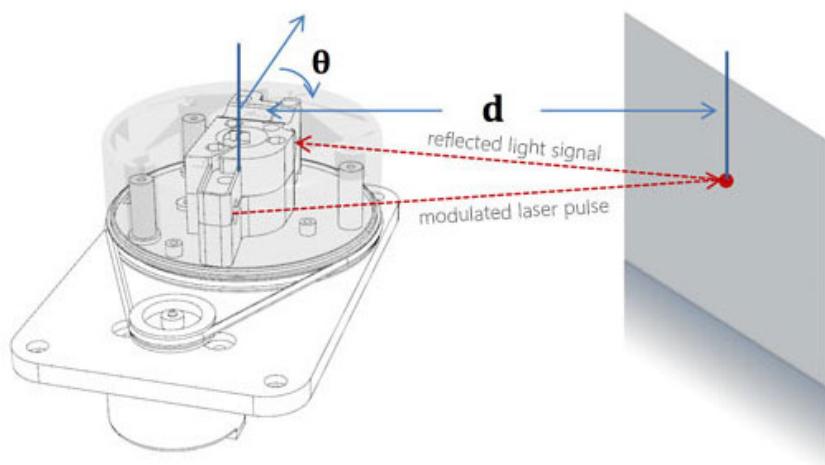


Figure 1.3: 2D lidar, illustrated with the RPLidar. Courtesy Robotshop

### 1.2.3.1 Example 1, the XV-11 from Neato Robotics

Neato describes their autonomous vacuum cleaner as:

*Patented laser-guided technology scans and maps the room, plans, and methodically cleans—instead of just bumping around.*(*Neato XV Series*, 2016)

Figure 1.4 displays the vacuum cleaner with its lid opened, where the lidar is clearly visible. *Janez Cimerman* (2015) published a tutorial on how to use the lidar from the XV-11 together with the Hector SLAM ROS packaged, from *Kohlbrecher S, M. J.* (2014). Therefore, it appears as the lidar from the XV-11 could be used instead of the RPLidar to perform SLAM and path planning as was done by Ueland (2016).

This looks similar to the . The similarity has been confirmed by *Janez Cimerman* (2015), which published a tutorial on how to use the lidar from the XV-11 together with the Hector SLAM ROS packaged, from *Kohlbrecher S, M. J.* (2014).

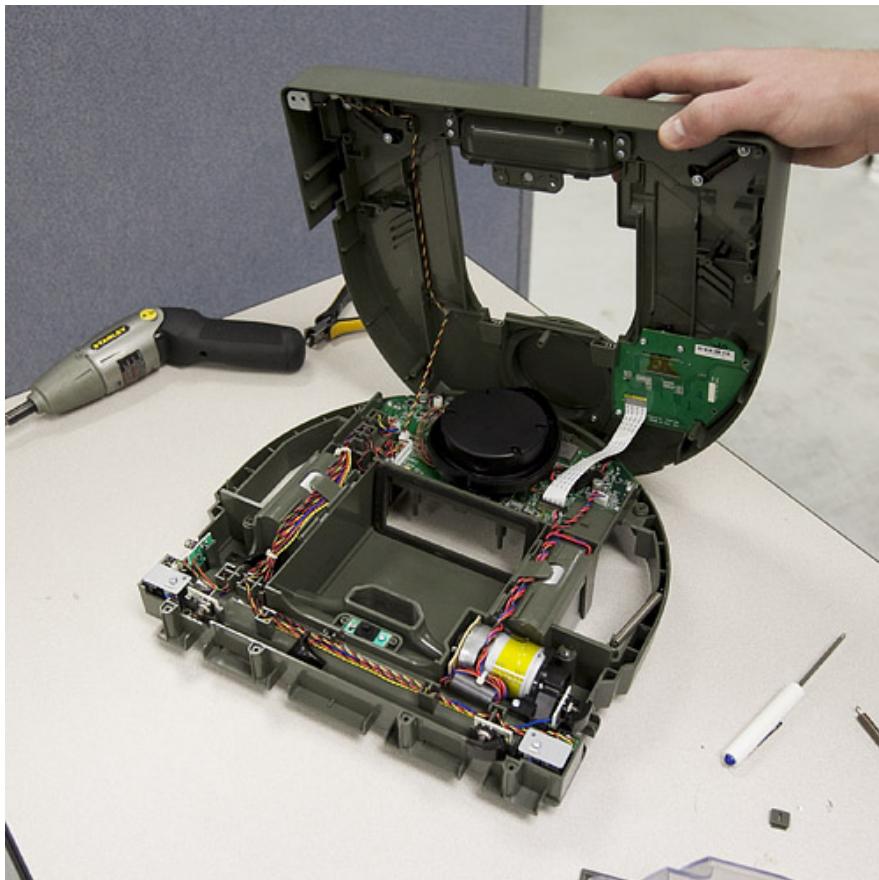


Figure 1.4: Lid opened on the vacuum cleaner XV-11 from Neato Robotics.  
The 2D lidar is clearly visible, illustrated with the RPLidar. Courtesy *SparkFun Electronics*

### 1.2.3.2 Example 2, the EZ10, the driverless shuttle

EasyMile describes their shuttle bus, the EZ10, as:

*[...] an electric shuttle dedicated to smart mobility designed to cover short distances and predefined routes in multi-use environments.*(EasyMile, 2016)

The bus was traveling a short predefined path around the west side of the main building at NTNU, see Figure 1.5a. A lidar is clearly visible in the front; there was also another in the back. It is believed by the author that it is the VLP-16, Figure 1.5b from *Velodyne LiDAR* (2016a). It has a vertical field of view of  $\pm 15^\circ$ , making it a 3D lidar.



(a) Ez-10 from *EasyMile* pictured in front of the main building at NTNU



(b) The VLP-16. Courtesy *Velodyne LiDAR*

Figure 1.5: The driverless shuttle from EasyMile, which may be using on PUCK from *Velodyne LiDAR* in the front and back.

#### 1.2.4 Autonomous systems

When discussing autonomy, it is important to distinguish between an automatic and an autonomous system. Automatic systems can perform pre-defined task, without any human intervention, such as path following. An example of this is how the EZ10 shuttle bus had been programmed to follow a pre-defined path, its path was in between prohibitory line, to keep human away. The same bus, in an open world environment, would be considered as autonomous. Autonomous systems are more intelligent when unexpected events occur; they are designed to model and plan their actions to make choices.

To distinction two autonomous systems from each other it is common to classify systems according to their level of autonomy. Figure 1.6 shows a 10 level distinction, while four levels are presented by Ludvigsen and Sørensen (2016), both systems are ranging from remote control to fully autonomous.

For a detailed description of each of the ten levels, see National Research Council (2005). Three of the levels will now be presented.

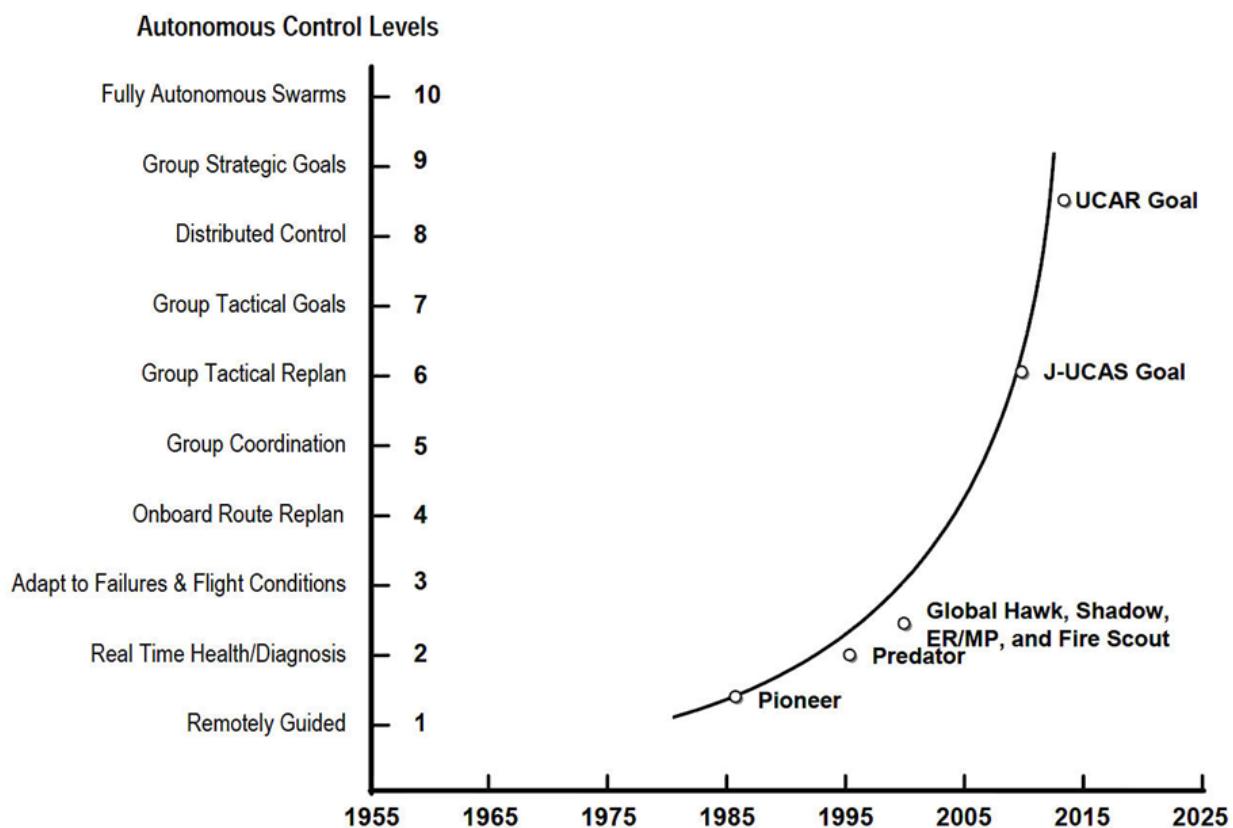


Figure 1.6: 10 levels autonomous control and how autonomy will improve over time. Corutecy National Research Council (2005)

Autonomy level 1 is the same as an automatic operation, there is no decision-making ability, a remote operator controls the vessel through high-level mission planning. The level of autonomy is increasing, level by level, until level 10, which defines "Full autonomy". The system independently plans the mission to meet the defined objectives, possibly through collaboration with other vessels. The human may monitor the process without any supervision, i.e. human-out-of-the-loop.

Level 7 is defined as "Local sensor fusion" with robust planning and negotiation of complex terrain, environmental conditions, hazards, and objects. Operator interaction is at a minimum, typically to intervene with certain decisions. Level 7 is specifically mentioned, the CS Saucer performing autonomous exploration (Ueland, 2016), would be placed here.

In Ludvigsen and Sørensen (2016) a "bottom-up" approach toward autonomy was presented, the architecture is shown in Figure 1.7, consisting of three layers. The top layer concerns the mission objectives and planning. Contingency handling, from the guidance system or payload sensor, may result in a re-planning. The guidance and optimization layer concerns reference signals from the controller along with waypoint management and generation to be used in the mission layer. The lowest layer is where the control execution takes place, in the controller, there

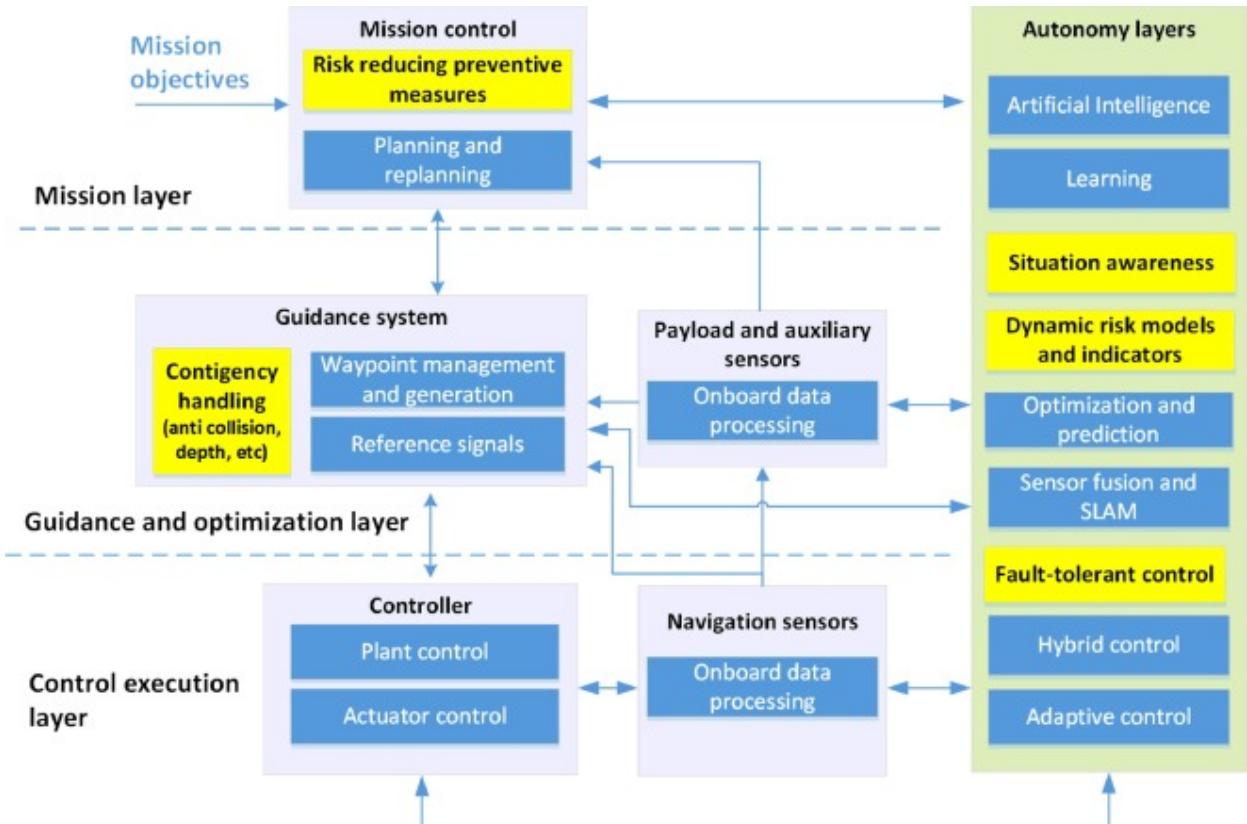


Figure 1.7: Control architecture for unmanned underwater vehicles. Ludvigsen and Sørensen (2016)

is a process plant to perform actuator control.

For topics on, among other field campaigns, see Ludvigsen and Sørensen (2016), for reading on plant and actuator control, see Sørensen (2005).

### 1.2.5 Unmanned Surface Vessels

An Unmanned Surface Vehicle (USV) is a vessel that operates on the water surface without the use of an operator on the vessel. In this chapter several examples will be presented.

#### 1.2.5.1 Maritime Robotics

Maritime Robotics has a USV called "Mariner", see Figure 1.8. It is described as a multipurpose, stable, unsinkable and near maintenance-free (*Maritime Robotics*, 2016). It can be built to fit into a 20 feet container, for cargo shipping, and is equipped with a payload room of more than one cubic meter. It has a 300mm moon-pool to mount any sub-surface sensors.

Maritime Robotics also offers a "USV Conversion System", they have, e.g., converted a military patrol boat to a vessel that can be used in unmanned operations. They have also converted leisure crafts, such as a Viknes 830. The latter was studied by Kjerstad (2010). The conversion is described to take 1-2 work days.

In Breivik (2010) formation control is presented where the Mariner USV and the Viknes 830 are recording the position of NTNU's research vessel *RV Gunnerus* (2016), a 30m long displacement vessel capable of 13 knots. The formation position were defined relative to Gunnerus' Body-fixed frame.

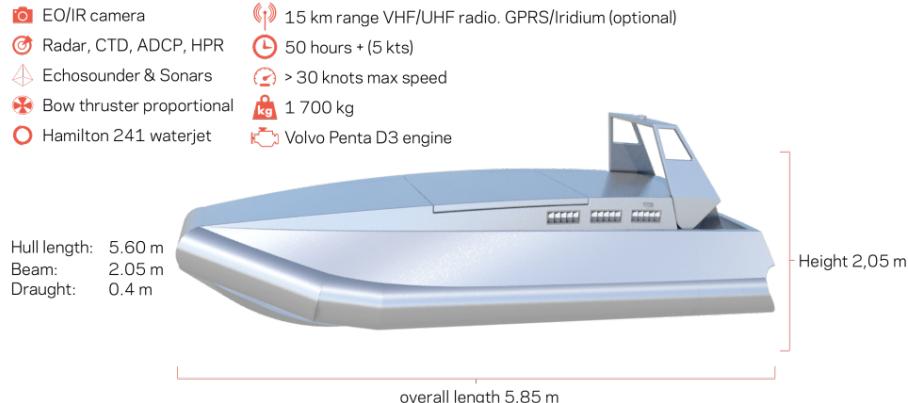


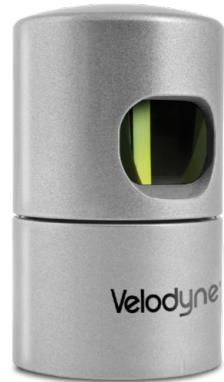
Figure 1.8: Mariner USV. Courtesy *Maritime Robotics*

### 1.2.5.2 Milliampere

A NTNU project started earlier in 2016, developed a model of an autonomous ferry to carry people across a canal in Trondheim, see Figure 1.9a, it is an alternative proposition to the NOK 60M concept of building a bridge (*adressa*, 2016). The model is in scale 1:2 and is a collaborative project between the Faculty of Engineering Science and Technology (*IVT Faculty* (2016)), the Faculty of Information Technology, Mathematics and Electrical Engineering (*IME Faculty* (2016)) and Centre for Autonomous Marine Operations and Systems (*AMOS* (2016)). The project was covered by the local and national press, among other by *Teknisk ukeblad* (2016a) which is a Norwegian technical magazine. The full-scale vessel will be 10 meters long and a width of 3,5 meters, with a capacity of 12 passengers. In the article lidar and IR-camera are mentioned as possible sensors. The vessel must also be able to dock and take-off, autonomously, and charge its batteries while it is docked. The author recommended the HDL-32E (*Velodyne LiDAR*, 2016b) prior to the publishing of the magazine article, which was bought and will be studied. See Figure 1.9b.



(a) Milliampere. Courtesy *Aanondsen, S. A.*



(b) HDL-32E lidar. Courtesy *Velodyne LiDAR*

Figure 1.9: An unmanned ferry is planned to cross the a canal in Trondheim. The HDL-32E lidar is considered to be used as part of the sensor suite.

### 1.2.5.3 Hrönn

Another autonomous vessel is the "Hrönn", also described in *Teknisk ukeblad* (2016b) (see Figure 1.10). Kongsberg and Automated Ships Ltd will build the 35 m by 10 m vessel. According to the article, the design process is still at an early state, the range of application are listed as ROV-operations, being a mother vessel for Autonomous Underwater Vessel (AUV), e.g. Hugin. The vessel will be tested in the Trondheimsfjord in 2018, and will, in the beginning, be remotely operated before becoming autonomous.

### 1.2.5.4 Astra yacht

This section is also described by Spange (2016). ASTRA Yacht delivers a custom system Boat Parking Assistance (BPA) which can be installed in any engine, such that the yacht autonomously orient itself relative to the sea floor, and can this way it counteracts wind and currents. The captain can maneuver the boat relative to the sea floor, making the yacht more agile than a car since it can rotate without any translation. ASTRA Yacht delivers, installs and calibrates the BPA no price is presented on their web page; however, since this is a product for yachts, it can be assumed that it is in the upper price range, and likely therefore also likely too expensive to implement in any leisure boat. Berretta et al. (2013) fitted a "Seacode Cabi 28" with BPA, the vessel had two stern thrusters, one side thruster in the bow and another in the stern.



Figure 1.10: The autonomous vessel Hrönn will be tested in the Trondheimsfjord.  
Courtesy *Kongsberg*

### **1.3 Thesis contribution**

The main contributions of this thesis are as follows:

- Suggestions on constrained optimal control along with thrust configuration matrices for four different thruster configurations, ranging from over actuated to underactuated emulating the configuration of a leisure boat.
- Presenting a method for deploying the CS Saucer any environment. As long as the primary colors and resolution match anything can be drawn and converted to the occupancy grid.
- Interfacing proximity sensors into the hardware and software architecture aiding the lidar in avoiding hazardous situations.
- Creating, and testing, two physical modules for the CS Saucer, they are attached to the exterior. Their purpose is to change the vessel dynamics.
- Developing a steering law which combines line-of-sight equations with setpoint generation from the velocity control law by Ueland (2016).
- Modifying the existing path planner, mathematical model, and control system to perform autonomous docking. The docking becomes an option for the operator.
- List of recommended tutorials for beginners with Robot Operating System are given in Appendix D.

## 1.4 Outline of the thesis

### 1.4.1 Scope and delimitation

This thesis is built up around the Cyber Ship Saucer, studying thrust allocation and implementation of proximity sensors to perform autonomous docking.

**Chapter 1** introduces the thesis to the reader. It provides background on thrust allocation, proximity sensors, lidars and autonomous vessels.

**Chapter 2** presents the previous work on the CS Saucer along with its operating system, software- and hardware architecture. Together with the Marine Cybernetic Laboratory this make up the experimental setup

**Chapter 3** covers the mathematical model, through explaining the kinematics and the kinetics.

**Chapter 4** presents, and concludes, control allocation based on fixed or rotatable thrusters.

**Chapter 5** presents how the motion control system and the map processing was altered to achieve the autonomous docking feature. It also gives an account of how the operator should interact with the map to perform docking and take-off.

**Chapter 6** presents the results from simulations and experiments.

**Chapter 7** covers the concluding remarks summarizing the thesis and presenting suggestions for further work.

**Appendix A** presents two MATLAB scripts which are specifically referred to in the thesis.

**Appendix B** presents an overview of the electronic attachment provided with this thesis.

**Appendix C** shows how to install the necessary software.

**Appendix D** presents tutorials for beginners in ROS.

**Appendix E** is a launch manual for deploying the vessel on water but also performing simulations.

**Appendix F** contains two figures of the ROS architecture, as the system is deployed on water and when running as a simulation, respective.

As recognized by Ueland (2016), the following address some of the limitations of the resulting system:

- Experiments have to be performed in a controlled environment with the following properties
  - There are no waves or current present during testing.
  - Walls and hindrances are vertical and not transparent. These requirements are necessary for both the lidar and the proximity sensors.
  - The performed operations are small-scale, meaning that there are always some objects within the range of the lidar.
  - To steer the heading a divinycell module needs to be attached to the vessel.
- An external computer, connected to the system through WiFi, is needed to run the exploration node.
- Obstacles need to be fixed. It is expected that the system will experience problems in a dynamic environment.

### 1.4.2 Notation

Throughout the thesis regular style font is used for all math notation. Scalar, vector or matrix should be clear from context. Text written in cursive, is a definition or a direct copy from the reference. Text written in quotation marks refers to a specific name, such as a MATLAB-file or a Robot Operating System (ROS) node

# **Chapter 2**

## **Experimental setup**

### **2.1 CS Saucer**

The vessel used in this thesis is the model-scale, surface vessel, CS Saucer. It has a circular water-plane area and was designed to be omnidirectional. Its top and bottom diameter are  $548\text{mm}$  and  $398\text{mm}$ , respectively. The mass is about  $3.4\text{kg}$ . This chapter presents the CS Saucer, both by reviewing previous works and by presenting new developments.

#### **2.1.1 Previous works**

CS Saucer was designed and built by Idland (2015). Idland also designed a control system implementing it in LabVIEW from National Instruments.

Ueland (2016) redesigned the control system to ROS, introducing Arduino and Raspberry Pi 2 (RP2). Ueland implemented a lidar sensor and developed an autonomous guidance algorithm that performed SLAM to autonomously explore a test environment set up in the Marine Cybernetics Laboratory (MCLab).

## 2.1.2 The Robot Operating System

The programming platform utilized on the vessel is the The Robot Operating System (ROS), which was released in 2007. It provides services that resemble that of operating systems, such as message parsing between processes, and package management. A ROS system consist of one or several independent processes, called nodes, which perform computations. To enable communication between nodes, each node subscribes any number of needed topics, i.e. inputs, and may publish any number of topics, i.e. outputs.

Figure 2.1 illustrates a system consisting of two nodes and a single topic. "NodeA" subscribes to nothing, but publishes a single topic. "NodeB" subscribes to the same topic, but does not publish any new topic. Other than the shared content of the topic, the nodes are ignorant of each other.

In general, a topic might subscribe and publish several topics. Figure F.1 and F.2 displays the node/topic structures for the system designed in this thesis running during experimental results and in the simulator, respectively.

Tutorials for beginners in ROS are presented in Appendix D

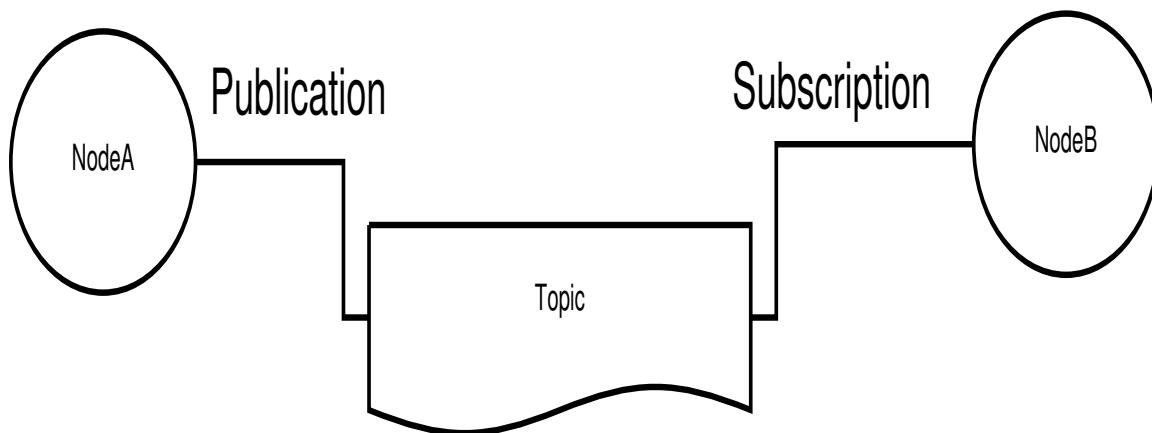


Figure 2.1: ROS publisher/subscriber architecture. Courtesy Ueland (2016)

### 2.1.3 Software architecture

Figure 2.2 gives a general overview of the signal flow between components. In general, it applies to the system flow of the CS Saucer, and other vessels set up in a similar manner.

A RP2 serves as the onboard computer running Linux as operating system with ROS installed. The Arduino has a Flash memory and runs an uploaded code, often called the Arduino sketch, it is run a separate ROS node, "/serial\_node" in Figure F.2. It is responsible for controlling the actuators, both angle, and thrust, but also the proximity sensors.

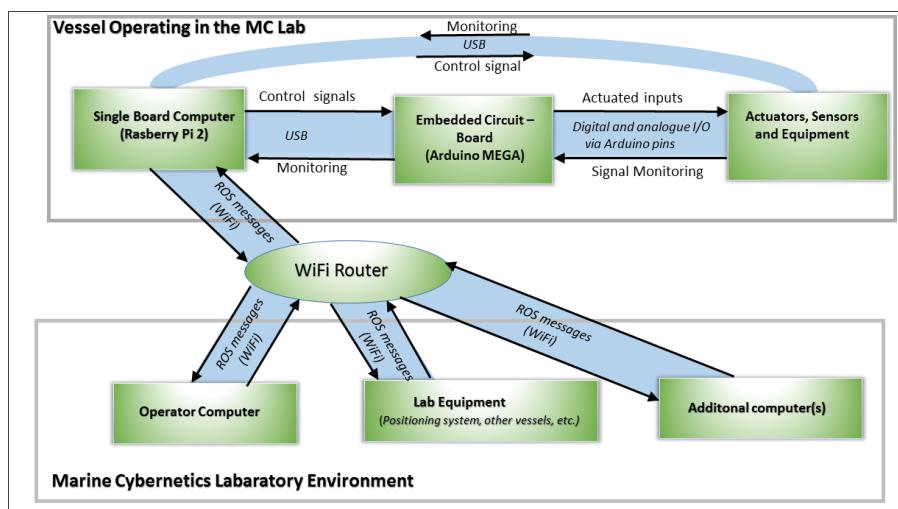


Figure 2.2: Signal flow between components on a system with the suggested architecture, set up in the MCLab. Courtesy Ueland

## 2.1.4 Hardware architecture

### 2.1.4.1 Breadboard

A 170 point breadboard with 17 rows and 10 columns, typically called the "mini breadboard", is used where column A-E and F-J are electrically connected. The breadboard is used with the acoustic sensors to supply 5V and connect them to ground. See Figure 2.3

### 2.1.4.2 Arduino Mega

The Arduino Mega is a microcontroller board based on the ATmega1280 (*Arduino Webpage*, 2016). It is a worldwide popular low-cost embedded circuit where open source code and drivers are easily available. The board has 54 digital pins, where 14 may be used as Pulse Width Modulation (PWM) outputs.

In the implemented system the four digital pins are responsible for triggering the acoustic sensors, while four PWM outputs register the response from each sensor. Another six PWM outputs control the angle and revolution of each servo and motor, respectively. Transmitted signals have a frequency of 50Hz. For a full overview of the setup see Table 2.1.

### 2.1.4.3 Raspberry Pi 2

The Raspberry Pi 2 (RP2) is a single-board computer. It is installed with Ubuntu, and is running multiple ROS nodes during experiments. The RP2 is connected via USB to the Arduino and lidar,

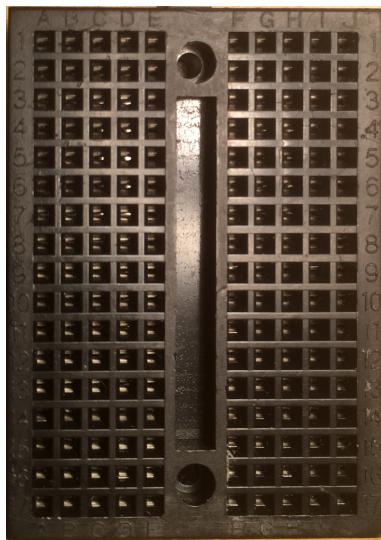


Figure 2.3: A 170 point breadboard.

it is also connected to the WiFi through a wireless USB adapter.

#### 2.1.4.4 Motors and servos

Three azimuth thruster are installed, each of which is driven by a separate Torpedo 800 motor. The azimuth thrusters can all be rotated by the motion from a servo; each motor has its corresponding servo of the type Graupner Schottel drive unit II. The arrangement is unaltered from the original setup, designed by Idland (2015).

#### 2.1.4.5 RPlidar

The lidar installed on the vessel is of the type PRlidar, which is among the cheapest on the market, under \$400 at *Robotshop* (2016). Commercially it has been replaced by its successor, the RPlidar A2 (\$450). The rotational speed can be set 2-10Hz, but the sampling frequency is fixed 2 kHz. The range of the scan is 6 meters, which is almost the same as the width of the MCLab, see Section 2.2. The lidar is responsible for generating a point cloud of the horizontal plane, corresponding to different attack angles. This data is subsequently processed and utilized by the implemented software system.

The lidar is placed on top of a small pasteboard box elevating it 10 cm above the lid, this was

Table 2.1: Pin connection overview

Pin	Description	Type
3	Angle Servo-1	PWM (Output)
5	Angle Servo-2	PWM (Output)
6	Angle Servo-3	PWM (Output)
7	Acoustic sensor-1	PWM (Output)
8	Acoustic sensor-2	PWM (Output)
9	Revolution Speed Motor 1	PWM (Output)
10	Revolution Speed Motor 2	PWM (Output)
11	Revolution Speed Motor 3	PWM (Output)
12	Acoustic sensor-3	PWM (Output)
13	Acoustic sensor-4	PWM (Output)
5V	Supply for acoustic sensors	Power
DGND	Ground Servos and motors	Ground
GND	Ground Battery/Acoustic Sensors	Ground
D50	Trigger Acoustic sensor-1	Digital
D51	Trigger Acoustic sensor-2	Digital
D52	Trigger Acoustic sensor-3	Digital
D53	Trigger Acoustic sensor-4	Digital



Figure 2.4: The HC-SR04 acoustic sensor. Courtesy DealExtreme

done since many cables are drawn from the ship interior to the proximity sensors placed on the hull (see Figure 2.5b), but also to ensure some obstacles was only detectable by the acoustic sensors. The elevation creates a horizontal plane where the lidar scans, approximately 20 cm above the water surface. The lidar is connected to the Raspberry Pi 2 over USB and is interfaced to the ROS framework through a separate node, ”/rplidarNode” in Figure F.2.

#### 2.1.4.6 The ultrasonic ranging module HC-SR04

The proximity sensor used in this thesis is the ultrasonic ranging module "HC-SR04", see Figure 2.4. The sensor contains 4 pins labeled "Vcc", "Trig", "Echo" and "Gnd". "Vcc" is the 5V DC power input pin. "Trig" is the pin to trigger the acoustic beam, it receives high or low and can be triggered by a regular digital port from the Arduino Mega. "Echo" is the output port, it outputs a PWM signal and hence needs to be connected to a unique PWM port on the Arduino. "Gnd" refers to ground. Other important data is listed in Table 2.2.

Four low-cost ( \$2) acoustic sensors were implemented. The Arduino code, in Appendix B.5.4, is set up such that it outputs a vector of six elements. The last two are zero, hence two additional

Table 2.2: Datasheet, HC-SR04

Voltage (Vcc)	DC 5V
Current	15mA
Frequency	40kHz
Range	2cm-4m
Measuring angle	$\pm 15^\circ$
Resolution	3mm
LWD	45x20x15mm

sensors could easily be implemented. The sensors are allowed to have a varying pitch and yaw angle; however, this was not utilized, see Section 7.2.

#### 2.1.4.7 Battery

Two 6400mAh, 11.1V, Lithium polymer (LiPo) batteries were utilized during the experimental period. A battery supplies the Raspberry Pi for about 12h, running the thrusters reduces the duration to about 4 hours; however, charging takes less than an hour so batteries can be changed often. Fully charged the voltage is 12.5V, it is recommended to check the voltage frequently and charge it when the voltage is close to 11.1V. The new batteries have the balance wires integrated with the connector, hence it is not possible to monitor the voltage as previously done by Ueland (2016).

#### 2.1.4.8 Laptop

A laptop also running ROS is connected through WiFi to the system. The laptops purpose is to establish a Human Machine Interface (HMI) where the operator may select desired areas to investigate, but also monitor the process. Due to limited process power on the RP2 it runs a ROS node responsible for generating paths for the system to follow.

During exploration a second computer, connected to the ROS network, runs the motion controller node, see Figure F.1, this is beneficial since it lightens the RP2 workload, but also adds safety in case of unexpected behavior.

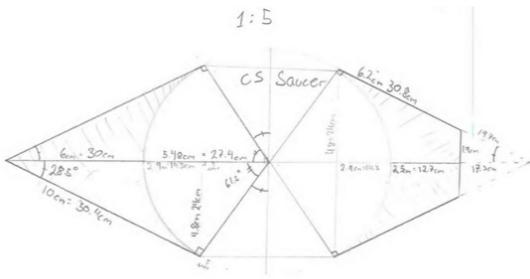
A stronger machine than the RP2 could run the entire, C++ compiled, this requires to modify and generate C++ code of the exploration node.

#### 2.1.4.9 Modules for increasing yaw resistance

Ueland (2016) wrote:

*The ability of the vessel to efficiently maneuver in both surge and sway is an advantage for this project, as it means that the vessel's heading does not need to be considered as a parameter in the path-planning process.*

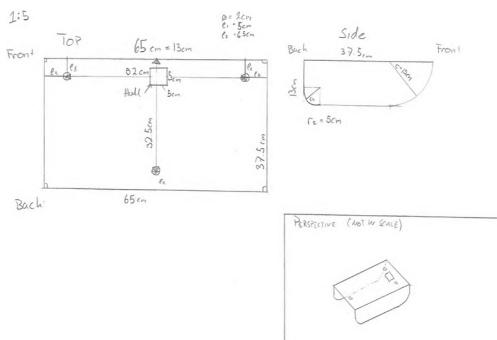
However, when considering auto-docking, heading needs to be considered, and a steering law is described in Chapter 5.2.1. To achieve this two different, physical, modules have been developed with the purpose of altering the behavior in sway and yaw for CS Saucer through altering the mass, waterplane area and resistance. Figure 2.5a and 2.5b shows the first suggestion where blocks of divinycell H60 have been fitted to the front and stern of the vessel. Since the density



(a) Sketch, module 1



(b) Result, module 1



(c) Sketch, module 2



(d) Result, module 2

Figure 2.5: Two different approaches to making the vessel behave more like a leisure boat, increasing sway and yaw resistance as a goal.

is very low ( $60 \text{ kg/m}^3$ ) the front and stern piece need to be loaded with 2.5 kg each, to get a sufficient draught.

Figure 2.5c and 2.5d shows the second suggestion. Its mass is 2.54 kg, and a weight of 1 kg needs to be placed in the bow to counteract the moment. The plate is designed with constrained Thrust Allocation in mind. It covers the stern half of the vessel such that the wake from the bow thruster, which is in a  $90^\circ$  degree position, is unaffected. The stern thrusters may rotate from pointing backward, with constraints; however, the plate will not interfere with their wake. Thrust allocation is further described in Chapter 4.2.

## 2.2 Marine Cybernetic Laboratory

The Marine Cybernetics Laboratory (MCLab) basin, installed with varying obstacles, constitutes the environment of operation for the CS Saucer. The lab is suited for testing of small marine vessels, equipped with a wave maker, a towing cart and a positioning system, Qualisys. Figure 2.6 shows a picture of the basin. The MCLab dimensions (LxWxD) are 40 m x 6.45 m x 1.5 m (*IMT, NTNU, 2016*).

### 2.2.1 Obstacles

Figure 2.6 shows the MCLab with the objects present. In the far end plastic and pasteboard bags create a wall, this is to reduce the available area of the basin, but also enables other experiments to be run on the other side. The author often shared the lab with others, hence compromises were made. The red boat on the left side served as another obstacle, but also to make the basin more narrow, such a large object are quickly recognizable on the generated map, see Section 6.2.2, which makes it easier to orient in. The small model boat on the right side is one of four used during testing. Two of the boats have led weights in them, this is to keep the entire model below the plane scanned by the lidar, hence only detectable by the proximity sensors. The last two was stacked on top of each other to be clearly viable for the lidar



Figure 2.6: Marine Cybernetics Laboratory, along with obstacles used in this thesis.

# Chapter 3

## Mathematical model

The mathematical modeling of the system will be separated into two, kinematics and kinetics. The former describes the geometrical aspect of a dynamic system, how to describe motion in different frames. The latter handles forces on the system, both external, environmental, forces and forces generated by the thrusters.

### 3.1 Kinematics

There are four different reference frames used in this thesis:

- Basin-fixed reference frame, Figure 3.1a
- Basin-relative reference frame, Figure 3.1a
- Body-fixed reference frame, Figure 3.1b
- Hector-SLAM generated frame.

#### 3.1.1 The Basin-fixed and Basin-relative reference frames

The Basin-relative reference frame is realigned relative to the basin for each new trial. This is caused by how Hector-SLAM initializes the coordinate system it represents the map in. The angular rotation between the Basin-relative and the Basin-fixed reference frame is denoted by  $\Delta\psi_{\text{rel}}$ , see Figure 3.1a. When the Hector-SLAM is initialized, the positive x-axis of the Basin-relative reference frame is aligned with the x-axis of the lidar, i.e. the Hector-SLAM generated frame. Their origin is also equal at initialization; however, in the Hector-SLAM frame, the z-axis is pointing upwards, whereas it is pointing downwards in the Basin-relative frame. This

difference causes that positive y-coordinates and a positive yaw angle in the Basin-relative frame becomes negative in the Hector-SLAM frame. The transformation to align the two frames was solved by Ueland (2016) through the ROS-node "Hector2VesselPos-node".

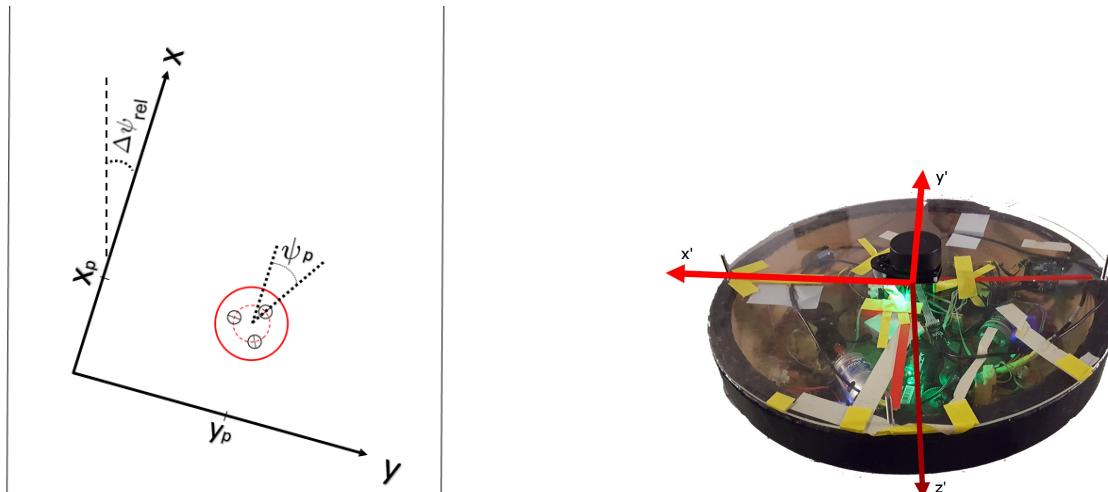
Since the vessel is round, no clear bow can be identified, the heading of the vessel have thus been defined as zero when thruster 1 is pointing along the x-axis relative to the center of origin of the vessel. It is crucial that the modules, which have a distinct bow, are in alignment with the definition. The z-axis is pointing downwards, while the heading is defined as positive in the clockwise direction. The positions  $x$  and  $y$  and heading  $\psi$  in the Basin-relative frame in a vectorial format is given as follows:

$$\eta = [x \ y \ \psi]^T \quad (3.1)$$

In Figure 3.1a the vessel is illustrated in the Basin-relative frame. The edges mark the basin edges, which are parallel to the x-axis of the Basin-fixed reference frame. In the example, the vessel has moved from the origin, i.e. where it was initialized, to the position  $\eta_p = [x_p \ y_p \ \psi_p]^T$

### 3.1.2 Body-fixed reference frame

When describing local behavior of the vessel, it is convenient to have a frame fixed to the vessel body, which is the Body-fixed reference frame, see Figure 3.1b. The axes in the Body-fixed reference frame are denoted  $x'$  and  $y'$ , while the velocities are denoted  $u$  along the  $x'$  axis,  $v$  along the  $y'$  axis, and  $r$  for the angular velocity about the  $z'$ -axis.



(a) Basin-relative reference frame. Borders illustrate basin edges (the Basin-fixed frame).  
Courtesy Ueland

(b) Body-fixed reference frame. It is important to align the lidar with the x-axis.  
Courtesy Ueland

Figure 3.1: Basin-relative and Body-fixed reference frame are the most used throughout this thesis.

Linear velocities  $u$  and  $v$ , and angular velocity  $r$  in a vectorial format for the Body-fixed reference are given as follows

$$\nu = [u \ v \ r]^T \quad (3.2)$$

### 3.1.3 Transformation between reference frames

To establish a relationship between the above-mentioned reference frames, one needs to be able to transform from one coordinate system to another. The transformation frequently used in this thesis is between the Body-fixed reference frame and the initialized Basin-relative reference frame. The system is considered in surge, sway and yaw are described, while heave, roll, and pitch are neglected, i.e., the 3DOF where it is assumed that the CS Saucer is self-stabilizing in heave, roll and pitch, due to hydrostatic forces. This leads to a transformation from one frame to another to only be dependent on the yaw angle. The relationship can be expressed as:

$$\dot{\eta} = \nu^s = R(\psi)\nu^b \quad (3.3)$$

where  $\nu^b$  is the velocity in the Body-fixed Frame,  $\nu^s$  is the corresponding velocity in the Basin-relative frame, and the transformation matrix  $R(\psi)$  is given by,

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

To transform the other way, the inverse  $R^{-1} = R^T$  is used.

## 3.2 Kinetics

In Ueland (2015) the CS Saucer was modeled to find an adequate equation of motion. The model is essential for the simulation model developed in Ueland (2016), but also in the observer that is implemented into the motion control system. The 3DOF maneuvering equation of motion for a general vessel can be represented by the following equation: (Fossen, 2011, Eq. 6.2)

$$M_{RB}\dot{v} + C_{RB}(v)v + M_A\dot{v}_r + C_A(v_r)v_r + D(v_r)v_r + g(\eta) + g_0 = \tau_{\text{external}} \quad (3.5)$$

where,

$v$  is the Body-fixed velocities in surge, sway and yaw.

$v_r$  is the Body-fixed velocities relative to local current in surge, sway and yaw. ( $v=v_r$  since the current is set to zero)

$M_{RB}$  and  $M_A$  are the inertia matrices for the rigid body and the added mass

$C_{RB}(v)$  and  $C_A(v_r)$  are the Coriolis centripetal matrices for the rigid body and the added mass

$D(v_r) = D + D_n(v_r)$  (Fossen, 2011, Eq. 6.57) is the damping matrix, consisting of the linear and nonlinear damping matrices.

$g(\eta)$  is a vector of gravitational/buoyancy forces and moments

$g_0$  is a vector used for pretrimming (here  $g_0 = 0_{3x1}$ )

$\tau_{\text{external}} = \tau = [X \ Y \ Z]^T$  is the external forces acting on the vessel excluding those mentioned above, in this thesis is will be the thrust vector generated from the three thruster, which will be discussed in Chapter 4.

Since there are no restoring forces in surge, sway and yaw  $g(\eta) = 0_{3x1}$ . This results in the follow-

ing system matrices (Ueland, 2015):

$$M = M_{RB} + M_A = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & 0 \\ 0 & 0 & I_z - N_{\dot{r}} \end{bmatrix} \quad (3.6a)$$

$$C(v) = C_{RB}(v) + C_A(v) = \begin{bmatrix} 0 & -1.5mr & 0 \\ 1.5mr & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.6b)$$

$$D = - \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & 0 \\ 0 & 0 & N_r \end{bmatrix} \quad (3.6c)$$

$$D(v) = - \begin{bmatrix} X_{u|u||u|} & 0 & 0 \\ 0 & Y_{v|v||v|} & 0 \\ 0 & 0 & N_{r|r||r|} \end{bmatrix} \quad (3.6d)$$

For the resulting numerical values of the CS Saucer see Ueland (2016). In Section 2.1.4.9 modules for altering the resistance in yaw was proposed, to ease the development of a steering law, but also to increase the resistance in sway. Specifically the following parameters have been altered:  $m$ ,  $I_z$ ,  $Y_{\dot{v}}$ ,  $N_{\dot{r}}$ ,  $Y_v$ ,  $N_r$ ,  $Y_{v|v|}$ ,  $N_{r|r|}$ , while  $X_{\dot{u}}$ ,  $X_u$ ,  $X_{u|u|}$  was unaltered. The mass was altered to  $m = m_v + m_e$ , i.e. the mass of the module was added to the vessels mass. Table 3.1 sums up all parameters.

Note that the added mass and damping matrix, in general, will depend on frequency. However, in this thesis, it is assumed constant for all frequencies.

It is emphasized by the author that the new values have not been found through any methods of identification, e.g. towing, as was done in Idland (2015). They were chosen such that the simulation model behaved as expected when a steering law was under development.

Table 3.1: Parameters used in the maneuvering equation

Mass and inertia related		Drag related	
$m_v$	6.34	$X_u$	-1.96
$m_e$	5/3.5kg	$Y_v$	-4
$I_z$	0.2	$N_r$	-1.4
$X_{\dot{u}}$	-3.5	$X_{u u }$	-7.095
$Y_{\dot{v}}$	-6.5	$Y_{v v }$	-14
$N_{\dot{r}}$	-0.1	$N_{r r }$	-5



# Chapter 4

## Control allocation

In this chapter thrust allocation using fixed and rotatable angles is studied. In the first case the goal is to find the force for the thrusters,  $f$ , from the desired  $\tau$  given by (3.5). In the latter case, the constrained control allocation for azimuth thrusters will be applied to the two aft thrusters to find thrust and angle. The thrust configuration matrix,  $T(\alpha)$ , relates the forces from the thrusters through the generalized force relationship  $\tau = T(\alpha)f$ .

Figure 4.1 will be used throughout this chapter; it illustrates the thruster configuration on the CS Saucer. The shaft of the thrusters are all located on the circumference of a circle with radius  $r_t = 0.138m$ , and they are placed with  $120^\circ$  between them, i.e. symmetrically. In the general case:

$$\tau = \begin{bmatrix} c(\alpha_1) & c(\alpha_2) & c(\alpha_3) \\ s(\alpha_1) & s(\alpha_2) & s(\alpha_3) \\ r_t(c(\chi_1)s(\alpha_1) - s(\chi_1)c(\alpha_1)) & r_t(c(\chi_2)s(\alpha_2) - s(\chi_2)c(\alpha_2)) & r_t(c(\chi_3)s(\alpha_3) - s(\chi_3)c(\alpha_3)) \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad (4.1)$$

where  $\chi_i$  is the position of each thruster, i.e.  $[\chi_1, \chi_2, \chi_3] = [0, -120^\circ, 120^\circ]$ ,  $f_i$  and  $\alpha_i$  is the force and its angle, respectively,  $c(\cdot)$  and  $s(\cdot)$  represent  $\cos(\cdot)$  and  $\sin(\cdot)$ , respectively.

In the introduction, Section 1.2.1, the definition of under-, fully- and overactuated was presented. If the thrusters on CS Saucer are fixed, with singularity avoidance in mind, the vessel becomes fully actuated with  $r = n = 3$ . When an actuator can be rotated, without constraints, it is analogous to an additional control variable;  $r$  becomes  $r + p$  where  $p$  denotes the number of rotatable actuators. If the three thrusters could rotate it would result in an overactuated system with  $r = 6$ ; however, in this thesis, the bow thruster (thruster 1) is fixed, and the stern thrusters

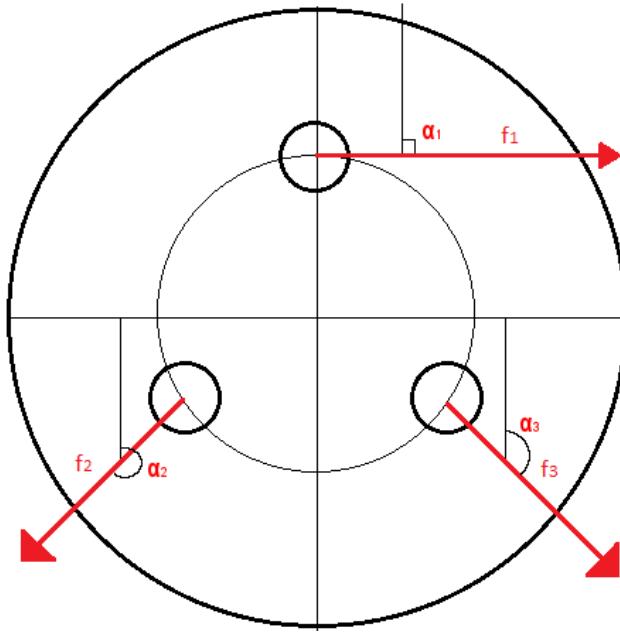


Figure 4.1: CS Saucer setup.  $\alpha_1 = 90^\circ$ ,  $\alpha_2$  and  $\alpha_3$  are fixed in Section 4.1, and rotatable in 4.2

(thruster 2 and 3) may rotate 45° to either side. Since both negative and positive thrust can be applied, the thruster are constrained from producing a force vector in 180 out of 360 degrees. In these cases,  $p$  is therefore defined as 1/2. Later four configurations will be presented, following the definition of  $p$ , which is overactuated, fully-actuated, and the last two underactuated with an  $r$ -value of 4, 3, 2.5 and 1.5, respectively.

## 4.1 Fixed angles

According to Ueland (2016) the thrust angles where fixed to maximize the yaw moment of each thruster, i.e.  $\max(r_t(c(\chi_i)s(\alpha_i) - s(\chi_i)c(\alpha_i)))$ . This is achieved by selecting  $\alpha_i = \chi_i + 90^\circ$ , which is to align them tangentially to a circle with center in origin. Resulting in:

$$\begin{aligned}\alpha_1 &= 90^\circ \\ \alpha_2 &= -30^\circ \\ \alpha_3 &= -150^\circ\end{aligned}\tag{4.2}$$

The thrust in the body-frame,  $\tau$ , of the vessel becomes:

$$\begin{aligned}\tau &= Tf \\ &= \begin{bmatrix} 0 & \cos(-30^\circ) & \cos(-150^\circ) \\ 1 & \sin(-30^\circ) & \sin(-150^\circ) \\ r_t & r_t & r_t \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}\end{aligned}\tag{4.3}$$

Since T is square  $f$  is found from taking the inverse:

$$f = T^{-1}\tau\tag{4.4}$$

## 4.2 Constrained rotatable thrusters

The research done in this thesis on TA is inspired by Frederich (2016). This section uses Frederich (2016) results, on a vessel with 6 azimuth thrusters, fitting it to the CS Saucer. The aim is to study four different thrust configurations to resemble an underactuated leisure boats. A typical leisure boat has one or two main thruster, and possibly a bow thruster, see Figure 4.2.

Four configurations will be studied:

1. Two main thrusters and one bow thruster
2. Two main thrusters
3. One main thruster and one bow thruster
4. One main thruster (thruster 2 and 3 aligned)

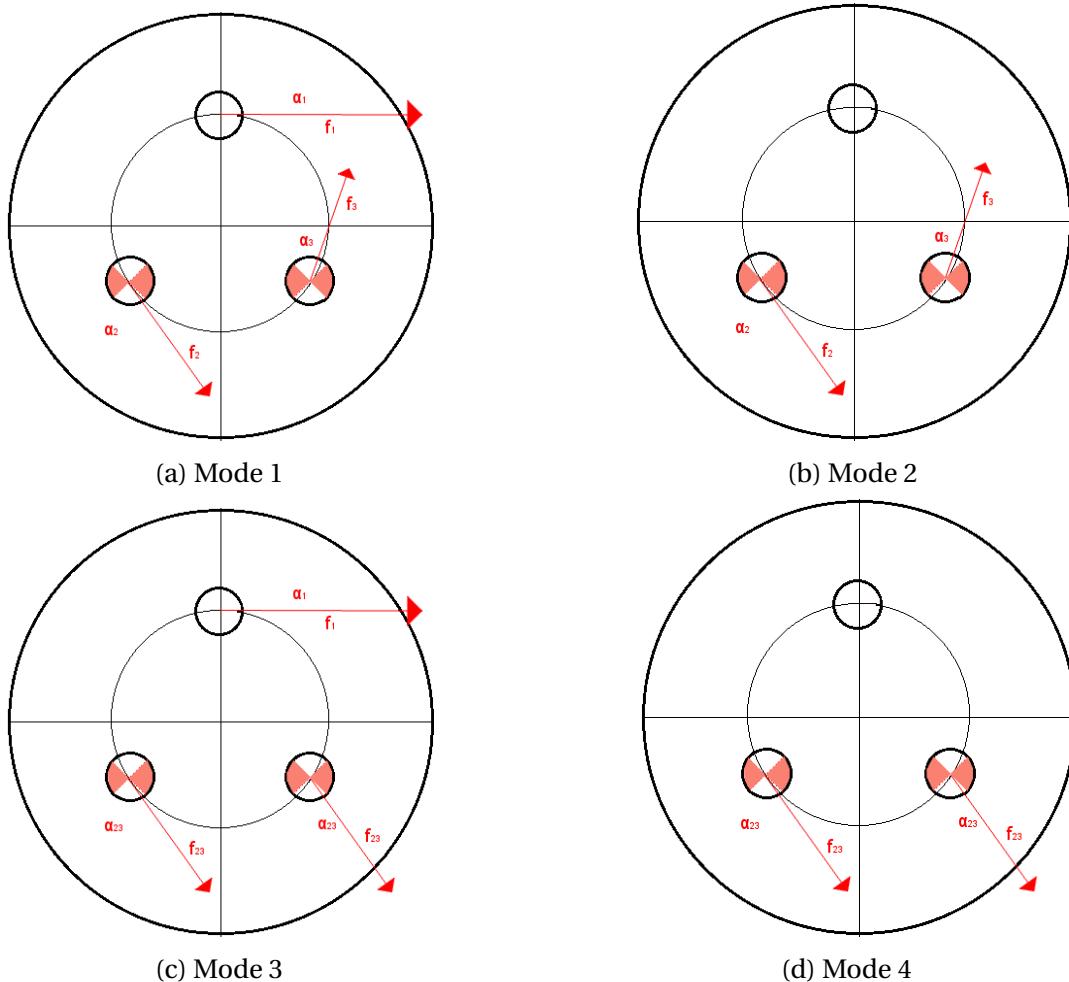


Figure 4.2: The four modes, showing dead zones for the stern azimuth thrusters. Note that  $f_3$  in Mode 1 and 2 is negative,  $\alpha_3 \approx 145^\circ$

An outboard motor could be modeled as a turntable, constrained, azimuth thruster. Typical maximal deflection is  $45^\circ$ ; hence its dead zones would be from  $-135^\circ$  to  $135^\circ$ . The bow thruster is modeled as a fixed thruster. To solve the problem with a fixed step size in Simulink one could use an iterative solution of quadratic programming, as suggested in Fossen (2011) and Johansen et al. (2004). The problem can be solved by using the inbuilt, Mathworks, function *quadprog.m*. The function attempts to solve the quadratic problem :

$$\begin{aligned} \min_x \quad & 0.5x^T Hx + q^T x \\ \text{subject to} \quad & Ax = b \\ & x_{min} < x < x_{max} \end{aligned} \quad (4.5)$$

Where  $A = [T_e - I_{3x3}]$ ,  $b = \tau$ ,  $x = [f_e \ s]^T$ ,  $H$  is a weighting matrix and  $q$  is a zero vector.  $T_e$  is the extended thrust configuration matrix to avoid the nonlinearities in the control of  $\alpha$  in  $T(\alpha)$ .  $f_e = [f_1 \ f_{2x} \ f_{2y} \ f_{3x} \ f_{3y}]$  is the extended force vector where  $\sqrt{f_{ix}^2 + f_{iy}^2} = f_i$  and  $\alpha_i = \text{atan2}(f_{iy}, f_{ix})$ . From (4.5):

$$\begin{aligned} Ax &= b \\ [T_e - I_{3x3}][f_e \ s]^T &= \tau \\ T_e f_e &= \tau + s \end{aligned} \quad (4.6)$$

, where  $s$  is a slack variable to allow  $T_e f_e$  to differ from  $\tau$ , the cost of using  $s$  is weighted through  $H$ . Typically, the weighting of  $s$  versus  $f_e$  is of order 1000, represented as  $k$  in (4.7) - (4.11) below.

### 4.2.1 Configuration 1

The first configuration presented above yields the highest controllability of the four considered configurations. It is common for leisure boat that the angle of two aft thrusters is controlled identically. Usually, the motors are mounted on the same shaft, thus given the same deflected, and outputting the same thrust. If they could be controlled independently, the distance between them should be as large as possible to minimize thruster interaction and maximizing yaw moment. In most cases, though not in the CS Saucers case, they are very close to each other.

With configuration 1 in mind and looking at Figure 4.2a above, Equation 4.1 becomes:

$$\begin{aligned}\tau &= \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ l_{x1} & -l_{y2} & l_{x2} & -l_{y3} & l_{x3} \end{bmatrix} \begin{bmatrix} f_1 \\ f_{2x} \\ f_{2y} \\ f_{3x} \\ f_{3y} \end{bmatrix} \\ H &= \begin{bmatrix} I_{5x5} & 0 \\ 0 & kI_{3x3} \end{bmatrix} \\ q &= 0_{8x1}\end{aligned}\tag{4.7}$$

where the extended notation  $T_e$  and  $f_e$  is used and:

$$\begin{aligned}l_{x1} &= r_t = 0.138m \\ l_{x2} &= r_t \cos(-120^\circ) = -0.069m \\ l_{y2} &= r_t \sin(-120^\circ) = -0.1195m \\ l_{x3} &= r_t \cos(-120^\circ) = -0.069m \\ l_{y3} &= r_t \sin(-120^\circ) = 0.1195m\end{aligned}\tag{4.8}$$

## 4.2.2 Configuration 2

Without the bow thruster  $f_1$  is removed from the equation, resulting in:

$$\begin{aligned}\tau &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -l_{y2} & l_{x2} & -l_{y3} & l_{x3} \end{bmatrix} \begin{bmatrix} f_{2x} \\ f_{2y} \\ f_{3x} \\ f_{3y} \end{bmatrix} \\ H &= \begin{bmatrix} I_{4x4} & 0 \\ 0 & kI_{3x3} \end{bmatrix} \\ q &= 0_{7x1}\end{aligned}\tag{4.9}$$

## 4.2.3 Configuration 3

To achieve the behaviour of one stern thrusters the two aft thruster could be superposed to one, given that their deflection and output is equal. The superposed thruster will be on the x-axis

since  $l_{y2} = -l_{y3}$ . The extended notation becomes:

$$\begin{aligned}\tau &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ l_{x1} & 0 & l_{x23} \end{bmatrix} \begin{bmatrix} f_1 \\ f_{23x} \\ f_{23y} \end{bmatrix} \\ H &= \begin{bmatrix} I_{3x3} & 0 \\ 0 & kI_{3x3} \end{bmatrix} \\ q &= 0_{6x1}\end{aligned}\tag{4.10}$$

Where  $f_{2x} = f_{3x} = 0.5f_{23x}$ ,  $f_{2y} = f_{3y} = 0.5f_{23y}$  and  $\alpha_2 = \alpha_3 = \alpha_{23}$

#### 4.2.4 Configuration 4

Analogous as with configuration 2,  $f_1$  is removed, yielding the least controllability of the vessel.

$$\begin{aligned}\tau &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & l_{x23} \end{bmatrix} \begin{bmatrix} f_{23x} \\ f_{23y} \end{bmatrix} \\ H &= \begin{bmatrix} I_{2x2} & 0 \\ 0 & kI_{3x3} \end{bmatrix} \\ q &= 0_{5x1}\end{aligned}\tag{4.11}$$

#### 4.2.5 Simulink implementation

The quadratic optimization from Frederich (2016) was altered to four versions complying with 4.7 - 4.11 above. Creating a subsystem of each configuration, a simple scripts enables the correct subsystem corresponding to the selected mode.

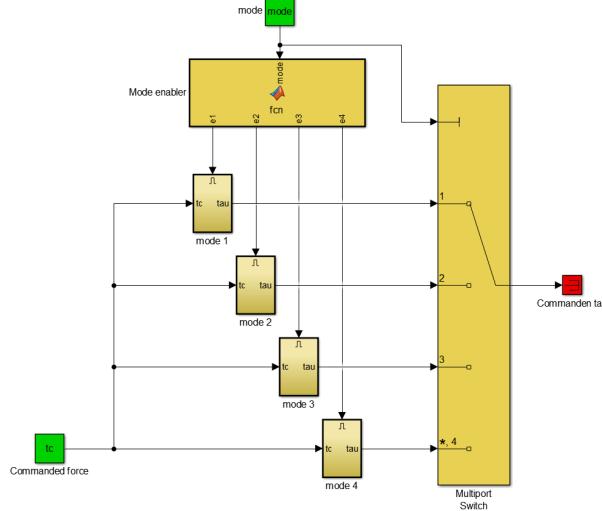


Figure 4.3: Top level of simulink model. Mode 1-4 enables the correct subsystem.

### 4.3 Simulations

The four different constrained rotatable thruster configurations were implemented in Simulink (Figure 4.3). The system was initialized with the thrusters at rest with  $\alpha_1 = -90^\circ$  (fixed),  $\alpha_2 = 8^\circ$  and  $\alpha_3 = 5^\circ$ , see Figure 4.1. In the case of mode 3 and 4,  $\alpha_{23} = 5^\circ$ . The desired thrust was set to  $\tau_d = [1, 0, 0]$ , i.e. a pure surge force. The result of mode 1-4 can be seen in Figure 4.4a-4.4d, respectively. The Figures have a y-axis on the right related to the angles of the thrusters. In all the cases one sees that thruster 1, if present, is fixed at  $-90^\circ$  while thruster 2 and 3 turns toward  $0^\circ$ . The left axis is related to the desired thrust,  $\tau_d$ , the commanded thrust from the TA,  $\tau_c$  and the produced thrust from each motor,  $f$ . In all the modes thruster 1, if present, is not used since it only produces coupled sway and yaw forces. In mode 1 and 2, Figure 4.4a and 4.4b, the force of  $f_2$  and  $f_3$  both converges to 0.5 with an overshoot, superposed they produce the desired surge force. In mode 3 and 4, Figure 4.4c and 4.4d,  $f_{23}$  converges 1, this also fits well with what was described in Section 4.2.3-4.2.4 since  $f_{23}$  is the superposed thruster of 2 and 3, i.e.  $f_2 = f_3 = 0.5f_{23}$ . As the motors produce their thrust, the angles have not converged to zero yet, this results in a sway force, and a hardly noticeable yaw-moment, these goes to zero along with the angles. See the dashed blue lines.

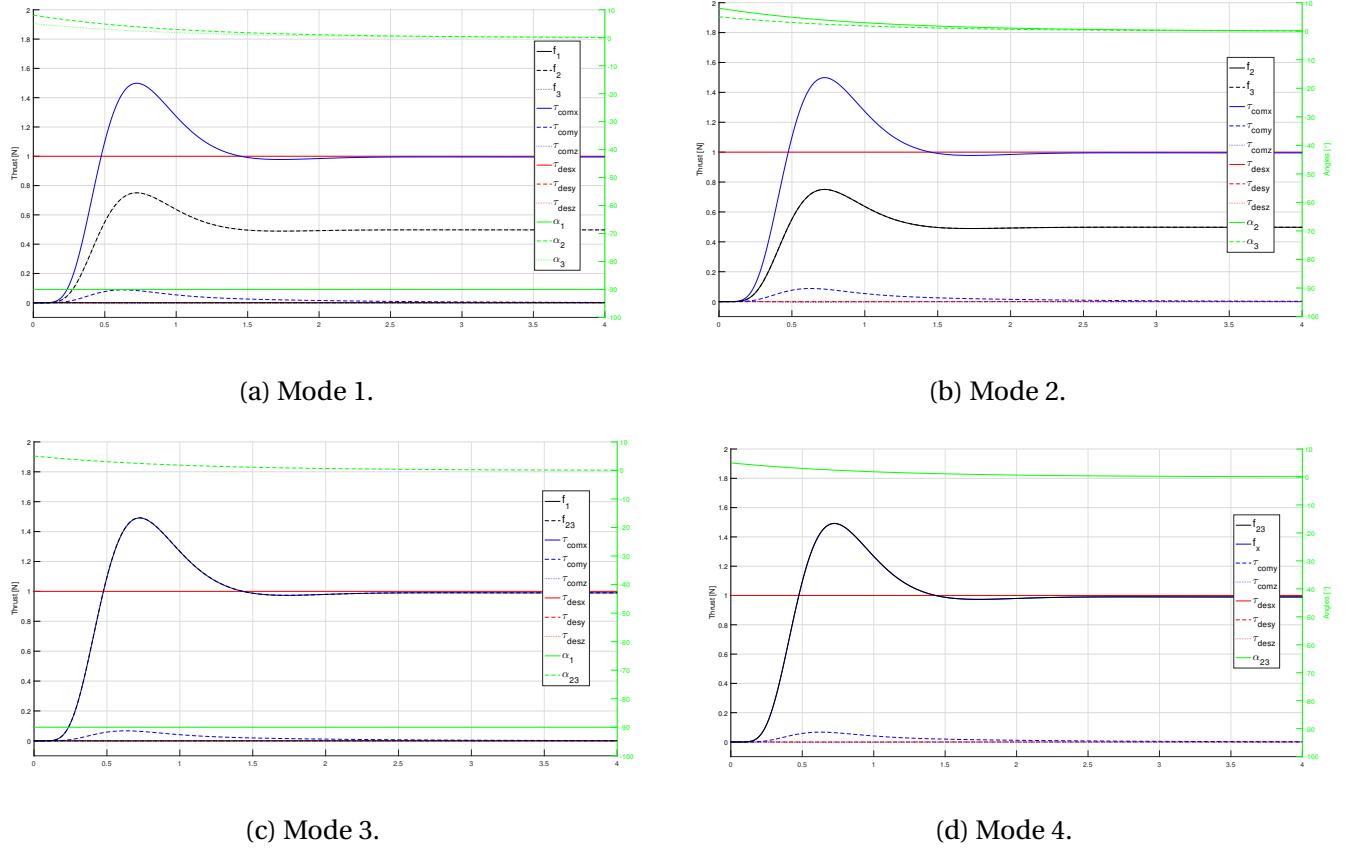


Figure 4.4: Mode 1-4 trying to produce  $\tau_d = [1, 0, 0]$ . Left y-axis is thrust [N], right are angles [degree] and the x-axis is in seconds.

It is possible to run a model as a node directly from Simulink; hence it was possible to test the quadratic solver in the simulator, bypassing the obstacles of creating a C++ code, as mentioned by (Frederich, 2016). The control scheme was implemented into the motion control node in the ROS environment. It would also be possible to test it online (from a laptop) in the MCLab. Figure 4.5 shows how the model from Figure 4.3 is fitted into the "MotionControl" Node, the gray box, in the lower right, is the original TA made in Ueland (2016). The control scheme was tested; however, it proved to be too unreliable. If the quadratic solver could not find a solution, which happened in many cases, the output became zero, i.e. no thrust. In practice, it was shown that the implemented Quadratic Controller was not robust enough to handle more complex operation where the commanded thrusts were changing rapidly. For this reason, it was chosen to continue utilizing the fixed thruster setup, as presented in Section 4.1

From now on the setup with fixed angles developed in Ueland (2016), presented in Section 4.1 will be used, both in simulation and experimental results.

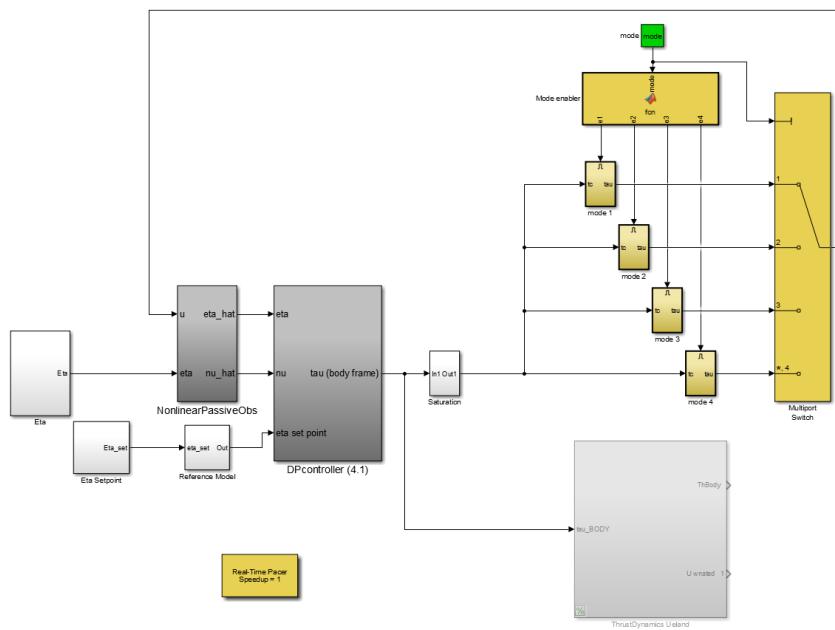


Figure 4.5: The "MotionControl" Node by Ueland (2016), where the Thrust Allocation is replaced with the constrained rotatable optimization.

# **Chapter 5**

## **Autonomous docking**

This chapter will discuss how the autonomous docking scheme was created through altering the works by Ueland (2016)

### **5.1 Problem statement**

The problem of autonomous docking is stated as:

- Given the CS Saucer which has the capabilities of simultaneous localization and mapping (SLAM) and autonomous exploration by using a 2D lidar, perform autonomous docking. The autonomous docking should emulate that of leisure boats, such that the thrust allocation and shape of the vessel need to be considered. In order to improve the autonomous capabilities of the CS Saucer proximity sensors, supporting the lidar sensor should be implemented.

The problem includes getting a detailed understanding of work done by Ueland (2016), altering the software and hardware architecture. In order to provide an answer, the following items should also be performed::

- Modify the online visualization of the exploration process to include obstacles found by the proximity sensors, also include vessel heading.
- Modifying the existing path planner and setpoint generator to perform docking and take-off

At the top of this thesis is description sheet formulated in cooperation with the supervisor, it introduces sub-objectives to accomplish the main objective, above.

## 5.2 Motion control system

This section presents the steering law added to the motion control system, made by Ueland (2016), it also presents how the PD-controller had to be altered as a consequence.

### 5.2.1 Steering law

In this section a steering law will be presented, which is essential for performing docking. In Ueland (2016) heading of the vessel was not an important factor, the vessel was circular and fully actuated, i.e., there was no difference moving in surge or sway; hence the heading was set to 0 for all time.

To achieve path following, a Line-of-Sight (LOS) steering law will be implemented. The algorithm limits the system to only see a certain radius around itself, following the path it sees. In Spange (2016) and Fossen (2011) the desired heading of the vessel was to always aim at a point in front the vessel,  $(x_{los}, y_{los})$ ,

$$\chi_d(e) = \alpha_k + \arctan\left(\frac{-e}{\Delta}\right), \quad (5.1)$$

where  $\Delta$  is the lookahead distance from the intersection between e and the path to  $(x_{los}, y_{los})$ . See Figure 5.1a. Ueland (2016) developed a velocity control law where he redisccretized the optimal path, identified the closest point on that path, but also found a setpoint for that the vessel should go to, see Figure 5.1b. The steering law uses the same setpoint as if it were  $(x_{los}, y_{los})$ , see Figure 5.1a. Simplifying the LOS algorithm with the the desired heading becoming:

$$\chi_d = \text{atan2}(P_d(2) - P_v(2), P_d(1) - P_v(1)) \in \mathbb{S} \triangleq [-\pi, \pi] \quad (5.2)$$

Where  $\text{atan2}(x,y)$  is the four-quadrant version of  $\arctan(y/x)$  defined as:

$$\text{atan2}(y, x) \triangleq \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0, \end{cases} \quad (5.3)$$

$P_d$  and  $P_v$  are the chosen setpoint and the vessel position, in the basin relative reference frame, respectively.

### 5.2.2 Docking and take-off

When the vessel is instructed to dock it follows the steering law, traveling along the planned path, until it is "sufficiently" close (an adjustable parameter) to its goal. The desired heading is changed to  $\psi_{\text{docking}}$ , as a step. The vessel will then rotate and glide the last distance into the dock.

Take-off is when the vessel leaves the dock, the operator selects any spot and the path planner ensures the vessel leaves the dock in a safe manner.

### 5.2.3 PD-controller

The control law that determines the forces in the Body-frame,  $\tau$ , is a PD-controller:

$$\tau = R(\psi)^T K_P (\eta_d - \hat{\eta}) - K_d \hat{v} \quad (5.4)$$

where  $\eta_d$  and  $\hat{\eta}$  is the desired and the observer estimated position in the Basin-relative reference frame, respectively; hence they need to be rotated with  $R(\psi)$ .  $\hat{v}$  is the observer estimated velocity, in the Body-frame. The control gains were altered too, with a heavier weighting on the heading. The surge and sway components were increased and decreased in the derivative and

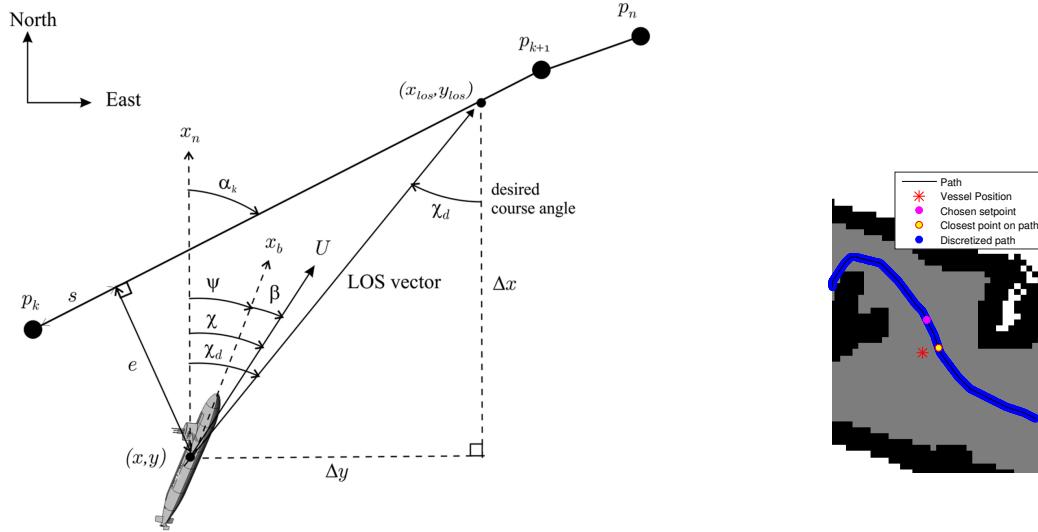


Figure 5.1: A LOS steering law is implemented, utilizing existing parameters from the path planner.

proportional gains, respectively. Resulting in

$$\begin{aligned} K_p &= 1.5I_{3 \times 3} \\ K_d &= 4I_{3 \times 3} \end{aligned} \tag{5.5}$$

## 5.3 Map processing

To achieve autonomous docking, the online map displaying the progress has to be altered. To visualize the heading, the vessel is plotted a patch with a distinguishable bow. In this section map processing will be presented, which includes a method for drawing an arbitrary environment for the simulator, interfacing the proximity sensors into the system, putting a cost on large heading changes and a new operator interaction where the user is asked if the vessel should dock or not when a location is selected.

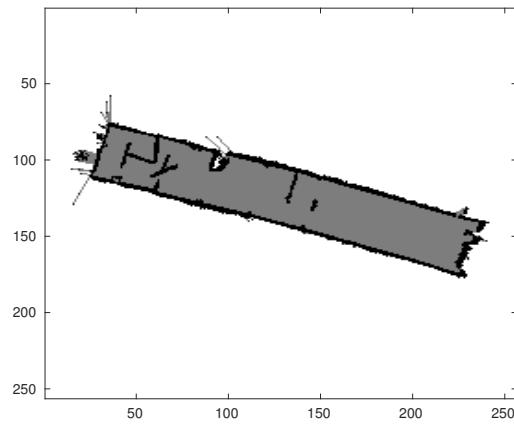
### 5.3.1 Draw test environment

In the simulations performed by Ueland (2016) a reference map, created by scanning the MCLab is deployed as a reference map for the simulator, see Figure 5.2a. The map is a  $256 \times 256$  matrix consisting of three entries:

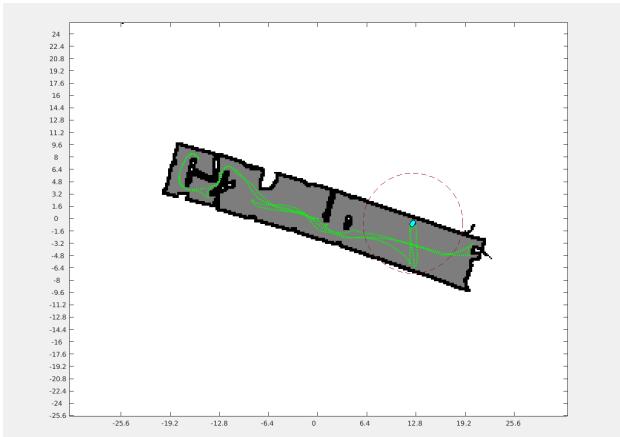
- 0: Unexplored region
- 1: Investigated and free cell
- 2: Investigated and occupied cell.

Moreover, each pixel represents an area, i.e., the resolution of the picture. In Figure 5.2b below the x range from  $-25.6$  to  $25.6$ , hence the resolution is  $0.2$ , meaning that one pixel represents  $0.2m^2$  in the real world.

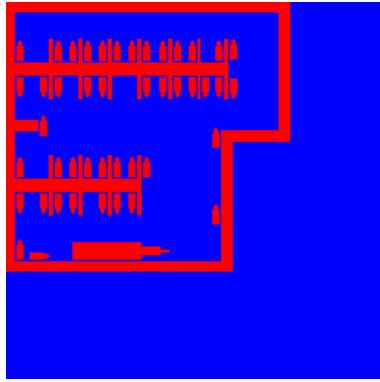
In the simulations to be performed in this thesis, it is desirable to deploy the vessel in a harbor like environment. For this reason a tool, allowing for arbitrarily design of the reference map, by the use of MS paint was created. A picture of resolution  $n \times m$  could be represented as a matrix  $[P] = [n, m, 3]$ , where the third dimension represents the amount of red, green and blue, respectively, ranging from  $0 - 255$ . Figure 5.2c is drawn in Paint, where red represent an occupied cell and blue a free cell. It is easiest to draw in the primary colors since a pixel would be represented as a vector of two zero entries and one 255, e.g. the red pixel  $(i, j)$  would be  $P(i, j, :) = [255, 0, 0]$ . A script was made in MATLAB to transform a the drawn map in Paint from a PNG file to a corresponding map matrix in MATLAB, see Appendix A.1.



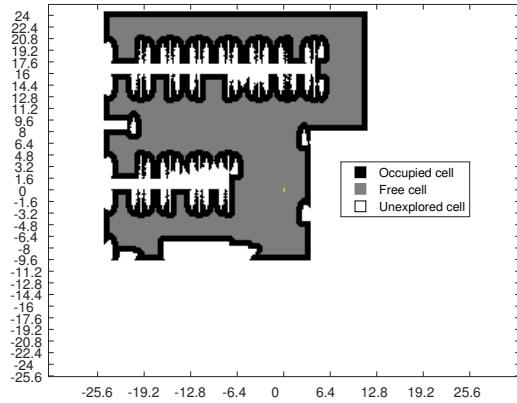
(a) The map scanned by the CS Saucer through lab experiments by Ueland (2016).



(b) For each simulation the CS Saucer is deployed in the origin and begins exploring.



(c) A figure drawn in Paint



(d) Resulting scan from simulation

Figure 5.2: White, gray/blue and black/red represent an unexplored, free and occupied cell, respectively.

### 5.3.2 Including the proximity sensors

In Section 2.1.4.6 the acoustic proximity sensor was presented. If all the obstacles where visible for the lidar there would be no need for any additional proximity sensors; however, installing a 3D-lidar or making several 2D-lidars cooperate are more expensive than adding to cheap acoustic sensors warning the system if they detect an obstacle. The analogy with autonomous docking is that the lidar is typically placed on top of the vessel to be able to get a 360° view, anything below it, e.g. smaller vessels or the quay, would not be detected, by a 2D-lidar. The proximity sensors will be placed on the outside of the vessel, above the water, to detect that the lidar is not able to identify. See Figure 2.5d, 6.3a, and 6.3b for pictures where the acoustic sensors are

mounted on the vessel.

As described above the map used is a visualization had three entries to describe the cell. An obstacle detected by the proximity sensor have been implemented as fourth entry. The logic implemented follows the following step:

1. if the measured distance is below 1m and greater than 15cm
2. locate the cell in the map, if it is free or unexplored
3. set the value of that cell to 3, which will be interpreted as occupied.

Since the live map, e.g. Figure 5.2b, is just a visualization of the matrix, the new obstacle pops up by itself, the author only had to add an extra legend entry. For numerous examples see Chapter 6 and 6.2.

### 5.3.3 Inflating the map

The online map processing presented by Ueland (2016, Sec 4.2.2) includes a step where the map is inflated. The inflation is to ensure that the vessel had room to maneuver, with some safety distance. The cells closer than a defined inflation radius will be labeled as occupied. This process was repeated for the occupied cells occurring from the proximity sensors.

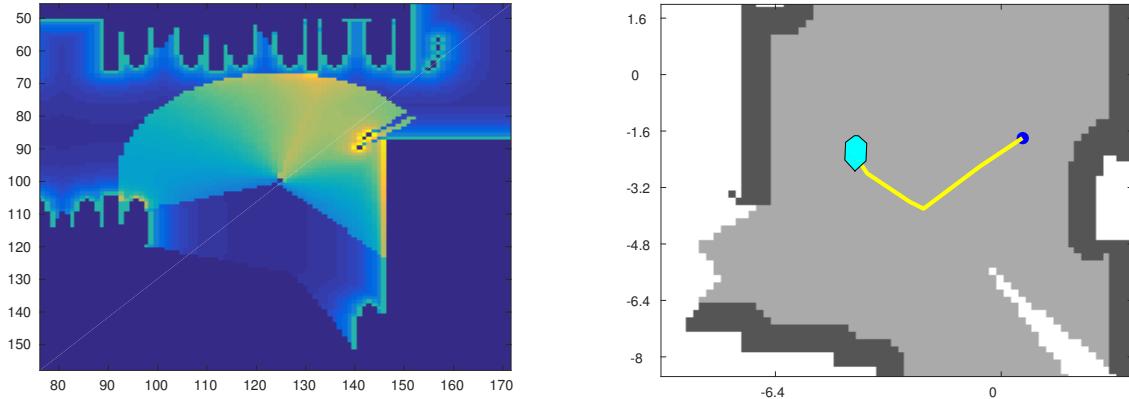
### 5.3.4 Feasibility correction of the cost-map

A weighting of the cells in the map was done by Ueland (2016, Sec 4.3.2.4), such that cells close to an obstacle were costly to travel through, this resulted in a behavior where the optimal path tried to stay away from obstacles. The weight  $w(s)$  on node  $s$  was:

$$w(s) = 1 + \frac{5}{0.1 + d_{obj}} \quad (5.6)$$

where  $d_{obj}$  is the Euclidean distance from the node to the closest occupied cell.

A natural consequence of introducing the steering law, earlier in this chapter, is to modify the cost-map such that the optimal path takes heading into consideration. When the vessel was round and heading was disregarded, the course angle could be changed with 180° without causing any problems. This 180° turn was typical in the exploration mode; when a closed area was mapped the optimal path would be to go back and discover a new area. Equation 5.6 was updated to also include a cost of nodes with a large angular difference relative to the heading of the vessel, where the nodes with a 180° difference were the most expensive. The method is shown in Appendix A.2 (it is too comprehensive to expressed as an equation), there a second cost-map



(a) Cost-map where the heading of the vessel is considered.

(b) Result of adding a heading cost on the path planner.

Figure 5.3: Heading dependent weighting introduced to the system

is created and added to the map described above. The result can be seen in Figure 5.3a, where the new cost-map is shown. In the Figure a sector in front of the vessel,  $60^\circ$  to either side, are not penalized, while the rest are. The nodes behind the vessel are the most expensive. This causes the optimal path to becoming an arc, see Figure 5.3b.

### 5.3.5 Path planner

The A\* search algorithm described in Ueland (2016, Sec 4.3.2) produced an optimal path for the vessel to follow, with respect to the cost-map discussed above. The algorithm, implemented as a MATLAB-function, searched for elements that are unexplored, 0, or occupied, 2, and adds them to a closed-matrix such that the path cannot be laid through these cells. Including the occupied cells from the proximity sensors, 3, ensured that only free cells were planned through.

Ueland (2016) searched entries equal to 2, to find accounted cells, to include the new type of occupied cells was merely a manner of changing to  $\geq 2$ . This change also enables even new types of occupied cells, defined as 4 or larger.

### 5.3.6 Map layers

In conclusion, the online map consists of three layers, where information is stacked upon each other. The first layer is map generated from SLAM along with the vessel position. The second layer consist of the obstacles detected by the proximity sensors, the sensors are not part of the

SLAM, they are added after, but before the path planner begins. The third layer is the path computed to reach the goal and avoid the obstacles.

### 5.3.7 Operator interaction

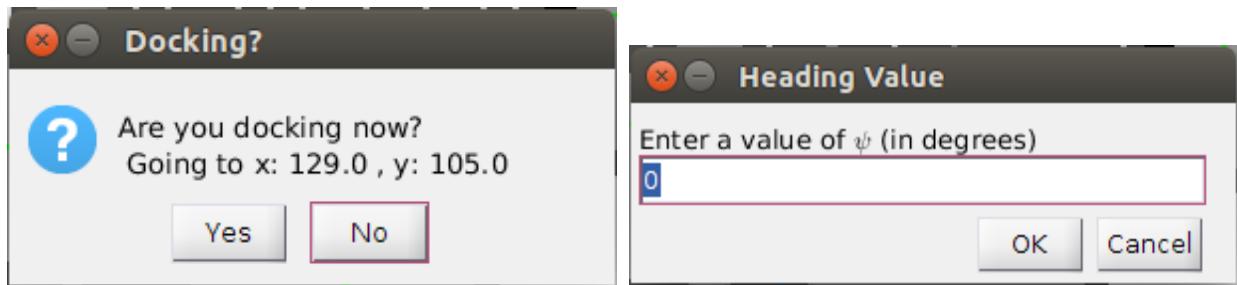
The operator interaction has been altered to confirm docking. By clicking on a pixel, in the window that monitor the exploration process, a box is pops up displaying the x- and y coordinates, along with a question if the vessel should dock there, see Figure 5.4a.

Selecting "Yes" yields a new box where the desired docking angle can be entered, see Figure 5.4b. Selecting 0 yields be the same as the initialization heading of the vessel, which is along the positive x-axis or towards right, with increasing  $\psi$  counterclockwise. The path planner would then do the exact same thing, but if the vessel came sufficiently close to the desired spot, the desired heading would change from the course angle, to the desired heading given in Figure 5.4b.

## 5.4 Sequence of docking and take-off

This Section aims at presenting a short and precise order of action for how to perform docking, and then take-off.

In order to perform docking at the desired spot (cell in the occupancy grid) it has to be explored by the lidar and defined as free, note that an undetected acoustic obstacle (not detectable by the lidar) could be there. If so the vessel would try to go there, but the acoustic sensors would detect the obstacle and soon after it would be inaccessible. The operator selects a cell, answers "Yes" to docking and inputs the desired heading. The path planner then designs an optimal, safe, path



(a) Displaying position of cell along with a question if the the vessel should try to dock here or not. (b) The desired heading of the vessel where it should dock needs to be entered.

Figure 5.4: Heading dependent weighting introduced to the system

towards the dock. When the vessel is close, the desired heading shifts from that of the steering law to the constant, user input, heading.

Take-off is when the vessel is leaving the dock. The operator selects any cell in the occupancy grid (does not have to be explored) and the path planner constructs a path towards that goal. The weighting of the cost map ensures no collision with the dock, since its defined as an obstacle.



# **Chapter 6**

## **Results**

This Chapter present results from the simulator and the experiments in the MCLab. In both cases, results on interfacing the proximity sensors are presented along with autonomous docking trials.

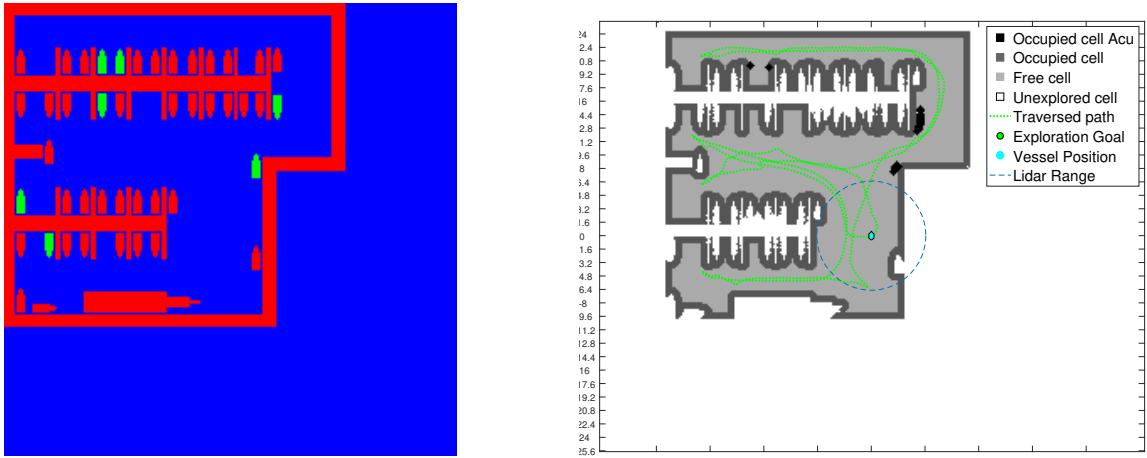
### **6.1 Simulations**

#### **6.1.1 Adding proximity sensors to the mapping environment**

The reference map (Figure 5.2c) was altered to including obstacles only seen by the proximity sensors, green objects in 6.1a. A proximity sensor emulator was added to the "mappingsimulator\_node" in a similar fashion as the lidar was added by Ueland (2016). It searched as four sensors placed normal to the vessels surface with no pitch, in the bow, port, stern and starboard on the vessel. The same assumptions were used such that there are no imperfections in the measurement. Similar to the physical trials, four sensors have been added to the vessel; they were placed normal to the vessels surface with no pitch, their locations were the bow, port, stern and starboard on the vessel.

In the system by Ueland (2016) the node in charge of creating the occupancy grid applied two mapping arrays during simulation. Both of them represented the occupancy grid; however, one was the pre-generated reference-map, i.e. the environment that the vessel was deployed in. The other was the explored-map array, which was dynamically explored, that represents how much of the reference-map that the vessel has explored.

To include proximity sensors into the system with minimal impact on the existing system a third mapping array was introduced, it contained the map-matrix generated by running Figure 6.1a



(a) Map to explore, green is only detectable by proximity sensors.

(b) Result of completed scan in autonomous mode, the vessel have returned to its initial position.

Figure 6.1: The reference map, drawn in Paint, prior to exporting to MATLAB, and the result after an autonomous exploration. Notice how the proximity sensors detect only 4 out of 7 vessels. This is because the Lidars range is larger than the proximity sensor.

through the script in Appendix A.1. The acoustic obstacles (green) were removed and defined as free in the reference-map for the lidar.

As can be seen by comparing Figure 6.1a and 6.1b the green areas are defined as free cells by the lidar and SLAM; however, they become black if the vessel was close and the proximity sensors detected them. Some objects are not detected simply because the vessel was not close enough. A comparison results in the fact that 4 out of 7 vessels where detected, because the lidar has a larger range than the proximity sensors. This is not a problem, nor a fault in the system, since the proximity sensors should prevent collision. Looking at the two rightmost, green, vessels they where endangering the exploration; however the proximity sensors detected them, and the path was replanned.

### 6.1.2 Autonomous docking

After the environment had been mapped (Figure 6.1b) a desired location was selected, docking was chosen and the docking angle was set to  $90^\circ$ , i.e. pointing up in the map. The navigation in the map can be seen in Figure 6.2a-6.2c. The convergence of  $\eta$  to  $\eta_d$  can be seen in Figure 6.2d and 6.2e. Notice how  $\psi_{set}$  jumps to  $90^\circ$  according to the selected docking angle from Figure 5.4b.

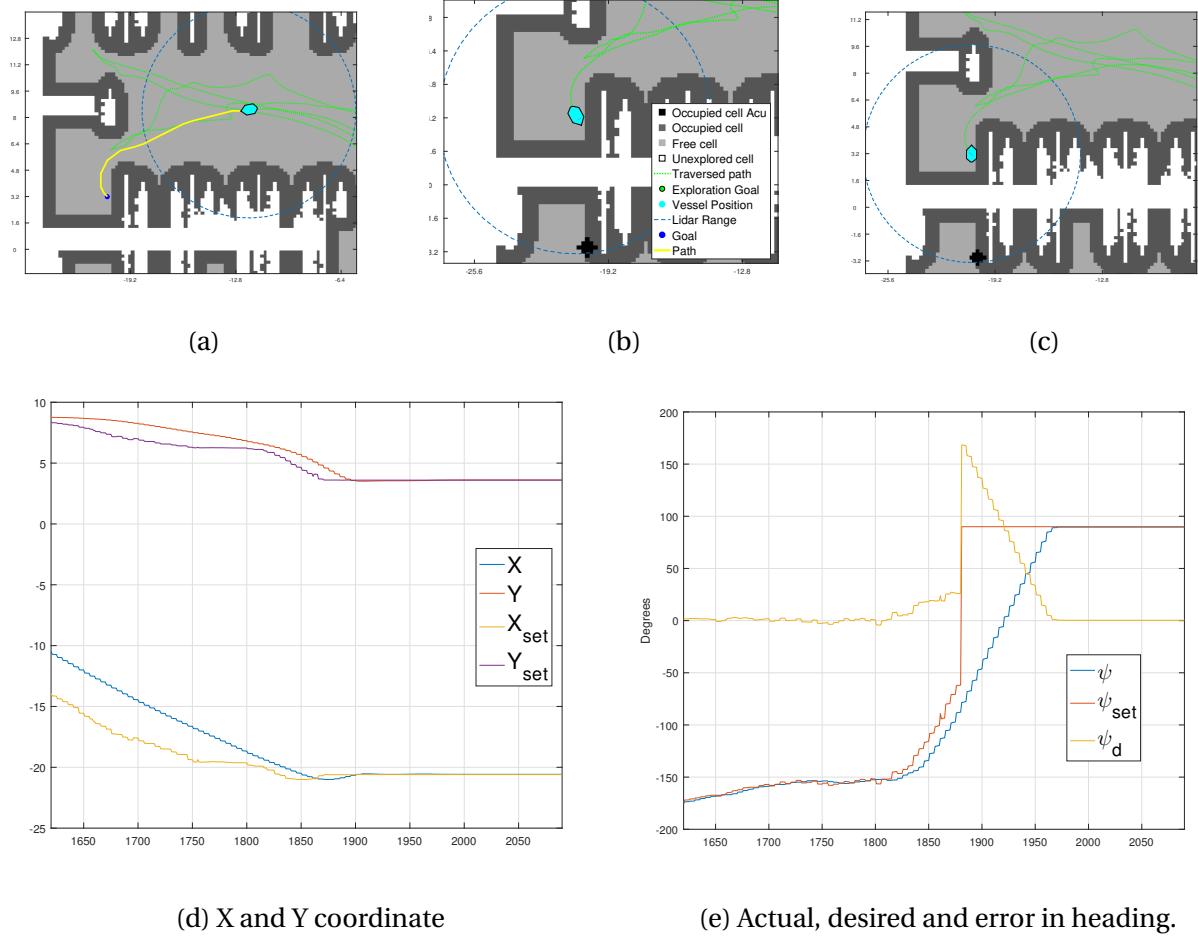


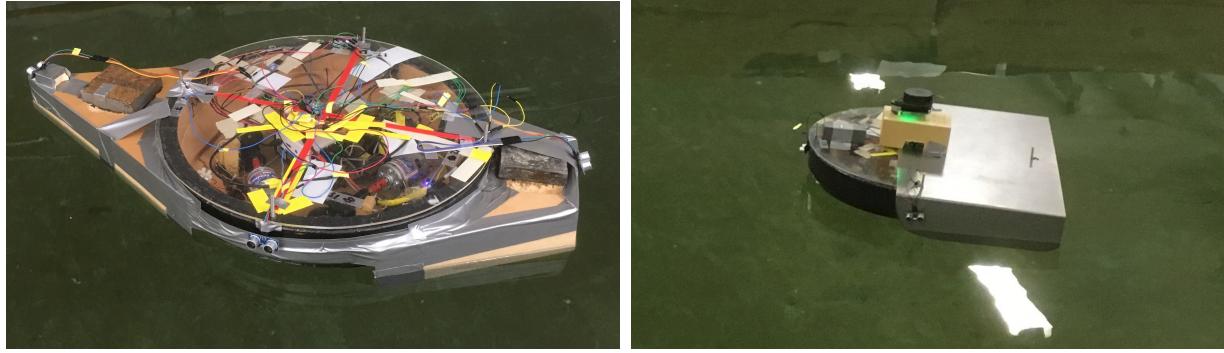
Figure 6.2: Docking of the CS Saucer, desired docking angle was set to  $90^\circ$ .  
Notice the jump in  $\psi_{set}$ , this is when the vessel is close enough and docking is initiated.

## 6.2 Experiments

In the following Section the experimental results from the MCLab will be presented. In the first experiment only the heading controller was implemented and the two modules where compared. In the second the proximity sensors where tested, obstacles below the scan plane of the lidar was used. The last experiment tested the autonomous docking. Note that video and pictures of the CS Saucer in the MCLab is appended to the electronic attachment.

### 6.2.1 Testing of modules

In the description of the experimental setup, Section 2.1.4.9, two modules for increasing the yaw resistance was introduced. The first was two pieces of Divinycell H60 which where fitted of the bow and stern of the CS Saucer, see Figure 6.3a.



(a) First Module

(b) Second module.

Figure 6.3: Both of the modules where tested in the basin, the first proved to be the best. Notice that the proximity are installed.

The first module turned out to be cumbersome to attach to the vessel, a substantial amount of duct tape was needed. When placed on water the vessel exhibited the desired behaviour where both sway and yaw resistance where increased, i.e. the vessels preferred direction became surge. This module increased the mass of the system with 5kg increasing the inertia, it also increased the waterplane area.

The second module is easily mounted on the vessel, a 1kg weight was needed in the bow to counteract the trim. The module increased the sway resistance, but did not contribute significantly to increase the yaw resistance. The total additional mass became 3.5kg, but in contrast to the first module the waterplane area did not change significantly, the thickness of the plate was 1mm.

Both modules where tested in a exploration mode toward a userdefined exploration goal. Through visual inspection the first module was deemed to increase the controllability of the vessel the most, hence only the Divinycell module was used in the last two experiments.

### 6.2.2 Proximity sensors

To test the acoustic sensors an environment in the basin was cut off by a wall (Figure 2.6), such that roughly 1/3 of the basin was used. Four obstacles where used, where two are below the lidars scan plane and hence only detectable by the proximity sensors.

The result of a scan can be seen in Figure 6.5. Two distinct groups of black cells can be seen around [0, 0] and [6.4, 0], which are the objects only seen by the proximity sensors. Other cells are darker gray, they represent are occupied cells defined through the lidar scan. In the figure some black cells are seen next to gray cells, the proximity sensor rounds the distance down to find a valid cell, hence it may identify an object as one cell closer than what the lidar did. Also seen in

the figure are two pair of black cells, surrounded by water, they represent faulty measurements, they could lead to errors.

Two other dark dark spots are in the middle of the water, they are false readings due to an unknown error in the measurement from the proximity sensor.

The shattered dark gray are around the basin are obstacles detected by the lidar, they are the walls of the MCLab (and not the basin). A laser beam might hit outside the basin walls if the vessel is vibrating a little. This was known and was discussed in Ueland (2016, Sec. 6.3)

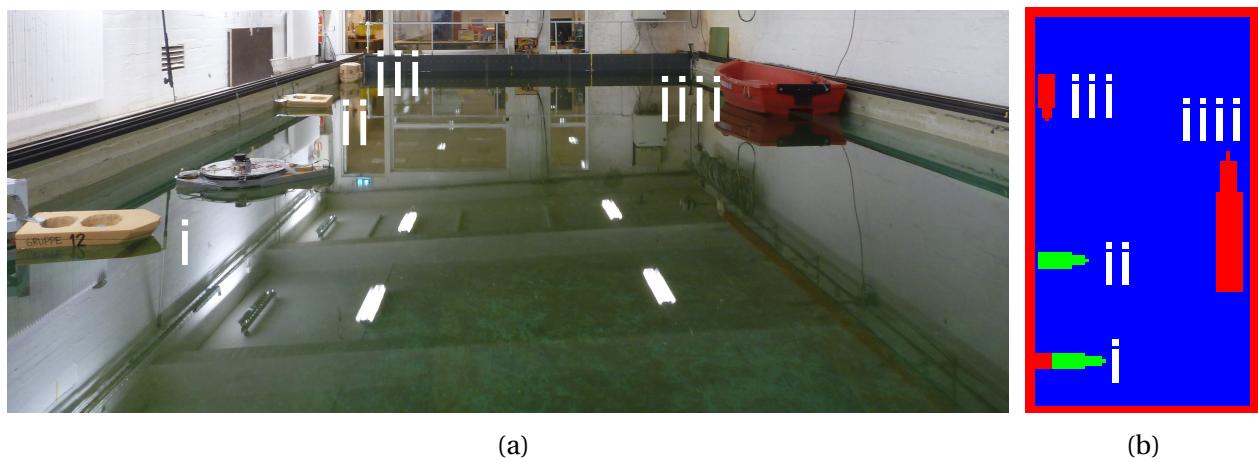


Figure 6.4: Test environment in the basin. i-ii are only detectable by the acoustic sensors, while the lidar is able to detect iii and iv. The constructed below the picture to section off the basin. The right picture illustrates the setup, schematically, drawn in Paint.

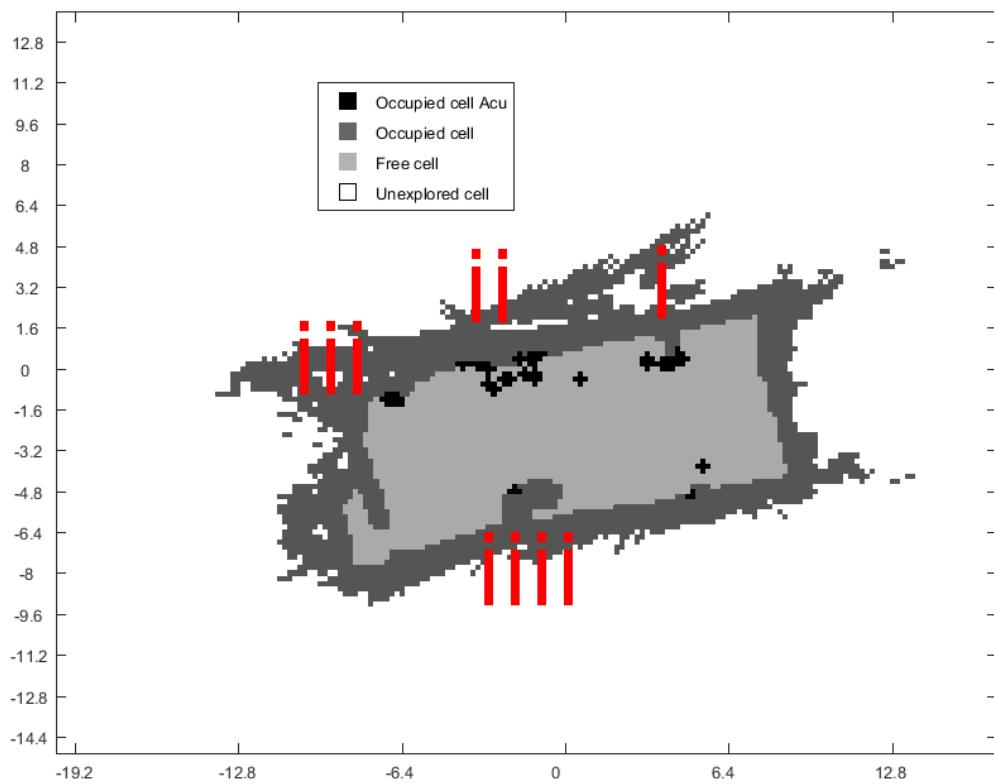


Figure 6.5: Scan of the basin showing obstacles detected by the lidar and the acoustic sensors. Numbers relate to Figure 6.4

### 6.2.3 Autonomous docking

The last experiment done in the basin was to test the autonomous docking. The vessel was initialized parallel to the basin wall, and started by mapping the area of operation. Since the used section was only 1/3 of the basin, the exploring was done quickly. A spot in the right side of the basin was selected and the docking angle was set to 0, i.e. the same as the initialization heading, which was along the wall. The x and y coordinate is plotted in Figure 6.6a below, while the heading can be seen next to it. Note than in Figure 6.6b the angles goes outside the regime  $-180^\circ - 180^\circ$ , this is only for readability.

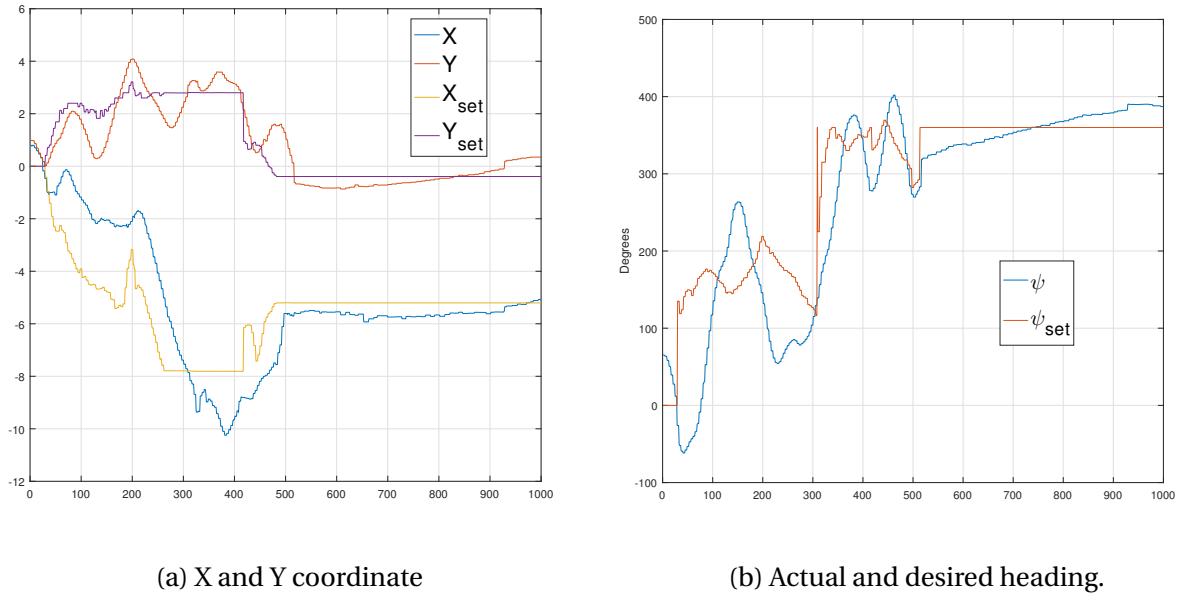


Figure 6.6: Vessel behaviour before docking. See the jump in  $\psi_{rel}$  around sample 500.

### 6.3 Discussion

In this Section the results from the previous two sections will be discussed. The focus of discussion will be on how well the simulator represents the real life scenario and how the CS Saucer behaved.

In Table 3.1 the parameters used in the maneuvering equation was presented, where the variables related to sway and yaw had been altered. They were changed such that the vessel would exhibit a behaviour more similar to a conventional boat. The vessel with the new parameters along with the heading controller behaves satisfactorily in the simulations; however, the simulator cannot be expected to replicate the physical system in every detail.

The Divinycell module proved to improve the stability and performance of the vessel one it was tested in the MCLab, yet improvements can still easily be seen. By the author's opinion the source to the lack of stability are small imperfections in the thruster mapping for low RPM. The lowest produced thrust is too high, and the vessel is not capable of coming to a complete stop, it will always glide in the water before jerking back. This again causes problems during exploration and docking. In the first case the vessel could crash with the obstacles unseen by the lidar, if they are detected late by the proximity sensors, in the latter case it meant that the vessel either bumped the wall or came to a stop a before the wall, and then had a drive off.

The implementation of the acoustic sensors was a success. It was actually implemented and tested in the MCLab prior to its implementation in the simulator. This had the benefit that it was easy to create the proximity sensor emulator in the simulator. It was only to replace the the

ROS-subscription block from the node used in the lab. A problem was that the proximity sensors produce a lot of noise, so a lower limit of  $15\text{cm}$  had to be set since anything closer than that was not registered. The sensors have a timeout after  $12\text{ms}$  which is equivalent to a distance of  $4\text{m}$ , hence objects further away than  $2\text{m}$  will not be detected. Nevertheless, the sensor frequently responded with a nonzero output when it should not have; however, these numbers would often stay around  $10\text{cm}$ , they were excluded since a lower truncation limit of  $15\text{ cm}$  was utilized. Further work needs to be done on signal processing of the acoustic sensors. The emulator is perfect with no noise; however the investigation logic is the same such that distances less than  $15\text{cm}$  will be disregarded in the simulator too.

The autonomous docking, and take-off, was successful in both the simulator and the lab experiment, it is believed that the logic behind switching to a constant docking angle when the vessel is "sufficiently" close is a valid scheme. Since the vessel with its constrained thrusters is fully actuated, it glides and rotates into the spot, perfectly, in the simulator. Although, admittedly the vessel was not able to dock perfectly in all experimental trials (it bumped into a wall, or stopped too far way), the trials that went without any incidents can be viewed as a proof of concept.

# **Chapter 7**

## **Concluding remarks**

### **7.1 Conclusion**

The previously implemented ROS-based platform made it possible to augment the CS Saucers hardware and software in an efficient manner. The acoustic sensors were designed to be used with an Arduino. Through inbuilt Arduino-functions the measured distance could be published as a ROS topic (Figure F.1).

This thesis has presented work on Thrust Allocation inspired by Frederich (2016) and how it could be implemented on the CS Saucer. It presents works on interfacing acoustic sensors with the developed system by Ueland (2016) to detect obstacles outside the plane of the laser scan from the lidar. A steering law has been implemented with the ultimate goal of docking the vessel at a specific location with a corresponding heading. To achieve this two different, physical, modules have been creating which can be attached to the CS Saucer.

The control allocation presented focused on four configurations, were a fixed bow thruster and two rotatable stern thrusters were combined. The two first modes assumed independent controllable stern thrusters, which few twin-engine leisure boats have since the engines are mounted on a common shaft. Regardless of how many stern thrusters a vessel may have if they are mounted on a common shaft, they can be combined into one thruster in the control allocation. The two least controllable modes, configuration 3 and 4, are the most realistic with respect to the fleet of leisure boats. The four modes were implemented into the control scheme of the CS Saucer; however it proved to unreliable. Thoughts on how to solve this are presented in chapter 7.2.

Four HC-SR04 acoustic sensors were successfully installed and integrated with the system. If they detected an obstacle in a cell, which was defined as free, it would be redefined as an occupied cell. For the sake of clarity they appear in the online map as black cells, while obstacles

detected by the lidar are dark gray. For the system it is treated as an occupied cell in the same manner as if it was the lidar that detected it, i.e. the cells get inflated, avoided by the path planner, etc.

Autonomous docking was developed by the use of the simulator; the physical parameters was altered and then a steering law was designed along with altering the cost-map to favor nodes in front of the bow. In the simulator, the docking was executed smoothly since the vessel is fully actuated. In the lab, a proof of concept has been proven.

The reader should be aware that the results from the controlled laboratory experiments cannot be expected to function well in a non-ideal environment. Roll and pitch will affect the lidar scan, the sensors need to be environmentally protected without reducing their ability and object at a skew angle might not be detected by the lidar, nor the acoustic sensors.

Summarized, new technology has been presented, if it was to be assigned to a Technology Readiness Level (TRL), according to the definition from the European Commission, it would be level 3, see Table 7.1. By the author's opinion further work has to be done to claim TRL4.

Table 7.1: TRL in the European Commission (*European Commision, 2014*)

<b>TRL</b>	<b>Description</b>
TRL 1.	basic principles observed
TRL 2.	technology concept formulated
TRL 3.	experimental proof of concept
TRL 4.	technology validated in lab
TRL 5.	technology validated in relevant environment (industrially relevant environment in the case of key enabling technologies)
TRL 6.	technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies)
TRL 7.	system prototype demonstration in operational environment
TRL 8.	system complete and qualified
TRL 9.	actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies; or in space)

## 7.2 Further work

This section will present all the aspects of this thesis that need to be improved. It will also give account, by the authors' opinion, on how the vessel could enhance its TRL for deployment in a more harsh and dynamic environment.

The primary objective of this thesis is to perform autonomous docking and integrating proximity sensors into the sensor suite for better performance; however, the real life application was always aimed for leisure boats, which are underacted. Hence an effort was made to study TA for constrained rotatable thrusters. Further work on implementing the system needs to be done. In order to improve the optimal control, the author recommends looking into how the thruster angles are defined, according to the body frame. In Frederich (2016, Figure 3.2.2), the thruster angles have an offset of  $180^\circ$  with respect to heading, i.e. a thruster angle of 0 points backward. Other work could be to make a feasibility study of the maneuvering of the vessel in the four modes and create a more accurate cost map with inaccessible cells in the infeasible region.

The acoustic sensor had the problem of sending faulty messages detecting obstacles very close. So all readings below 15cm had to be neglected an observer is recommended to handle this issue.

If future work would like to use physical modules with the CS Saucer, the author recommends creating a 3D model, in a drafting software, for print. The recommended module in this thesis was hand drawn and made by on a band saw. It did not fit perfectly around the curvature of the CS Saucer; it also had a constant area such that a gap arise on the underside, see Figure 7.1a. After making a suitable model, the vessel should be remodeled.

The author bought a kit for mounting the acoustic sensor on a servo, see Figure 7.1b; however, this was never tried. The idea was that the proximity sensor could look around, either in the horizontal plane, or the vertical. A logic could be implemented such that the acoustic sensor tracked an object, once found, to continuously measure the distance. The author has already made software implementations where each proximity sensor is assigned a yaw and a pitch angle, together with a position. Controlling the servo from the Arduino, developing a logic and testing needs to be done.

The thesis title reads "Autonomous docking for marine vessels using a lidar and proximity sensors". By the author's opinion further work could be done on improving the autonomy regarding the docking. Instead of depending on the user inputting the correct docking angle the user should, for instance, select among four choices which read "Bow", "Stern", "Port" and "Starboard", depending on where the quay should be.

As written by Ueland (2016), it is a future goal that the system should handle dynamic object. The SLAM algorithm assumes fixed obstacles when creating the map. The author recommends

developing a second, independent, positioning system in cascade with an Inertial Measurement Unit (IMU), thus measuring its own movements, the system would be able to deduce how much the obstacle is moving. Further the dynamic objects should be removed from the positioning part of SLAM and analyses by a separate ROS node would be in charge of collision avoidance.

If an unknown current would be introduced the author recommends updating the steering law to an integral-LOS, according to Caharija et al. (2014).

Another feature needed, to increase the TRL level, is to handle pitch and roll. One solution could be to mount the lidar on a gimbal to keep it level at all times. Another solution could be to use a 3D-lidar, such as the HDL-32E presented in Figure 1.9b. It can use SLAM and produce a 3D map of its environment even; however, its price of almost \$30 000, according to *Hizook* (2016), may prove too expensive.



(a) Gap between hull and module.



(b) Acoustic sensor mounted in a Servo.

Figure 7.1: Suggestions for future work, the module needs to fit the vessel better and the acoustic sensors is suggested to be mounted on a servo.

# Bibliography

*adressa* (2016). [Accessed 18 Dec. 2016].

**URL:** <http://www.adressa.no/nyheter/trondheim/article6640757.ece>

*AMOS* (2016). [Accessed 15 Dec. 2016].

**URL:** <http://www.ntnu.edu/amos>

*Arduino Webpage* (2016). [Accessed 30 Nov. 2016].

**URL:** <https://www.arduino.cc/en/Main/arduinoBoardMega>

Berretta, D., Urbano, N., Formentin, S., Boniolo, I., Filippi, P. D. and Savaresi, S. M. (2013), Modeling, identification and control of a boat parking assistance system, in ‘Control Conference (ECC), 2013 European’, pp. 3012–3017.

Breivik, M. (2010), Topics in guided motion control of marine vehicles, PhD thesis, Norwegian University of Science and Technology.

Caharija, W., Pettersen, K. Y., Sørensen, A. J., Candeloro, M. and Gravdahl, J. T. (2014), ‘Relative velocity control and integral line of sight for path following of autonomous surface vessels: Merging intuition with theory’, *Proceedings of the Institution of Mechanical Engineers Part M: Journal of Engineering for the Maritime Environment* **228**(2), 180–191.

**URL:** [www.scopus.com](http://www.scopus.com)

*EasyMile* (2016). [Accessed 29 Nov. 2016].

**URL:** <http://easymile.com/>

*European Commision* (2014). [Accessed 13 Dec 2016].

**URL:** [http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014\\_2015/annexes/h2020-wp1415-annex-g-trl\\_en.pdf](http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf)

Ferguson, M. (2016), ‘Rplidar source code.’. [Accessed 17 Dec. 2016].

**URL:** <http://wiki.ros.org/rplidar>

Fossen, T. I. (2011), *Handbook of marine craft hydrodynamics and motion control*, John Wiley & Sons, Ltd.

Frederich, P. (2016), Constrained Optimal Thrust Allocation for C/S Inocean Cat I Drillship, Master thesis, Norwegian University of Science and Technology.

*Hizook* (2016). [Accessed 14 Dec. 2016].

**URL:** <http://www.hizook.com/blog/2010/08/24/velodyne-hdl-32e-new-high-end-laser-rangefinder>

Idland, T. K. (2015), Marine Cybernetics Vessel CS Saucer: Design, construction and control, Master thesis, Norwegian University of Science and Technology.

*IME Faculty* (2016). [Accessed 15 Dec. 2016].

**URL:** <http://www.ntnu.edu/ime>

*IMT, NTNU* (2016). [Accessed 30 Nov. 2016].

**URL:** <https://www.ntnu.edu/imt/lab/cybernetics>

*IVT Faculty* (2016). [Accessed 15 Dec. 2016].

**URL:** <http://www.ntnu.edu/ivt>

*Janez Cimerman* (2015). [Accessed 29 Nov. 2016].

**URL:** <http://meetjanez.splet.arnes.si/2015/08/22/neato-xv-11-to-ros-slam/>

Johansen, T. A. and Fossen, T. I. (2013), ‘Control allocation—a survey’, *Automatica* **49**(5), 1087 – 1103.

**URL:** <http://www.sciencedirect.com/science/article/pii/S0005109813000368>

Johansen, T. A., Fossen, T. I. and Berge, S. P. (2004), ‘Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming’, *IEEE Transactions on Control Systems Technology* **12**(1), 211–216.

Kjerstad, O. K. (2010), Weather-optimal positioning control for underactuated USVs, Master thesis, Norwegian University of Technology and Science.

*Kohlbrecher S, M. J.* (2014). [Accessed 29 Nov. 2016].

**URL:** [http://wiki.ros.org/hector\\_slam](http://wiki.ros.org/hector_slam)

Ludvigsen, M. and Sørensen, A. J. (2016), ‘Towards integrated autonomous underwater operations for ocean mapping and monitoring’, *Annual Reviews in Control* **42**, 145 – 157.

**URL:** <http://www.sciencedirect.com/science/article/pii/S1367578816300256>

*Maritime Robotics* (2016). [Accessed 28 Nov. 2016].

**URL:** <http://www.maritimerobotics.com/>

National Research Council (2005), *Autonomous Vehicles in Support of Naval Operations*, The National Academies Press, Washington, DC.

**URL:** <https://www.nap.edu/catalog/11379/autonomous-vehicles-in-support-of-naval-operations>

*Neato XV Series* (2016). [Accessed 05 Nov. 2016].

**URL:** <https://www.neatorobotics.com/robot-vacuum/xv/>

*Robotshop* (2016). [Accessed 30 Nov. 2016].

**URL:** <http://www.robotshop.com/en/rplidar-360-laser-scanner.html>

*ROS-community* (2016). [Accessed 17 Dec. 2016].

**URL:** <http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF>

*RV Gunnerus* (2016). [Accessed 15 Dec. 2016].

**URL:** <https://www.ntnu.edu/oceans/gunnerus>

Spange, J. (2016), A Study in Concept Development of Dynamic Positioning Functionality for Leisure Boats, Project thesis, Norwegian University of Science and Technology.

Sørdalen, O. (1997), ‘Optimal thrust allocation for marine vessels’, *Control Engineering Practice* **5**(9), 1223 – 1231.

**URL:** <http://www.sciencedirect.com/science/article/pii/S0967066197843614>

Sørensen, A. J. (2005), ‘Structural issues in the design and operation of marine control systems’, *Annual Reviews in Control* **29**(1), 125 – 149.

**URL:** <http://www.sciencedirect.com/science/article/pii/S136757880500012X>

*Teknisk ukeblad* (2016a). [Accessed 05 Nov. 2016].

**URL:** <http://www.tu.no/artikler/verdens-forste-førerløse-passasjerferge-kan-gå-over-en-kanal-i-trondheim/363790>

*Teknisk ukeblad* (2016b). [Accessed 05 Nov. 2016].

**URL:** <http://www.tu.no/artikler/norsk-selskap-bak-verdens-forste-autonome-skip-til-kommersiell-drift/363811>

*Tesla Motors* (2016). [Accessed 28 Nov. 2016].

**URL:** [https://www.tesla.com/no\\_NO/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware](https://www.tesla.com/no_NO/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware)

Ueland, E. (2015), Preparing the thruster and control systems on the CS Saucer for autonomous tasks., Project thesis, Norwegian University of Science and Technology.

Ueland, E. (2016), Marine Autonomous Exploration using a Lidar, Master thesis, Norwegian University of Science and Technology.

*Valhalla, G.* (2010). [Accessed 17 Dec. 2016].

**URL:** <http://se.mathworks.com/matlabcentral/fileexchange/29107-real-time-pacer-for-simulink>

*Velodyne LiDAR* (2016a). [Accessed 29 Nov. 2016].

**URL:** <http://velodynelidar.com/products.html>

*Velodyne LiDAR* (2016b). [Accessed 16 Sep. 2016].

**URL:** <http://velodynelidar.com/hdl-32e.html>

# Appendix A

## MATLAB scripts

### A.1 Create map for simulation

```
1 imdata = imread('Picture_to_load.png'); %Must be size 256*256
2 [a,b,c] = size(imdata);
3 if (a ~= 256 || b~= 256)
4     error('Picture size in %ix%i, must be 256x256',a,b)
5 end
6 MAP = 66*ones(256,256); %Initiate
7 for i = 1:256
8     for j = 1:256
9         if (imdata(i,j,1) > 250) %Red
10             MAP(i,j) = 2; %Occupied, visible for all.
11         elseif (imdata(i,j,2) > 250) %Green
12             MAP(i,j) = 3; %Occupied, outside the Lidar 2D...
13             pane.
14         elseif (imdata(i,j,3) > 250) %Blue
15             MAP(i,j) = 1; %Free
16         else
17             error('Elementary color not found at (%i, %i) RGB = [%i,%i,%i]\n',...
18                 ,i,j,imdata(i,j,1),imdata(i,j,2),imdata(i,j,3))
19         end
20     end
21 save MAP.mat MAP
```

## A.2 Angular dependent cost-map

```
1 Anlge2Cell=zeros(Xlength,Ylength); %Angle to each cell
2 for Degangle=-180:1:180
3     Radangle=Degangle*pi/180;
4     for r=0:.1:6.5/resolution;
5         XPOS=PosX+round(r*cos(Radangle-Psi));
6         YPOS=PosY+round(r*sin(Radangle-Psi));
7         if (XPOS == PosX && YPOS == PosY)
8             1;
9         elseif map(YPOS,XPOS) > 1
10            break
11        elseif (Anlge2Cell(YPOS,XPOS) == 0 && XPOS<256 && XPOS > 0 && YPOS...
12            <256 && XPOS > 0)
13            if 2*abs(Radangle) < 3
14                Anlge2Cell(YPOS,XPOS)=0;
15            else
16                Anlge2Cell(YPOS,XPOS)=2*abs(Radangle);
17            end
18        end
19    end
20 end
```

# **Appendix B**

## **Electronic attachments**

Files listed in this appendix are included in the electronic attachment. They will also be available through [https://github.com/NTNU-MCS/CS\\_Saucer\\_ROS](https://github.com/NTNU-MCS/CS_Saucer_ROS). Most of the files are from Ueland (2016); however, alterations by the author is clearly commented in the code.

### **B.1 Parameter generations files**

#### **VesselParameterSet.m**

MATLAB script for setting parameters for both the "motioncontroller.slx" Simulink model and the "Vessel\_simulator.slx" Simulink model.

#### **ParameterSet256.m**

MATLAB script for setting parameters prior to running simulations.

#### **Map256.mat**

Files containing stored MATLAB workspaces. For simple setting the dimensions of the Simulink nodes.

#### **Quay256.mat/Quay\_acu256.mat**

Matrices of the maps shown in Figure 5.2c and 6.1a, respectively, after conversion through the script in Appendix A.1.

#### **Posedata.mat**

Files containing stored MATLAB workspaces. For initializing of simulations.

## B.2 Path\_exploration node

### B.2.1 Exploration\_Pathplanner node

#### Exploration\_pathplanner.slx

Responsible for running the exploration node. Subscribes to position and map, and publish a path to the ROS architecture. To be run on operator computer. Depends on the following sub-scripts

- **ExplorationMain.m**

Main script for exploration and pathplanning. Responsible for generating path according to implemented guidance and exploration strategy

- **addAcousticSignals.m**

Responsible for adding the acoustic obstacles to the occupancy grid

- **InflateMap.m**

Responsible for inflating (expanding) objects according to the preset inflation radius

- **Poppout.m**

Responsible for reducing the map to reachable cells

- **LidarUpdate.m**

Assumes that the vessel can see ahead in section where the lidar has not received a signal.

Assumes that there is no objects in these directions

- **PathAstar.m**

Generates a path in the occupancy grid to goal nodes according to the implemented A\* search algorithm

- **AtarFindCost.m**

Find the cost of travelling between two connected nodes. Only used when all nodes in a connection are weighted

- **LidarFindGap.m**

Identifies Gaps in the map. Only used if the Gap Based Exploration strategy is applied

- **InvestigateGap.m**

Examine whether a Gap can be checked as explored or not. Only used if the Gap Based Exploration strategy is applied

## B.2.2 Path2SetPoint node

### **Path2SetPoint node folder**

To be run as a regular ROS node.

#### **Path2SetPoint.slx**

Simulink model used to generate the node in C++. Generates appropriate setpoint on the planned path.

## B.2.3 Scan2SetPointDist

### **Scan2SetPointDist node folder**

To be run in as regular ROS node.

#### **Scan2SetPointDist.slx**

Simulink model used to generate the node in C++

Find the closest object based on lidar scan. Contains logic for finding setpoint distance.

## B.2.4 Hector2VesselPos

### **Hector2VesselPos node folder**

To be run in as regular ROS node.

#### **Hector2VesselPos.slx**

Simulink model used to generate the node in C++

Transforming position vector from lidar coordinate system to that of the vessel.

Includes a quaternion transformation

## B.2.5 Motion\_Controller

### **Motion\_Controller node folder**

to be run as regular ROS node.

#### **Motion\_Controller.slx**

The control system of the vessel, responsible for controlling the vessel to the desired setpoint. This is perform this by publishing appropriate signals to the actuators. It is advised to use the Simulink motion controller rather than the C++ compiled version of this node.

#### **Motion\_Controllercompile.slx**

Simulink model used to generate the node in C++

## B.2.6 Arduino code

### **CS SaucerThrustRPMVoltage.inu**

Arduino code responsible for outputting the PWM signals to actuators, publishing the Acoustic signals (in *cm*) and for monitoring the voltage of battery. Code is written in C++ and utilizes Arduino/ROS libraries.

### **CS SaucerSimple.inu**

Simpler version that only publish to signals to actuators. Does not monitor RPM signals nor, voltage of battery.

## B.2.7 Hector-Slam nodes

Open source nodes that are used for SLAM

See [http://wiki.ros.org/hector\\_slam](http://wiki.ros.org/hector_slam)

## B.2.8 RPLidar node

Open source nodes that are used as driver for the RP-lidar

See <http://wiki.ros.org/rplidar>

## B.2.9 ROS serial node

Open source packadge got the Arduino

See <http://wiki.ros.org/rosserial>

## B.3 Simulator nodes

### B.3.1 Vessel simulator node

#### Vessel Simulator node folder

To be run in as regular ROS node.

#### VesselSimulator.slx

Simulink model used to generate the node  
in C++. Node that simulate the vessel dynamics

### B.3.2 Mapping fimulator node

#### Mapping Simulator node folder

To be run in as regular ROS node

#### MappingSimulator.slx

Simulink model used to generate the node in C++.  
Node that simulates map generation. Sections where lidar rays are not reflected is not updated  
in this version. The reference map and grid size needs to be defined in Simulink before code  
generation.

## B.4 Launch files

#### Res01.launch

Launch RPLidar and Hector Slam for mapping with resolution 0.1, and a gridsize (256x256)

#### Res02.launch

Launch RPLidar and Hector Slam for mapping with resolution 0.2, and a gridsize (256x256)

#### Simulator.launch

Launches simulator nodes. Launched with the "roslaunch" command.

#### LaunchNoLidar.launch

Launches nodes for deployment of vessel, excluding the RPLidar and Hector-SLAM nodes

## B.5 Other

### B.5.1 Real-Time Pacer

Open source Simulink block for slowing simulations down to real-time. (*Valhalla, G.*, 2010)

### B.5.2 Marine Systems Simulator

The Marine Systems Simulator (MSS) is a MATLAB/Simulink library and simulator for marine systems. It includes models for ships, underwater vehicles, and floating structures. The library also contains guidance, navigation, and control (GNC) blocks for real-time simulation. (*Valhalla, G.*, 2010)

### B.5.3 Photo and video documentation

Pictures and some video from the experiments are presented.

### B.5.4 Thesis files

This thesis along with all figures etc.

# **Appendix C**

## **Software set up and installation**

This manual is also presented in Ueland (2016), minor changes are done, to help the reader. It is intended for use at NTNU, and especially for students that wants to use ROS as their software framework for projects in the Marine Cybernetics laboratory at NTNU. Though specifically intended for students at NTNU though it is hoped that it might also be useful for other readers.

The goal of the manual is to provide the necessary steps such that users with as little effort as possible can set up their ROS-framework for use in the MC Lab. This is particularly relevant for future master's students at NTNU, which by utilizing this manual will get more time to focus on their own thesis.

The manual is split into two parts. The first part explains how to set up the system architecture with ROS on an with Raspberry Pi 2, computer and an Arduino. This part is not specific for the CS-Saucer platform and is intended for persons wishing set up similar ROS-frameworks on their own systems.

The second part goes into detail on the interfacing of ROS for this particular project and explain step by step how the codes have been generated and how to apply, edit and reuse the generated software.

The manual is written for inexperienced users, meaning that the process is explained step by step. The manual will not go in depth on how to use ROS beyond what's needed in order to achieve the desired setup, and a more thorough investigation is needed in order for the reader to obtain the full understanding of how to utilize ROS.

The author of this manual has used countless hours to implement the software system. Issues that in retrospect has simple solutions have often taken a lot of time to solve. It is hoped that the next user do not use the same amount of time in the same issues.

Note that due to the high level of development of ROS and robotics the manual will probably need to be adapted to future versions. For example, Raspberry Pi 3 had just arrived in March 2016, while ROS 2.0 is under development.

Please note the following:

- In the manual, the dollar sign \$ indicate a line of text that should be written in the Ubuntu-terminal window.
- In the manual Gedit is used as the text editor. This can be replaced with the readers favourite text editor.
- The manual is written and tested for ROS-Indigo.

## C.1 Installing ROS and UBUNTU

For this section you will need the following:

- Single Board Computer (Tested in this manual: RaspberryPi-2)
- Laptop/computer for installment of Ubuntu
- Micro-SD card (recommended storage of 16 GB) and means of connecting it to computer (Micro-SD/SD adapter or Micro-SD/usb adapter)
- Hardware for interfacing with the RP2 (monitor, ethernet-cable or WiFi adapter, HDMI-cable, mouse and keyboard).

### C.1.1 Ubuntu and ROS on your personal computer

Use your favorite method to install UBUNTU 14.04-lts on your personal computer. This might be installed through via Oracle Virtual Box, or as its own partition.

Now install ROS indigo. [<http://wiki.ros.org/indigo/Installation/Ubuntu>]

### C.1.2 Ubuntu and ROS on your single board computer (RP2)

Follow the instructions given in the link , which in detail explain how to install Ubuntu 14.04 on Raspberry Pi 2. [<https://wiki.ubuntu.com/ARM/RaspberryPi>]

The key steps for performing this operation will be summarized below:

- Download the Ubuntu 14.04 Trusty image on your personal computer.
- Install image on microSD-card through this manual (Windows):  
[[www.raspberrypi.org/documentation/installation/installing-images/windows.md](http://www.raspberrypi.org/documentation/installation/installing-images/windows.md)]
- Insert micro-SD card to RP2 and connect the RP2 to internet, monitor, mouse and keyboard.
- Now install Ubuntu according to manual given in the link above. For reference this video is good: [<https://www.youtube.com/watch?v=UGSQ7nzVCs4>]

You now want to install ROS-Indigo on RP2. Use the on the following instructions: [<http://wiki.ros.org/indigo/>]

## C.2 Getting started with ROS

The following commands generate a personal workspace on ROS. (Do this both on RP2 and the laptop)

Create the workspace:

```
1 $mkdir -p ~/catkin_ws/src  
2 $cd ~/catkin_ws/src$  
3 $catkin_init_workspace
```

Now you want to source the workspace each time you open a new terminal. Therefore open the bashrc-file through the following command:

```
1 $sudo gedit ~/.bashrc
```

And add the following line at the bottom of the bashrc-file:

```
1 source ~/catkin_ws/devel/setup.bash
```

Now ROS should be installed. To learn more on ROS, check out the following tutorials;

- <http://wiki.ros.org/ROS/Tutorials>
- <https://cse.sc.edu/~jokane/agitr>

## C.3 Communicating between Raspberry Pi 2 and computer

### Getting WiFi on RP2

You need a WiFi USB adapter in order to communicate to the RP2 over WiFi. (Raspberry 3 will have WiFi built in). In this project, the following adapter was used: (TP-LINK TL-WN725N).

The wifi driver was installed on the RP2 using the following manual. [<http://askubuntu.com/questions/381574/drivers-for-tp-link-tl-wn725n-nano-usb-wireless-n-adapter>].

Note: If you are using Virtual Box on the laptop, then use bridged wifi network sharing in the settings of Virtual Box.

Now you should note the IP addresses for the Raspberry Pi 2 and Laptop. You can find the IPs by the command:

```
1 $ ifconfig
```

In the following example the IP and username of the laptop and RP2 is as follows:

Unit	Username	IP
RP2	ubuntu	192.168.0.232
Laptop	einar	192.168.0.107

Now edit the hosts file

```
1 $ sudo gedit /etc/hosts
```

Add the following line on the hosts file on both the laptop and RP2 host file:

```
1 192.168.0.232 ubuntu  
2 192.168.0.107 einar
```

Now edit the bashrc file

```
1 $ sudo gedit ~/.bashrc
```

Add the following lines on the hosts file on both the laptop and RP2 host file:

```
1 export ROS_MASTER_URI=http://ubuntu:11311
```

Where "ubuntu" refer to the username of the computer that will be the rosmaster. You need to comment out this line again if no longer which to have RP2 as the ROS master.

You should now check that you can both SSH and send ROS messages back and fourth between RP2 and laptop over internet. This can be checked this by doing step 1 in the following manual <http://wiki.ros.org/ROS/NetworkSetup>

In this particular setup, the RP2 did not receive enough power when powering other units such as the lidar and the Arduino . For this reason, the following current limit was changed on the RP2. The following line was added in /boot/config.txt

```
1 max_usb_current=1
```

### C.3.1 Arduino on ROS

In order to get the Arduino to ROS you should install Arduino IDE and the Rosserial package on your RP2 unit. Arduino IDE is a software for writing and uploading code to the Arduino, while the rosserial package is a protocol for transmitting standard ROS messages over devices such as network sockets and serial ports.

Run the following commands on the RP2:

```
1 $ sudo apt-get install arduino  
2 $ sudo apt-get install ros-indigo-rosserial
```

(For convenience, you may want to install these on your laptop as well).

You can now create and upload code to the Arduino. Enter Arduino IDE from the computer by the following commands:

```
1 ssh -X ubuntu@ubuntu  
2 arduino
```

For controlling the motor controllers the built in ROS-Servo example is an excellent one. See tutorial on this source. [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Servo](http://wiki.ros.org/rosserial_arduino/Tutorials/Servo)

## C.4 RP lidar and Hector-SLAM in ROS

**Installing RPLidar** Install the RPLidar through the following commands:

```
1 cd ~/catkin_ws/src  
2 git clone https://github.com/robopeak/rplidar_ros  
3 cd ~/catkin_ws  
4 catkin_make
```

If you are using Virtual-Machine you should at this point make sure that you have forwarded the USB-port to the virtual Machine.

Now test the RPLidar node.

```
1 roslaunch rplidar_ros view_rplidar.launch
```

The rviz visualization tool should now pop up, where red dotted datapoints represents observed data.

If problems occur you at this stage you might try:

```
1 sudo gpasswd --add ${USER} dialout
```

Or check out either of these sources: <http://blog.zhaw.ch/icclab/rplidar/> [http://wiki.ros.org/rplidar\]](http://wiki.ros.org/rplidar)

### Get the Hector-Slam and TF package

```
1 $ sudo apt-get install ros-indigo-hector-slam  
2 $sudo apt-get install ros-indigo-tf
```

Now locate the Hector\_mapping launch file

```
1 $ roscd hector_mapping  
2 $ cd launch  
3 $gedit mapping_default.launch
```

Add the following line

```
1 <node pkg="tf" type="static_transform_publisher" name="...  
  base_to_laser_broadcaster" args="0 0 0 0 0 /base_link /laser 100"/>
```

Also adjust parameters names of the file as explained in [http://wiki.ros.org/hector\\_slam/Tutorials/SettingUpForYourRobot](http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot)

Now locate the tutorial launch file

```
1 $ roscd hector_slam_launch  
2 $ cd launch  
3 $ gedit tutorial.launch
```

Add the following line

```
1 <include file="$(find rplidar_ros)/launch/rplidar.launch"/>
```

Also, without closing the editor, edit the following line:

```
1 <param name="/use_sim_time" value="true"/>
```

to

```
1 <param name="/use_sim_time" value="false"/>
```

Now test if Hector\_SLAM is able to process the lidar data by running the rp\_lidar and Hector\_SLAM nodes.

```
1 $roslaunch rplidar_ros rplidar.launch  
2 $roslaunch hector_slam_launch tutorial.launch
```

*If problems occur it may be that the time between the computer time and RP2 **may** need to be synchronized.*

*Also, copying from the PDF to the terminal/editor may loose some symbols, pay close attention to symbols like " and \_*

# Appendix D

## Tutorials: ROS

The following appendix contains links to recommended tutorials for beginners with ROS. The focus is on using ROS in Simulink and building models. Tutorials are from the ROS community or Mathworks, in the latter case they are often related to the Gazebo®.

<https://se.mathworks.com/help/robotics/getting-started-with-robotics-system-toolbox.html>

Contain 5 tutorials about ROS in MATLAB and communicating with a virtual machine.

<https://www.youtube.com/watch?v=IictXPCP5M4&t=618s>

First introduction with Simulink and ROS.

<http://se.mathworks.com/help/robotics/examples.html#d0e172>

5 Tutorials specific on Simulink and ROS.

<http://wiki.ros.org/>

The official wiki from ROS for all usage, not specifically MATLAB.

<http://se.mathworks.com/help/robotics/examples.html#d0e19>

More advanced use of the Robotics System Toolbox.

<https://se.mathworks.com/matlabcentral/fileexchange/56877-a---astar--search-algorithm-easy-to-use>

A\* search algorithm. Not specifically ROS.



# Appendix E

## Launch manual

### E.1 Deploy vessel for autonomous exploration

This section describes how to deploy the vessel for the operations seen in this thesis. It will be to the point, and assumes that components in the system are set up as they were when the project was terminated.

1. Make sure that the Arduino is connected as instructed in Table 2.1.
2. Connect the battery and the lidar to the Raspberry Pi 2. Subsequently place the lid as instructed in Section 3.1.1.
3. Connect to the MCLab network on the operator computer.
4. SSH into the RP2 and launch the RP2 nodes.

```
1 ssh ubuntu@ubuntu
2 cd catkin_ws/src
3 roslaunch LaunchNoLidar.launch
```

If not already performed, place the vessel on the water.

5. ssh into the RP2 and laucnh the mapping launch file according to desired resolution

```
1 ssh ubuntu@ubuntu
2 cd catkin_ws/src
3 roslaunch RES02256.launch
```

6. Launch of MATLAB node. The gridsize that is loaded prior to running this node should match that of the Hector-SLAM algorithms.

```
1 rosinit('ubuntu') (workspace commando)
2 load Frontier256.mat (workspace commando)
3 run Exploration_pathplanner.slx
```

7. Run the motion controller node. The motor should be set in neutral before this node is launched.

*Option a:* Connect a second computer to the system. From this the Simulink motion controller will be run. This yield more flexibility and more safety in case of unexpected behaviors or errors than the C++ compiled version of the motion controller yields.

```
1 rosinit('ubuntu')
2 run VesselParametersSet.m
3 run MotionControl.slx
```

*Option b* ssh into the RP2 and run the ROS\_controller node

```
1 $ ssh ubuntu@ubuntu
2 $ cd catkin_ws/src
3 $ rosrun motioncontroller compiled motioncontroller compiled node
```

Of these two options, *Option a* is by far the advised method and the one that has been applied the most in this thesis. There should be extra available computers in the MCLab

## TroubleShooting

- Check that you can ping the RP2 from the operator computer
- Make sure that you can SSH both back and forth between the RP2 and the operator computer. If not, there might either be a problem with the internet connection or the hosts file. Also, make sure that open-ssh is installed, and if your using Virtual Box, that you use Bridged Network.
- The IP addresses can change. Check the hosts file on both RP2 and operator computer is up to date.
- Check if you can echo on ROS signal sent from RP2 on the operator computer (and the opposite direction).

- Check that the bashrc file contains the following line: export ROS MASTER URI=http://ubuntu:11311 where ubuntu is the corresponding name to the RP2 IP as set in the hosts files.
- Check the voltage of the battery.
- Check that all pins are connected
- If one of the motors has stopped, check the lights on the motor-controller. They may signal an error as described in the following: <http://www.mtroniks.net/download.asp?ResourceID=1973>

Another helpful tool is to check if the ROS architecture is correct ,i.e., topics subscribe and publish correctly.

To generate the architecture, run:

---

```
1 $ rosrun rqt_graph rqt_graph
```

---

Compare the map with Figure E.1

NOTE: This will produce an error if ROS have not been initialized.

## E.2 Perform simulations

### E.2.1 Launching

In this manual, it is assumed that all nodes are run on operator computer, but some nodes could just as easily be run on the RP2. The mapping simulator has only been generated in C++ for a given Reference-map and a gridsize 256. For other Reference-maps and gridsizes, the Simulink version of the mapping simulator should be used. In that case, nodes should be launched individually and not by the use of a launch file.

- First, make sure that the line in the bashrc file that exports the ROS master is uncommented.
- Now perform the following:

```
1 rosinit (MATLAB WORKSPACE)
2 $ cd catkin ws/src
3 $ roslaunch Simulation.launch
4 Setstuff256 (running script from MATLAB workspace)
5 run Exploration_pathplanner.slx
```

### E.2.2 Run Simulink nodes independently

In order to edit the simulator one or several nodes needs to be left out of the roslaunch and launched through Simulink. To do so, follow the procedure:

```
1 $ cd catkin ws/src
2 $ gedit Simulation.launch
```

The content is:

```
1 <?xml version="1.0"?>
2 <launch>
3 <node pkg="vesselimulator" type="vesselimulator_node" name="...
   vesselimulator_node"/>
4 <node pkg="motioncontrolcompile" type="motioncontrolcompile_node" ...
   name="motioncontrolcompile_node"/>
5 <node pkg="path2setpoint" type="path2setpoint_node" name="...
   path2setpoint_node"/>
6 <node pkg="mappingsimulator" type="mappingsimulator_node" name="...
   mappingsimulator_node"/>
7 </launch>
```

Commenting out code, in a launch-file, is through:

```
1 <!--
2 !-->
```

Find the node ,e.g. "VesselSimulator.slx", edit it and run it as usual, prior to running "Exploration\_pathplanner.slx". When finished, build the model (CTRL+B), and imlement it intro the ROS architecture through:

```
1 $ ./build_ros_model.sh VesselSimulator.tgz ~/catkin_ws/
```



# Appendix F

## ROS architecture overview

Table F.1: Overview over nodes in the system in MCLab, see Figure F.1

Node name	Node function	Topics Subscribed to	Topics Published to	Described in
base_to_laser_broadcaster	Coordinate transformation between baselink of the robot and lidar position.	-	tf	<i>ROS-community</i> (2016)
Exploration_pathplanner	Responsible for generating path according to implemented guidance and exploration strategy	map Position	UNFGAPS MotorOnOff EtaSetList GAPS	Ueland (2016, Sec.4.3,4.4)
Hector2VesselPos	Transformation of position vector from Hector-SLAM to vessel coordinates. Includes a quaternion transformation	Poseupdate	Position	Ueland (2016, Sec.3.1.3)
Hector_SLAM nodes	Collection of nodes from the Hector-SLAM package. Responsible for performing SLAM based on the lidar data stream	scan tf	pose	<i>Kohlbrecher S, M. J.</i> (2014)
motioncontroller	Responsible for controlling the vessel to the desired setpoint. This is performed by publishing appropriate signals to the actuators.	Setpoint Motor_on_off Position	VesselSpeed nu Thrust1,Thrust2,Thrust3 a1,a2,a3	Ueland (2016, Sec.4.1)
Path2SetPoint	Generates an appropriate setpoint on the planned path	SetPointDist Path	SetPoint	Ueland (2016, Sec.4.3.3.1)
rlidarNode	Driver for the lidar. Generates range data in the 2D plane as registered by the lidar scann	-	scan	Ferguson (2016)
Scan2SetPointDist	Find the closest object based on lidar scan. Contains logic for finding setpoint distance	scan	SetPointDist ScanDistance	Ueland (2016, Sec.4.3.3.2)
serial_node	Node providing ROS communication protocols for the Arduino. Responsible for sending PWM signal to actuators, and for monitoring voltage level of battery and rotational speed of motors	Thrust1, Thrust2, Thrust3 a1 a2 a3	diagnostics BatterEncoder- ThrustRPM	<i>ROS-community</i> (2016)

**Table F.2: Topics in the system explained (Excluding topics of open source packages that are not directly utilized)**

Topic	Message Explained
a1 a2 a3	Actuator input for angle of thrusters, interpreted and transmitted by the Arduino
Acoustics	Measurement vector containing readings from proximity sensors
BatterEncoderThrustRPM	Measured revolution signal from each motor and measured voltage of battery. For recording/monitoring of data
GAPS	List containing GAPS that are candidates for investigation. For recording/monitoring of data
map	Binary occupancy grid represented as a vector. Contains information about the map as well, with the most important property being the resolution. Represents the environment that the lidar operates in
Motor_on_off	Toggle motor power between on and neutral
nu	Speed in surge, sway and yaw, as estimated by the observer
Path	List containing coordinates for the vessel path. 1:128 is x positions, 129:256 is y positions.
PoseUpdate	Position/Attitude of lidar as estimated by the Hector SLAM package. In general valid for 6 DOF
Position	Positions of vessel in surge, sway and yaw
scan	Data cloud representing intensities and ranges to objects in the vicinity of the vessel, as scanned by the lidar
ScanDistance	The shortest distance registered in the lidar data cloud. For recording/monitoring of data
SetPoint	Setpoint of vessel in surge, sway and yaw
SetPointDist	Distance to setpoint from the point on the path that is closest to the vessel to the setpoint forward in the path.
Thrust1, Thrust2, Thrust3	Actuator input for motor revolution. Interpreted and transmitted by arduino
UNFGAPS	List containing GAPS that have been examined. For recording/monitoring of data
VesselSpeed	Speed of the vessel in the direction it is travelling, as estimated by the observer

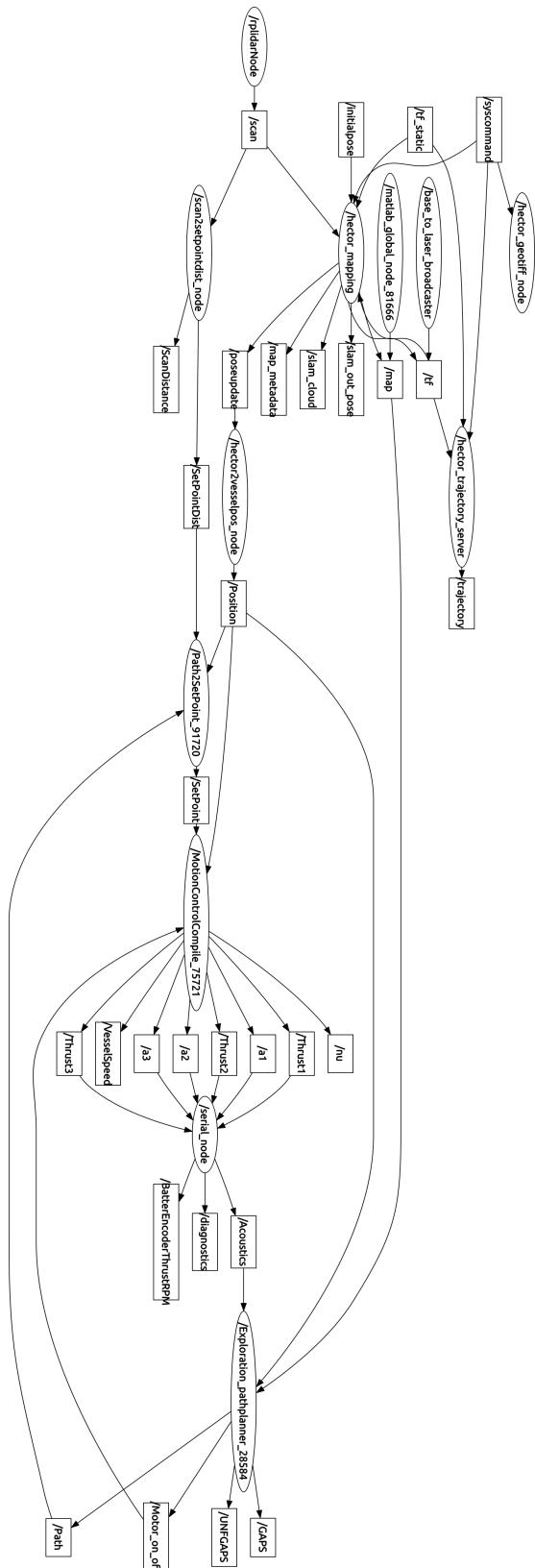


Figure F1: Overview over topics and nodes in the system, when running in the MCLab, as presented by the native `rqt_graph` tool in ROS.

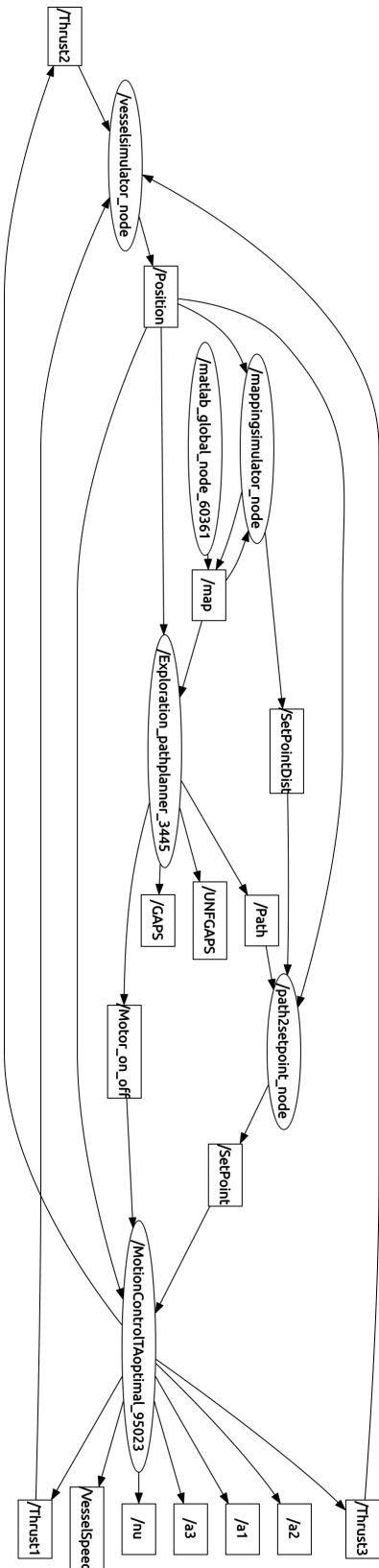


Figure F.2: Overview over topics and nodes in the system, when running as a simulation, as presented by the native rqt\_graph tool in ROS.