NTNU

Norwegian University of
Science and Technology

Marine Autonomous Exploration using a Lidar

Einar Skiftestad Ueland

Marine Technology
Submission date: June 2016
Supervisor: Roger Skjetne, IMT
Co-supervisor: Petter Norgren, IMT
Hans-Martin Heyn, IMT
Andreas Reason Dahl, IMT

Norwegian University of Science and Technology
Department of Marine Technology

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

MSC THESIS DESCRIPTION SHEET

Name of the candidate: Einar Skiftestad Ueland
Field of study: Marine control engineering
Thesis title (Norwegian): Marin autonom utforsking ved bruk av Lidar sensor
Thesis title (English): Marine autonomous exploration using a Lidar sensor

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.

The objective of this thesis is to use a Lidar sensor on the vehicle to scan the nearby unknown environment around the vehicle. Then this shall be used to autonomously survey and map a defined surface area for obstacles. When the entire area has been surveyed and a corresponding digital map has been created, it is possible to find optimal routes through the area. This shall all be done autonomously without intervention from a human operator.

Work description
1. Perform a background and literature review to provide information and relevant references on:
   - 2D Lidar sensors and how these have been applied to maneuvering in unknown terrains.
   - Unmanned surface vessels.
   - Autonomous mapping and path planning.
   - MC-Lab and the C/S Saucer model.

   Write a list with abbreviations and definitions of terms, explaining relevant concepts related to the background study and project tasks.

2. Modify the control system on the C/S Saucer, so that the system can utilize the Lidar for mapping and guidance. Develop and interface the corresponding payload system with the Lidar sensor to the control system.

3. Develop an autonomous guidance algorithm, interfaced to the Lidar sensor, that commands the C/S Saucer vessel to perform mapping of the terrain within defined boundaries of an area. Investigate methods and algorithms for path planning between locations specified by the mapping system. Provide references on the methods that are applied.

4. Develop a function that online visualizes the exploration process on the operator computer, and that allows for interaction by letting the user toggle between autonomous exploration and path planning to a specified location on the map.

5. Develop necessary control and observer algorithms for the C/S Saucer, that makes the vessel track the reference path provided by the autonomous guidance system.

6. Test the implemented system in the MC-Lab to verify that the system is working. Present and discuss the results.

Tentatively:

7. Develop a simulation model of the autonomous system, and simulate the system. Present and discuss the results.

8. Propose and implement several autonomous guidance strategies for mapping the terrain. Compare and contrast their 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 logical structure to give a clear exposition of background, 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: Title page, abstract, acknowledgements, thesis specification, list of symbols and acronyms, table of contents, introduction and background, problem formulations, scope, and delimitations, main body with derivations/developments and results, conclusions with recommendations for further work, 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. *natbib* 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: 15 January, 2016 **Due date:** As specified by the administration.
Supervisor: Roger Skjetne
Co-advisor(s): Andreas Reason Dahl

Trondheim, 10.06.2016

_____

Roger Skjetne (Supervisor)

NTNU – Trondheim
Norwegian University of
Science and Technology

Marine Autonomous Exploration using a Lidar

Einar Ueland

June 2016

MASTER THESIS
Department of Marine Technology
Norwegian University of Science and Technology

Supervisor: Prof. Roger Skjetne
Co-supervisor: Andreas Reason Dahl

Abstract

Autonomous exploration by the use of lidars on wheeled vehicles has been successfully performed in a number of scenarios. Yet, it is hard, if not impossible to find similar examples where marine surface vessels perform autonomous exploration by the use of lidars. The theory and methods developed for land-based surface vehicles have served as an inspiration for the development of the system seen in this thesis, where they are adapted to a marine control system.

This thesis considers the implementation of a lidar on a model-scale vessel, and the design of a control system that makes the vessel able to perform autonomous exploration of a small-scale marine environment. The completion of this system has involved the development of an autonomous system that merges exploration strategies, path planner, SLAM algorithms, motion controller, and a strategy for generating controller setpoints.

The experimental platform utilized in this project is the CS Saucer, a model-scale vessel built for testing in the Marine Cybernetics Laboratory at NTNU. Due to its circular form, the vessel should be able to respond fast and flexible to commanded control inputs, in any direction. These properties make it suited for autonomous exploration which involves a series of relatively rapid course changes.

The resulting system is demonstrated through both simulations and experiments. The individual elements of the system are all shown to function as desired, and in conclusion, and the set objectives are satisfactorily completed.

Trondheim, June 30, 2016

Einar Ueland

Acknowledgement

I would like to thank my supervisor Prof. Roger Skjetne for introducing me to the CS Saucer. He has provided me with feedback and support during the project period, and it was his idea to install the lidar on the vessel.

I would also like to thank my co-supervisor Andreas Reason Dahl. He has been supportive and provided detailed follow-up during the whole project period. This includes assistance in the Marine Cybernetics Laboratory and numerous fruitful discussions.

My classmate Rotem Sharoni, who has written a separate thesis on the control of an inverted pendulum by the use of the CS Saucer (Sharoni, 2016) has also been of great help, and should be thanked. He has in particular been involved with the electrical wiring of components on the CS Saucer.

Andreas Viggen and Stian Sandøy, two fellow students working in the Marine Cybernetics Laboratory also deserves a big thank. They are writing separate theses on the implementation and subsequent utilization of a Robot Operating System (ROS) based system on an ROV. They started their work with ROS prior to this project and have shared their experiences on ROS. Their help proved valuable as the ROS was implemented to the CS Saucer.

Senior engineer Torgeir Wahl, who maintains and organizes activities in the Marine Cybernetics Laboratory has been facilitating the author's work in the laboratory and deserves a thank as well.

I would finally like to thank the open source community of ROS, which has provided open source algorithms and packages vital for the success of this thesis. In particular, thanks should be offered to Mr. Stefan Kohlbrecher and his team at Technische Universität Darmstadt, who developed the Hector-SLAM algorithm, and published it in ROS under an open license.

Summary

This thesis reviews the complete design of a control system on a marine surface vessel, capable of autonomous exploration in small-scale marine environments. This involves the development of strategies for map exploration, path planning, navigation, and motion control. Further, the thesis describes how these components are merged into one autonomous system.

The experimental platform utilized in this project is the CS Saucer, a model-scale vessel built for testing in the Marine Cybernetics Laboratory at NTNU. The vessel has been extensively upgraded during this project. This includes the installation of new hardware and software, and significant improvements in the vessels capabilities of track following.

The system is installed on the Robot Operating System, a flexible platform with a large open-source community. This platform has made it possible to implement tools familiar within the robotics community such as algorithms for performing simultaneous localization and mapping.

Two strategies for exploration are considered in the thesis, where the Frontier Based Exploration strategy is the preferred one. In this strategy the vessel always moves to the edges between known and unknown area.

A version of the A* search algorithm has been implemented, responsible for planning paths to locations within the vessels environment. This algorithm is implemented such that node connections may span more than one cell, and with a scheme for weighting cells such that the vessel keeps a distance from walls. The generated paths are in general found to be satisfying.

A velocity control law that generates controller setpoints for the motion controller based on planned path and distance to nearby objects has been introduced to the system. The software components responsible for this operation iterates much faster than desired paths are recalculated. In this manner, the system is able to generate a steady stream of setpoints for the motion controller.

The resulting system has been tested through simulations, and subsequently verified in experiments performed in a basin facility. The experiments are well documented and are presented in a separate chapter of the thesis. A video demonstrating the successful experiments is referenced in the main body of the thesis. An interface where the operator can interrupt the exploration process and direct the vessel to any position in the explored map has also been created and successfully tested.

Sammendrag

Denne avhandlingen gjennomgår designet av et kontrolsystem på et marint overflatefartøy, i stand til å gjennomføre autonom utforsking i et småskala marint område. Dette innebærer utvikling av strategier for utforsking, korteste vei algoritmer, navigasjon, og bevegelseskontroll. Videre beskriver oppgaven hvordan disse komponentene er satt sammen til et autonomt system.

Systemet er installert på Robot Operating System, en fleksibel plattform med et stort bilbliotek av tilgjengelig åpen kildekode. Denne plattformen har gjort det mulig å implementere kjente verktøy innenfor robotikk, slik som algoritmer for utføring av samtidig lokalisering og kartlegging av fartøyet i sine omgivelser.

To strategier for utforskning har vært vurdert, hvor Frontier-basert utforskning er den strategien som har blitt foretrukket. Denne strategien innebærer at fartøyet alltid beveger seg til områder som befinner seg i overgangen mellom kjent og ukjent område.

En versjon av A* søkealgoritme, ansvarlig for ruteplanlegging til lokalisasjoner i kartet er implementert til systemet. Denne algoritmen er implementert slik at individuelle noder kan forbindes med noder mer enn celle unna, og med et en strategi for å vekte celler, slik at fartøyet holder en avstand fra hindringer i sjøen. Rutene som blir generert med denne algoritmen virker til å være tilfredsstillende.

En hastighetskontroll lov som genererer setpunkt for bevegelseskontrolleren basert på både planlagt rute og avstand til nærliggende objekter har blitt introdusert til systemet. Denne komponenten itererer mye raskere enn hva nye ruter blir planlagt, og genererer dermed en jevn strøm av oppdaterte setpunkter.

Det resulterende systemet har blitt testet ved simuleringer, og senere blitt verifisert gjennom eksperimenter utført i et basseng. Forsøkene er godt dokumentert og er presentert i et eget kapittel i avhandlingen. En video som viser de vellykkede forsøkene er referert i hoveddelen av oppgaven. Videre har et brukergrensesnitt, hvor operatøren kan avbryte utforskingen, ved å dirigere fartøyet til en posisjon i det kjente kartet har også blitt implementert, og testet med suksess.

Contents

List of Figures

List of Tables

List of Abbreviations

2D/3D. 2-Dimensional/3-Dimensional.

CS: "CyberShip". Prefix used for all the model vessels in the MC Lab.

GNC: Guidance Navigation and Control. System that process sensor data, and controls the movement of the craft.

HIL Hardware In the Loop. Simulation technique used to test systems in real time, where the control system is run on the same hardware as the experiments it simulates.

NED North-East-Down. Coordinate system defined relative to the earth's ellipsoid.

NI: National Instruments. Producer of virtual instrumentation software and automated test equipment.

NTNU: "Norges Tekniske og Naturvitenskaplige Universitet", Norwegian University of Science and Technology.

RPM Revolutions Per Minute. A measure of the frequency of rotation.

PWM Pulse-Width Modulation. Commonly used to control the power supply to electric motors.

ROS: Robot Operating System. Collection of software frameworks for robot software development.

SLAM: Simultaneous Localization and Mapping. The problem of creating and updating a map of the unknown environment, while simultaneously determining the objects position within it

SIL: Software-In-the-Loop. Simulation technique used to simulate systems. For software-evaluation.

USB Universal Serial Bus. Industry standard for communication protocols.

USV Unmanned Surface Vessel. Surface vessels that operate on the water surface without an operator on the vessels itself.

Nomenclature

d_{bot}	Diameter of vessel at bottom of hull
d_{top}	Diameter of vessel at top of hull
$\mathbf{M}_{\text{RB}}$	Rigid body mass and inertia matrix
$\mathbf{M}_{\text{A}}$	Added mass matrix
$\mathbf{D}$	Linear damping matrix
$\mathbf{D}(\boldsymbol{\nu})$	Non-linear damping matrix (Contains Linear Components)
$\mathbf{C}_{\text{RB}}$	Rigid body Coriolis matrix
$\mathbf{C}_{\text{A}}(\boldsymbol{\nu})$	Hydrodynamic Coriolis matrix
$\boldsymbol{g}(\boldsymbol{\eta})$	Restoring force matrix
$\boldsymbol{\tau}$	Thrust vector representing body-fixed forces.
$\boldsymbol{\eta}$	Position and attitude
$\boldsymbol{\nu}$	Linear velocities
$\Delta\psi_{\text{rel}}$	Angle between Basin-relative and Basin-fixed reference frame.
$\boldsymbol{\nu}$	Linear velocities
ψ	Heading of vessel
$\mathbf{R}$	Rotation matrix
$\boldsymbol{v}^{\text{b}}$	Vector in the Body-fixed Frame
$\boldsymbol{v}^{\text{s}}$	Vector in Basin-relative frame
u_{pwm}	Actuator input
$\boldsymbol{\eta}_d$	Desired position and attitude
$\hat{\boldsymbol{\eta}}$	Observer estimated position and attitude
$\hat{\boldsymbol{\nu}}$	Observer estimated velocities
$\hat{\boldsymbol{\nu}}$	Desired velocities
$\boldsymbol{\omega}$	Eigenfrequency of mass-damper-spring system
$\mathbf{Sat}_{\text{upper}}$	Upper saturations of velocity
$\mathbf{Sat}_{\text{lower}}$	Lower saturations of velocity
ζ	Dampening ratio
r_{t}	Radius from center of origin to thrusters
α_k	Angular position of the k^{th} thruster.
$\boldsymbol{f}_k$	Local force vector of the k^{th} thrusters
$\boldsymbol{l}$	Position vector of individual thruster
$\mathbf{T}$	Thrust allocation matrix
$\boldsymbol{\tau}_k$	Body-fixed forces from the k^{th} thruster.
$\mathbf{K_p}$	Gain matrix for proportional term in PD controller
$\mathbf{K_d}$	Gain matrix for derivative term in PD controller
X	Commanded force in surge
Y	Commanded force in sway
Z	Commanded force in yaw

$F_{\max}$	Max magnitude of force vector
$T_{\max}$	Max momentum
$\boldsymbol{b}$	Bias term
$\mathbf{T}$	Time constant related to bias term
$\mathbf{K}_k$	Tunable gain for injection terms in observer model
s	Node number s
$g(s)$	Cost of path from start node to node s
$h(s)$	Heuristic value. Estimate on cheapest path to goal from node s
$f(s)$	Estimate on total cost to reach goal via node s.
s_f	Frontier cell s_f
g'($\mathrm{s_f}$)	Extra cost of reaching frontier cell s_f.
w_d	Weight on distance between frontier and exploration goal
$d_{\mathrm{goal}}(s_{\mathrm{f}})$	Euclidean distance from frontier to exploration goal
$w(s)$	Weight on node s
g_{AB}	Cost of traversing the direct connection between node A and node B
d_{AB}	Euclidian distance between node A and B.
$\mathrm{d_{min}}$	Shortest distance that to an object, as recognized by the lidar.
d_{s_f}	The Euclidean distance from the vessels position to the frontier-cell s_f.
$\mathrm{d_{obj}}$	The Euclidian distance from node s to the closest occupied cell.

Chapter 1

Introduction

1.1 Motivation

Lidars are used for mapping and surveying within a wide array of fields, such as geology, seismology, and cartography. Within the field of robotics, the lidar can be utilized for locating and mapping the robot and its environment. With this information, the robot can navigate precisely relative to its environment and autonomously perform various operations. An example of a situation where this capability might be useful is for robots operating in disaster areas where they may be able to find and help human beings that are trapped.

The use of wheeled robots applying lidars to explore unknown environment is well documented. For marine surface vessels, it is hard to find projects that apply similar techniques. Thus, the apparent uniqueness of this projects is a large motivating factor in and of itself.

As the following examples illustrate, there exist several applications where a vessel's ability to autonomously explore and locate itself within a marine environment could be an advantage:

- The vessel could operate on a fish farm. Due to its mapping capabilities, it would be able to transverse and maneuver between buoys and cages while performing operations.
- The vessel could be coupled to an underwater vehicle that could use its position for localization. The vessel could simultaneously provide a means of communication between the underwater vessels and drones or satellites.
- Similar methods could be used for safely parking smaller vessels in crowded

harbors.

- The localization properties could be an advantage for vessels performing operations around marine structures, and that require precise positioning relative to nearby structures.

However, the usage of lidars on vessels operating in the open sea, cannot be expected to function as well as in the controlled laboratory environment. Challenges such as weather protection, roll and pitch affecting the lidar scan, and objects not being vertical, smooth walls need to be assessed before applying it to the mentioned examples. Even so, the establishment of a marine vessel capable of operating autonomously in the laboratory is the first step towards being able to utilize lidars for autonomous operations at sea.

Even if the use of lidars on marine vessels today is limited, many vessels are installed with radars, which can be seen as an analog to lidars. It is, therefore, the authors belief that the strategies developed for navigation and path-finding in this thesis are relevant, also on vessels operating with the use of global positioning systems and radars.

1.2 Background

1.2.1 Autonomous Mapping

Robots are increasingly being used to perform mapping missions in a wide array of fields such as within seabed mapping, space exploration, underground mine exploration and mapping of hazardous areas. Common to these is that the environment is not entirely known a priory. The robots need to handle the unknown environment, obstacle avoidance and exploration strategies to fulfill their missions. Also, global positioning systems are not always reliable, meaning that the robots need to be able to locate themselves based on what they sense.

In order to explore and interact with the environment, it is vital that the robot has a means of sensing the environment. Common types of sensors include radars, sonars, lidars and vision systems based on digital cameras.

In addition to an appropriate range sensor, in order to map unknown areas autonomously, the robots need to be able to perform the following four competencies:

- To build a map, while simultaneously locating itself within it. This is generally described as the SLAM problem.
- To evaluate which locations it should move to in order to fulfill its mapping objective.
- To plan obstacle-free paths to desired locations.
- To follow the planned path based on sensed data and inputs to its actuators.

1.2.2 Map Representation

The maps used in this thesis are represented as probabilistic occupancy grids, a data structure where the area that the vessel operates in is discretized into a grid. Each cell in the grid is represented by a probability for being occupied. The grid is also characterized by a given grid size and resolution. Occupancy grids of this form is a convenient form to represent detailed maps and was first introduced by Moravec and Elfes (1985).

1.2.3 Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is the problem of creating and updating a map of the unknown environment, while simultaneously determining the objects position within it.

There exist a wide array of algorithms that can apply data clouds from range scanners to performs SLAM. In this thesis, an existing software package is utilized to perform SLAM of the vessel. For this reason, the mathematical description of the SLAM problem is not presented in further details.

For a more detailed discussion on the SLAM problem, this thesis instead refers to the pioneering work on SLAM by Leonard and Durrant-Whyte (1991), and the work by Mullane et al. (2011) which discuss strategies and common algorithms for solving the problem.

1.2.4 The Path Planning Problem

The path planning problem as assessed in this thesis is the task of finding an efficient and collision-free trajectory from a start location to a goal location in a map. In order to map an unknown area, a robot needs to apply mapping schemes that involve a series of steps where the robot moves strategically between locations in its environment. This means that the path planning problem needs to be assessed.

The classic approach to solving the path planning problem is to apply graph theory. In this approach, one applies mathematical structures to set up a network of connections representing paths between nodes in the map. In this context, each node represents a discrete position in the map. These mathematical structures are usually referred to as cost maps.

The cost maps are typically built by identifying neighboring nodes that can reach each other in an obstacle free and straight path. Each of these connections is characterized by an estimate of the cost of traversing it.

Before cost maps can be constructed, nodes need to be generated and placed in the map. The probabilistic roadmap planner as described by Kavraki et al. (1996) is a popular method that generates nodes at random locations in the occupancy grids. Geraerts and Overmars (2004) provides further discussion of this approach and similar variants.

The strategy of letting each cell in the occupancy grid represent a node in the graph structure used in this thesis. The connections to neighboring nodes are now

very simple to asses. The drawback of this method is that one quickly gets a large amount of nodes, which increases the computer processing demands on the system.

By combining methods of constructing cost maps and shortest path search algorithms that utilize these cost maps, one gets path planners that can determine efficient paths between a starting configuration and goal of a robot.

Cormen (2009, Part VI), gives a comprehensive introduction to graph theory and presents common algorithms for finding the shortest path the graph structures. Popular algorithms include Dijkstra's algorithm (Dijkstra, 1959) , the A* algorithm (Hart et al., 1968), the Bellman-Ford algorithm (Ford, 1956), the D* algorithm (Stentz, 1994), and the Floyd-Warshall algorithm (Floyd, 1962).

1.2.5 Exploration Strategies

In this section, the two exploration strategies that have been implemented are introduced based on how they are described in the literature.

Aside from these, notable exploration strategies not reviewed include the Information gain exploration strategy (Stachniss et al., 2005) and the Feature-based exploration strategy (Newman et al., 2003).

1.2.5.1 Frontier-Based Exploration

In this approach, as proposed by Yamauchi (1997) the robot should always move to frontiers (edges) between explored and unexplored map. The method theorizes that by operating in this manner, the robot can map new territory rapidly.

The method is recognized for being relatively easy to implement, for its reliability, and for being relatively efficient. Simmons et al. (2000) extend the idea of frontier-based exploration to teams of robots working together, splitting frontiers among each other, while at the same time minimizing search overlaps. In the literature, it is further customary to group adjacent frontiers cells into sections where each section is considered a separate frontier.

1.2.5.2 Gap Navigation Tree Exploration

This approach, introduced by Tovar et al. (2004), is based on the robot identifying discontinuities in its field of vision. These discontinuities typically represent

corners or doorways and are in the literature labeled as gaps. The idea is that by moving to these gaps, the robot efficiently gains new information from its environment.

The method applies the Gap navigation tree, which is a graph structure that keeps track of discovered gaps and their properties. Specifically, the Gap navigation tree keeps track of which gap that preceded any new gap. Thus, a children/parent structure over identified gaps is generated. The algorithm further applies this structure during exploration to decide in what order it should explore identified gaps.

The algorithm applies the following main rules for maintaining and editing the Gap navigation tree:

1. If a gap disappears while the robot is examining another gap, it is deleted.

2. If a new gap appears while investigating another gap it is added as the child of the gap that is being investigated.

3. Two gaps that are children of the same gap and covers the same area when investigated are deemed redundant. The two gaps are subsequently merged.

An example by Nasir and Elnagar (2015) that illustrates how a simulated robot acts using the algorithm to explore a given map is included in Figure 1.1. The blue area in this figure is analog to what a 360-degree lidar installed on the robot will cover, given sufficient range of the lidar.

The example involves the following steps:

Event (a): Three discontinuities are at this point detected by the robot. This results in the identification of three new gaps.

Event (b): Gap a is examined. At this stage, five new gaps are identified. At the same time, the robot merges the gaps a and b because it recognizes that they are children of the same gap and covers the same area.

Event (c): The robot investigates the first children of a. At this point gap $a.1$ and $a.2$ are merged, and the gap $a.4$ is completely within the explored area and is deleted.

Event (d): The robot approaches the next unexamined children of a, namely $a.3$. While examining this gap, $a.5$ and c are completely within the visibility range and deleted. There are at this point no more nodes to investigate, and the mapping is complete.

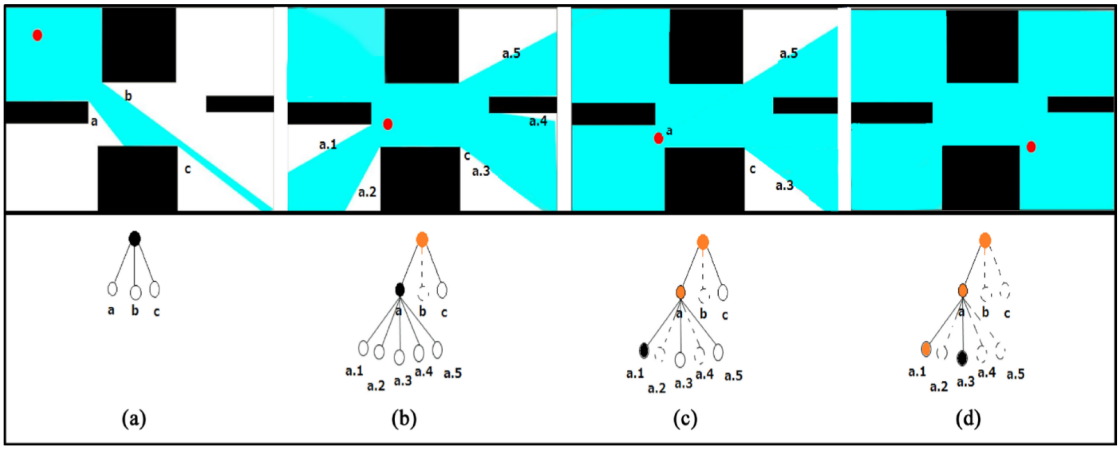


Figure 1.1: *Simulation using the Gap navigation tree method. Courtesy of Nasir and Elnagar (2015) .*

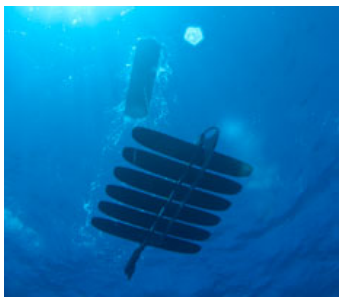
1.2.6 Unmanned Surface Vessels

Unmanned surface vessels (USV) operates on the water surface without the use of an operator on the vessel. The vessel type has not yet had the same level attention as unmanned crafts operating underwater, on land, or in the air (e.g. ROVs, autonomous cars, and drones). Still, the usage of USVs is a field in development where there currently is conducted a notable amount of research. One such research project is MUNIN (2016), a European research project performing a feasibility study on autonomous unmanned marine systems. According to MUNIN, expected future increase of transport volumes at sea, combined with a trend of slower steaming speeds will increase the future potential for autonomous vessels.

Even though the current use of USVs still is quite limited, there already exists several successful applications. Currently, one of their most prominent areas for use is within the field of oceanography where they are valuable for conducting exploration missions and collecting weather information. A notable example of such a vessel is the Wave Glider by Liquid-Robotics 2015, as seen in Figure 1.2a, which is entirely powered by wave and solar energy.

Several military uses can also be found, where applications include mine and anti-submarine warfare. An example of such a vehicle is the American Fleet-class unmanned surface vessel seen in Figure 1.2b. Other applications include vessels for surveillance or measurements and a number of experimental and academic vessels.

Note that the term robot, as applied in this thesis refer to the general description of autonomous crafts and thus also includes USVs.



(a) *The Wave Glider USV. Courtesy of Liquid Robotics.*

(b) *Fleet-class USV , Courtesy of Naval-technology (n.d.).*

Figure 1.2: *Examples of Marine Surface vessels*

1.2.7 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 functions by emitting a laser pulse that is reflected by the object it reaches. The returning signal is sampled by vision acquisition embedded in the lidar. The lidar measures the time that the light uses to return to it, and applies this information to produce a point cloud that 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 to be 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.

In addition to being applied to a wide array of research projects, lidars have been applied in various consumer products such as the Neato Vacuum-Cleaner (Neato-Robotics, n.d.).

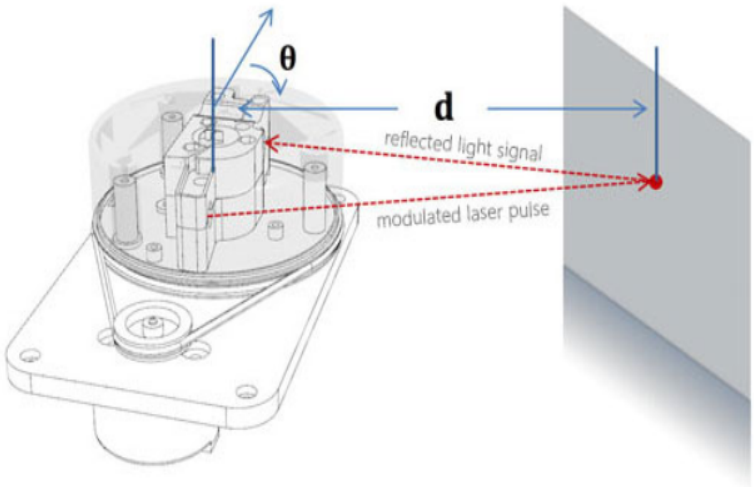


Figure 1.3: *Working principles of a 2D lidar illustrated with the RPlidar. Courtesy of Robotshop (2015) .*

(a) *Courtesy of Woo et al. (2014)*

(b) *Courtesy of Kohlbrecher et al. (2011)*

Figure 1.4: *Example of USVs installed with 2D-lidars*

1.2.7.1 Lidars on Marine Vessels

The implementation of lidars on marine surface vessels has been performed before, but most often by the use of 3D lidars, such as described by Leedekerken et al. (2014). Notable use of 2D lidar on marine vessels can be found in the international autonomous surface vehicle contest, Maritime Robot X. One example of a USV that participated in this contest by Woo et al. (2014), describes the vessel seen in Figure 2.1a. Another example of the use of lidars on marine vessel is the Hector-SLAM project (Kohlbrecher et al., 2011) which mounted a lidar on a USV as can be seen in Figure 2.1b.

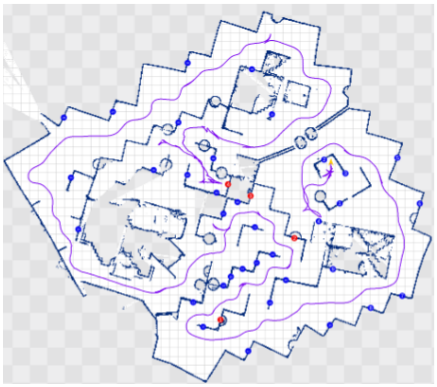
Though lidars have not been extensively used on marine vessels, it is quite common for these vessels to be installed with radars or sonars. This equipment relies on technologies similar to that of the lidar and can be seen as a full-scale analog to the lidar applied in this thesis. The main difference between the three technologies lies in what type of signal they emit and sample. Where the lidar emits and samples light signals, the sonar, and radar apply sound and radio waves respectively.

1.2.7.2 Lidars and SLAM

An analog to the marine surface vessel studied in this thesis is ground vehicles where 2D lidars have been utilized extensively. In particular, many sources describe mapping by the use of lidars on wheeled robots implemented with the Robot Operating System (ROS) and open source software.

One such popular open-source package that performs SLAM by the use of lidars

(a) *Robots used for map exploration in the Robcup 2015*



(b) *Corresponding map generated of the environment*

Figure 1.5: *Map generation by the use of Hector-SLAM in competition. Courtesy of Kohlbrecher et al. (2014b)*

is the Hector-SLAM package. Kohlbrecher et al. (2014a) provide a short review of the capabilities and motivation of the package.

In the Rescue Robot Leauge (Rescue-Robot-League,n.d.), a competition where robots operate in simulated disaster zones, the Hector-SLAM package has been applied with great success. Figure 1.5a displays team Technische Universität Darmstadt robots during this competition and Figure 1.5b displays the generated map in the same competition.

Other popular SLAM packages in ROS include the GMapping package (Gerkey, 2015). Unlike the Hector-SLAM package, this package requires odometry data, meaning a measure of the distance traveled by the vessel. Odometry data can be related to the rotations of wheels, and is thus simple to obtain for wheeled robots. On the marine vessel used in this thesis, however, odometry data is not available.

Since the Hector-SLAM package is popular, well tested and does not require odometry data, this is the package that has been chosen to perform SLAM on the implemented system

1.3 Objectives and Problem Formulation

The overall problem the thesis aim at answering is the following:

- Given the installation of a lidar on the unmanned surface vessel CS Saucer, how should a guidance navigation and control system be designed, making the vessel capable efficient and autonomous exploration?

The above problem formulation includes assessing relevant topics such as exploration strategies, path planning, motion control, user interface and how to mesh the individual components into one efficient system.

In order to provide a comprehensive answer to this problem, the following main objectives have been formulated:

- Implement a system on the CS Saucer that makes the vessel able to autonomously explore a small-scale marine environment (without operator interaction). The finished vessel should be able to plan paths and navigate within the explored map. This includes creating an interface where the user can specify desired destinations in the generated map.
- Investigate which control system, path planners and exploration strategies that are suitable for this kind of operation. In particular ideas and techniques used developed for land-based exploration and should be investigated and adapted to the marine environment.

For the purpose of accomplishing the above objectives, the sub-objectives attached in the very top of this thesis have been formulated in cooperation with the supervisor.

1.4 Scope and Delimitations

The thesis begins by performing a background and literature review on relevant methods and theory. If suitable, more than one solution to a particular problem are considered, and arguments are presented for which one that is preferred for the system. The thesis further describes how these methods are implemented, before the thesis finish off by reviewing the final system through both simulations and experiments.

The thesis revolves around the experimental platform the CS Saucer, where the system has been implemented. For this reason, considerable time is spent describing the vessel and how the system is implemented on it. The thesis attempts to explain each topic in general before it relates it to the implementation on the CS Saucer. The goal is both that the thesis be a contribution of interest on the investigated topics, and be designed in such a manner that future students may utilize it to continue the work on the vessel seamlessly.

The following address some of the limitations of the resulting system:

- The performed tests (and thus results) is limited to a controlled environment that satisfies the following properties:
  - There are no waves or current present during testing
  - Walls and hinders are in most cases solid and vertical.
  - The performed operations is small-scale, meaning that there are always some objects within the range of the lidar.

  For this reason, it is expected that the resulting system needs to be further developed in order for it to function at the open sea.
- In the implemented system, there is an external computer connected to the vessel over WiFi. From this machine, the essential exploration is node run. It is not trivial to compile this component to the onboard Rasberry Pi 2. One can, therefore, argue that the vessel is not autonomous in and of itself.
- The mapping process is performed in a static environment. In a dynamic environment, where objects move or drift, the system is expected to face new problems.

1.5 Thesis Contributions

The main contributions of this thesis are as follows:

- It has been demonstrated that the implemented system is successfully able to explore a small-scale marine environment by the use of a 2D lidar. This form of autonomous exploration has to the author's, best knowledge not been performed by a marine surface vessel before.
- Existing methods for autonomous exploration on wheeled vehicles have been successfully adapted and introduced to a marine control system.
- Suitable path-planner, exploration strategy, and control system that are shown to function well within the marine environment have been developed for the system based on existing methods.
- A simulation model capable of simulating exploration by a marine surface vessel within a closed area has been developed and successfully tested.
- A velocity control law for generating controller setpoints based on both planned path and distance to nearby objects has been developed and successfully implemented.

In addition to the above points, the following contributions relates to the development of the CS Saucer laboratory platform:

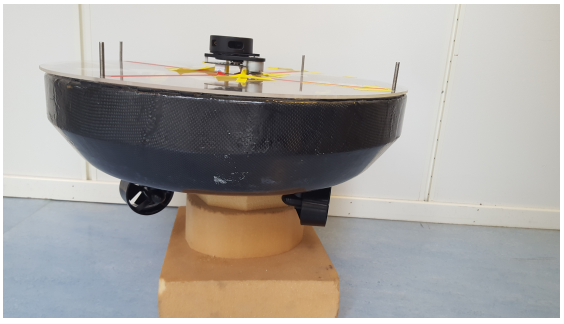
- Reference tracking has been significantly improved.
- A transition from LabVIEW to the Robot Operating System, which opens up for implementation of a large number of sensors and devices has been performed.
- Thorough documentation covering both the vessel and the utilization of the Robot Operating System in the local laboratory has been created.

Chapter 2

Experimental Platform: CS Saucer

2.1 Background

The system described in this thesis is implemented on the model-scale surface vessel, CS Saucer. The vessel is designed to be omnidirectional, with similar behaviors in surge and sway. It is further designed to be versatile and allows for many payload configurations. Containing the necessary hardware to run it, the mass of the vessel is about 3.4 kg. Its diameter is d_{top}= 548 mm at the top and d_{bot}=398 mm at the bottom. 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.



(a) *Vessel with lidar, as seen from the side*

(b) *Vessel and hardware*

Figure 2.1: *The CS Saucer*

Figure 2.2: *Initial setup of the CS Saucer*

2.1.1 Original setup

The vessel was built by Idland (2015), who designed it with a simple control system in the spring of 2015. In this setup, the control system on the CS Saucer was implemented on the National Instruments (NI) LabVIEW platform, where the embedded hardware device NI myRIO operated as the main processing unit on the vessel.

Figure 2.2 illustrates the play-load of the vessel as designed by Idland, while Figure 2.1b illustrates the payload utilized in this thesis. As is evident, the NI myRIO unit has been removed, while a series of new devices have been installed on the vessel.

2.1.2 Environment of Operation

The Marine Cybernetics Laboratory basin, installed with varying obstacles, constitutes the environment of operation for the CS Saucer.

2.1.2.1 Marine Cybernetics Laboratory

The CS Saucer is operated in the Marine Cybernetics Laboratory (MC Lab), a facility suited for testing of small marine vessels. The MC Lab is a small basin equipped with a wave maker, a towing cart, and a positioning system. During operations performed in this thesis, there are no waves, nor currents present in the basin. An image of the basin is provided in Figure 2.3.

The MC Lab dimensions (Length x Width x Depth =40 m x 6.45 m x 1.5 m) (NTNU/ime/labs, n.d.) are particularly relevant to this thesis, as they outline the area that the vessel may explore.

2.1.2.2 Obstacles in the Basin

To test, and demonstrate the vessels capability of autonomous exploration, varying obstacles should be present in the basin during operations.

Some of these obstacles are constructed by objects that are regularly present in the basin, independent of this thesis. These include camera mounts, other vessels, and mounts on the towing tank.

As can be seen in Figure 2.3 new objects, not naturally present in the laboratory have also been introduced to the basin. In the front of the basin, where the image is taken from, these hinders have been organized in a simple labyrinth-like manner. The hinders in the front are constructed by suspending PVC-blocs on wooden planks, while a cloth hanging over the basin bridge constitutes another obstacle.

During operations, all objects are kept static. For this reason, objects that otherwise would drift are fastened to the basin roof via thin ropes.

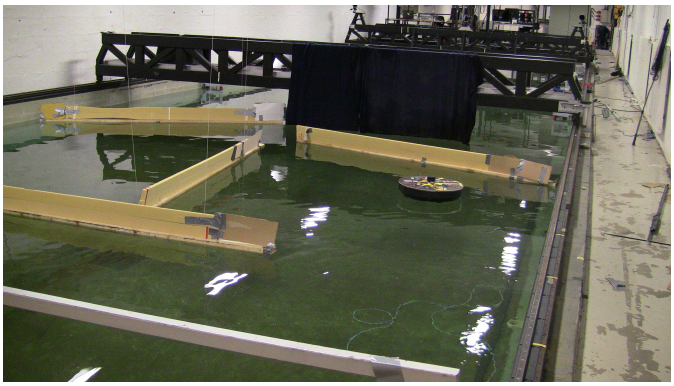


Figure 2.3: *The Marine Cybernetics Laboratory.*

2.2 Control System Redesign

This section presents a redesign of the vessels control system that involves transitioning the software platform from the original NI LabVIEW platform to the Robot Operation System (ROS).

The chapter explains the framework of the new system and how it is installed on the vessel. An overview of the software components compromising the new system, and how they are set up in a subscriber/publisher network is first presented once the individual parts of the system have been reviewed and can be found in Section 4.5

2.2.1 Motivation and Background

2.2.1.1 Review of Original Software Platform, LabVIEW

In the early phase of this project, the feasibility of implementing the lidar to the existing NI LabVIEW framework was reviewed. NI does not offer any official packages for interfacing lidars, but some third party contributors describe implementations of lidars to the NI environment. For example, NICommunityPostA (2016) provides a manual for the implementation of the Hokuyo-lidar to LabVIEW, which was successfully implemented to LabVIEW by the author. However, the method did not offer algorithms for processing and generating maps from the imported lidar data. In the end, it was concluded that the available resources for implementing and subsequently process the lidar data to the NI LabVIEW platform were insufficient.

2.2.1.2 The Robot Operating System

This section provides a brief introduction to ROS. The official ROS introduction page (ROS-community, c) offers a more comprehensive introduction, while the official tutorial by ROS (ROS-community, h) offers a detailed step-by-step tutorial on the use of ROS.

The Robot Operating System (ROS) was released as late as in 2007 and provides services that resemble that of operating systems, such as message parsing between processes, and package management. The versions available as of June 2016 runs on Unix-based platforms.

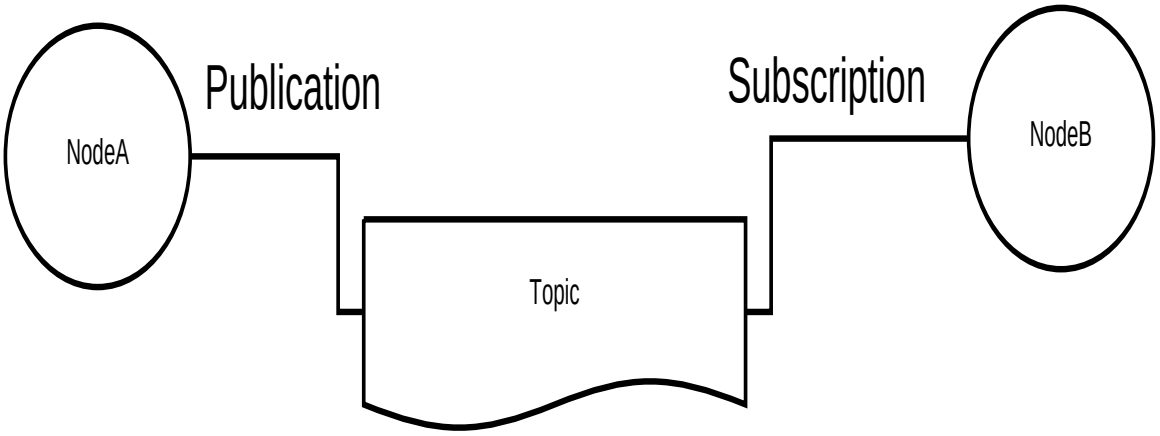


Figure 2.4: *ROS publisher/subscriber architecture.*

A ROS system consists of a number of small independent processes, called nodes, which perform computations. A full system is comprised of multiple nodes that communicate with each through messages, which are routed via topics in a publisher/subscriber framework. In this context, a topic represents the identifying name that describes the content of the messages that are passed.

Figure 2.4 illustrates the publisher/subscriber network for a simple system consisting of a single topic and two nodes. In this figure, Node-A is publishing a message to a topic, while Node-B is subscribing to the same topic. The two nodes act independently of each other in the sense that they are only connected through the common topic that they are publishing or subscribing on.

In general, a topic might have both several subscribers and publishers. Figure 4.26 displays the node/topic structure for the system designed in this thesis and provides an example of a complex system architecture.

Since this thesis presents a transition of CS Saucer's software platform from LabVIEW to ROS, it is appropriate to list some of the advantages of ROS as recognized by the ROS-community:

- Distributed computation
  ROS-community (d) explains how well-written nodes should make no assumptions on where in the network it runs. This means that ROS can be divided into small stand-alone parts, which makes it suitable for communicating between several computers.
- Software reuse
  The primary goal of the development of ROS is to support code reuse in robotics research and development (ROS-community, c). Smaller units of

nodes or packages are easily distributed and reused in multiple systems, which makes it ideal for an open source community and code reuse.

- Popularity of ROS
  ROS enjoy widespread support across the robotics community and in academic robotics research. In fact, O'Kane (2014) argues that ROS is becoming a de facto standard for robot software interoperability. As a result, the latest robotics hardware is supported by ROS and state of the art algorithms are available in ROS, and maintained by experts.

- Rapid testing
  As explained by O'Kane (2014) well designed ROS systems separate low-level control of hardware from high-level decision-making. This means that one can replace low-level programs and corresponding hardware with simulators in testing. ROS also provide simple means of recording and playing back sensor data, which is convenient when testing the system.

Based on the authors experience from this thesis, the following advantages that relate to students working in the MC Lab are added:

- In the MC Lab, there are often several students conducting separate projects on the same vessel. The modulated framework of nodes means that common parts of the software systems can be easily distributed and shared between the students. Also, the ROS framework offers opportunities for students to expand control systems on existing vessels, and at relative ease implement a wide array of devices. Examples of relevant devices that ROS supports include IMU systems (ROS-community, b) and Cameras (ROS-community, a).

- The relatively short period in which students conduct their master's thesis means that they don't have time, nor should focus on developing every underlying detail of a system for themselves. With ROS users do not have to *reinvent the wheel* and can utilize the extensive library available, or apply solutions that other students have implemented.

- Engineering students at NTNU are in general well educated on the use of MATLAB. For this reason, the relative ease at which MATLAB is applied in ROS appears to be of great advantage. Besides, by using MATLAB, students have simple means of recording and replaying all relevant data from experiments.

2.2.1.3 Review of New Software Platform, ROS

An early review on ROS revealed that interfacing lidar to the ROS framework would offer convenient open-source resources and algorithms for the implementation and processing of lidar data. Regarding interfacing the lidar, the capabilities of these resources appeared to far surpass the resources available in the LabVIEW environment. Combined with the presented advantages of ROS, it was for this reason, decided to transition the vessels software platform from LabVIEW to ROS, which involved replacing the myRio unit with a combination of a Raspberry Pi 2, Arduino and a Wireless USB Adapter.

2.2.1.4 Node Generation in ROS

Programming in the ROS framework is in general language independent, with the most popular languages being C++ and Python. The possibility of applying Simulink models as ROS nodes has been used a lot by the author in this project. In the implemented system the utilized nodes are a combination of open source packages and nodes produced by the author in either C++ or Simulink.

2.2.2 Software Architecture

Figure 2.5 schematically illustrate the signal flow between components in the new system architecture. The figure is not specific for the application seen in this thesis but applies to the system architecture of the CS Saucer (and other vessels set up in this manner) in general.

In the new system, the Raspberry Pi 2 is functioning as the onboard computer, capable of running multiple ROS nodes. The Arduino is connected to the system as a separate ROS node and is utilized for communication with onboard actuators, sensor, and devices. As is indicated in the figure, some devices such as the lidar may be connected directly to the USB port of the Raspberry Pi 2.

The setup allows for computers to be connected to the ROS framework over WiFi. These computers can in general run separate ROS nodes that are integrated to the same publisher/subscriber network as the Raspberry Pi 2. Other equipment in the basin such as the laboratory's position tracking system or other ROS based vessels may be connected to the ROS framework in the same manner.

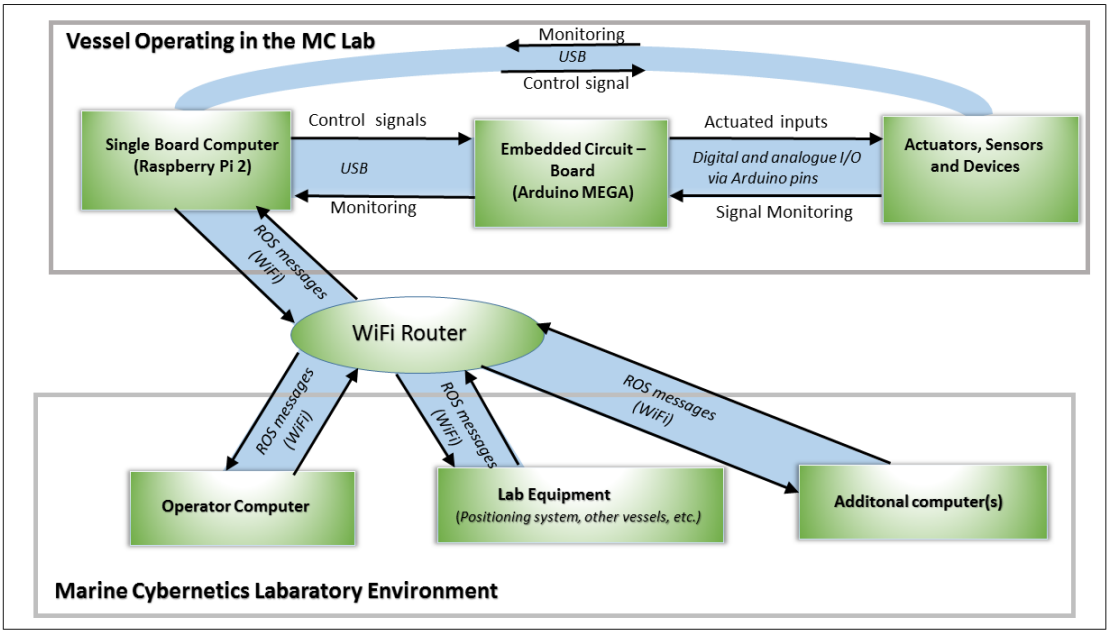


Figure 2.5: *General signal flow architecture between components on a system with the suggested architecture, set up in the MC Lab.*

2.2.3 Hardware Architecture

In order to facilitate for ROS and the operations performed in this project, substantial modifications have been made in the hardware architecture of the vessel. This involves removing the myRio unit and replacing it with a series of new devices.

Figure 2.6 illustrates the hardware architecture as installed on the vessel during the operations performed in this thesis. It further illustrates the signal and power flow between the various components. In this section each of the components, as shown in the figure is reviewed.

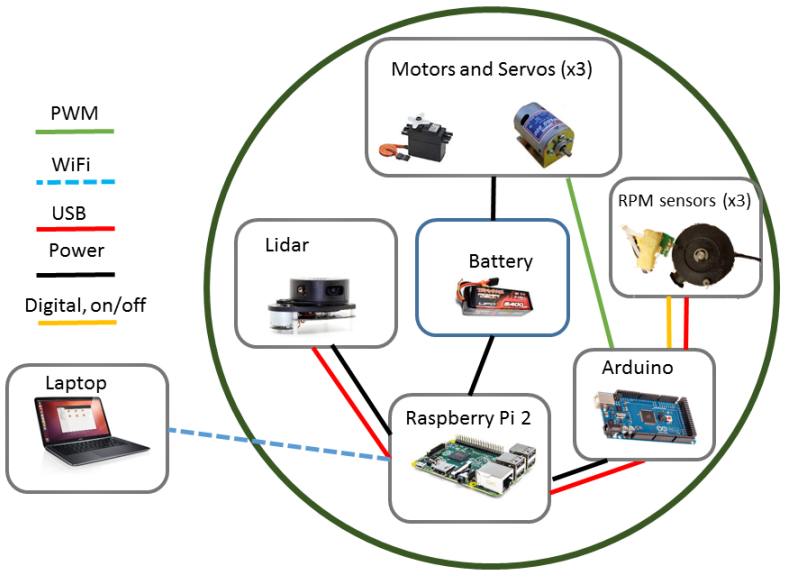


Figure 2.6: *Signal and power flow between hardware components on the system.*

2.2.3.1 Arduino Mega Embedded Circuit Board

The Arduino boards are low-cost embedded circuit boards that enjoy a large community which provides open source codes and drivers. The Arduino board implemented in this thesis is the Arduino Mega, which has a large number of available pins from where signals can be interpreted or transmitted.

In the implemented system, the Arduino is responsible for transmitting the appropriate PWM signals, as calculated by the control system to the motors and servos of the system. The PWM signals are sent from six separate pins on the Arduino board to each actuator.

The PWM signals that are transmitted have a frequency of 50 Hz and a duty cycle varying between 4.3 and 9.4 percent. The duty cycle that yields zero speed for the propellers, and their desired orientation are listed in Table 2.1. From the neutral position, the actuators are controlled by either decreasing or increasing the duty cycle.

In addition to transmitting PWM signal to the actuators, the Arduino is monitoring the rotational speed of each motor and the voltage of the battery. The full overview of the pin connections utilized in this project can be found in Table 2.1.

Table 2.1: *Pin connection overview*

Pin number	Corrosponding actuator/port	Duty Cycle at neutral position [% on]	Type
9	Rotational Speed Motor 1	6.85	PWM (Output)
10	Rotational Speed Motor 2	6.85	PWM (Output)
11	Rotational Speed Motor 3	6.85	PWM (Output)
3	Angle Servo-1	4.726	PWM (Output)
5	Angle Servo-2	0.0836	PWM (Output)
6	Angle Servo-3	0.0642	PWM (Output)
DGND	Ground Servos and motors	Not applicable	Ground
GND	Ground Battery/H.E. Sensors	Not applicable	Ground
V-In	Power	Not applicable	Power
D19	Hall Effect sensor 1	Not applicable	Digital, on/off (Input)
D20	Hall Effect sensor 2	Not applicable	Digital, on/off (Input)
D21	Hall Effect sensor 3	Not applicable	Digital, on/off (Input)
5v	Power for Hall Effect Sensors	Not applicable	Power
A0	Battery	Not applicable	Voltage sensor

The Arduino is connected to the Raspberry Pi 2 via USB and to the ROS framework through the ROS package Ros-Serial (ROS-community, f).

2.2.3.2 Raspberry Pi 2 Single-board Computer

The single-board computer Raspberry Pi 2 has been installed on the vessel. The board is the unit in which communication between the components of the system is routed through. It is responsible for running a number of ROS nodes, and thus for a lot of the computer processing performed on the system. It is running ROS on Ubuntu during experiments.

The Raspberry Pi 2 is connected to the lidar and Arduino via USB and to the local WiFi via a Wireless USB Adapter

2.2.3.3 Motors and Servos

The vessel is installed with three azimuth thrusters, each of which is driven by a separate Torpedo 800 motor. The azimuth thrusters can all be rotated, a motion controlled by three servos of the type Graupner Schottel drive unit II. The arrangement of the motors and servos are kept as they were in the original setup, designed by Idland (2015).

2.2.3.4 RPlidar 2D Lidar Scanner

The lidar installed on the vessel is of the type RPlidar, which is a low-cost laser scanner that performs 360 degrees, 2D scanning by the use of a rotating head. The rotational speed of the lidar is customizable from two to ten Hz, while it has a sampling frequency of 2000 Hz. The RPlidar has a range of 6 meters, which is about same length as the width of the MC Lab. The lidar has the advantage of being one of the cheapest on the market, while still being able to perform quite well. 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 the lid of the vessel, causing the horizontal plane that the lidar scan to be approximately 10 cm above the water surface during operations. The lidar is connected to the Rasberry Pi 2 over USB and is interfaced to the ROS framework through a separate node.

2.2.3.5 Hall Effect Sensors

Three Hall effect sensors have been installed on the vessel. The purpose of these sensors is to measure the rotational speed of the three motors and thus provide information that can improve the control of the vessel. The rotational speed of the motors is directly proportional rotational speed of the thrusters by a fixed gear ratio, and can thus be used to find the revolutions per minute (RPM) speed of the thrusters.

2.2.3.6 Battery

An 11.V, three cell 640 mAh lithium polymer battery from Traxxas, responsible for powering all devices on the vessel, is installed on the vessel. Fully charged it can power the vessel for a considerable amount of time (in the order of several hours). The battery can be damaged if the voltage drops under 11.1 V. For this reason, the voltage of the battery is monitored through the Arduino.

2.2.3.7 Laptop

A computer running ROS on Ubuntu is connected to the ROS framework during experiments. The computer has two main purposes. Firstly, it runs the essential

exploration node from MATLAB, and secondly, it is the station in which the operator can monitor and interface with the exploration process.

During the development of the vessel, the opportunity of running ROS nodes from the laptop has been used extensively. As the project has developed, these nodes have been transferred to the Rasberry Pi 2.

The exploration node responsible for generating a path for the vessel to follow has not been adapted for C++ code generation and cannot presently be run on the Raspberry Pi 2. Due to the node being quite computational intensive it has proved an advantage to have a relatively powerful computer run it.

Chapter 3

Modelling and Identification

3.1 Reference Frames

In this section, the reference frames that are utilized in the thesis, and transformations between them are investigated.

3.1.1 The Basin-Relative Reference Frame

This is the reference frame that is applied for local control of the vessel. The reference frame has its positive x-axis in the direction the lidar was pointing when the Hector-SLAM nodes were initialized, while its origin is located at the position of the vessel at initialization.

The reference frame is thus realigned relative to the basin in each new trial. This is a result of how Hector-SLAM initializes the coordinate system it represents the map in. The angular rotation between the Basin-relative reference relative to the basin, as defined in Figure 3.1 frame is denoted $\Delta\psi_{\text{rel}}$.

The heading of the vessel is defined as zero when thruster 1 is pointing along the x-axis relative to the center of origin of the vessel. The z-axis is pointing downwards, while the heading is defined as positive in the clockwise direction. The vessel position and heading in the Basin-relative frame in a vectorial format is given as follows:

$$\boldsymbol{\eta} = \begin{bmatrix} x & y & \psi \end{bmatrix}^T \tag{3.1}$$

Figure 3.1a illustrates the vessel in the Basin-relative reference frame. In this example, the vessel's location at initialization was at the origin, while its current

position is $\boldsymbol{\eta}_p = \begin{bmatrix} x_p & y_p & \psi_p \end{bmatrix}^T$

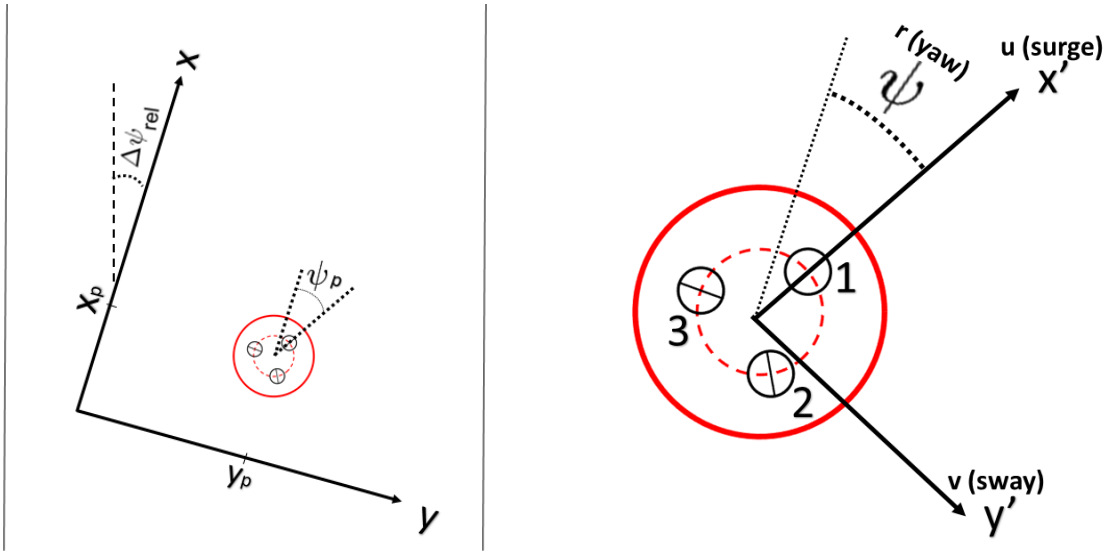
3.1.2 Body-Fixed Reference Frame

For control purposes, it is convenient to use the Body-fixed reference frame that moves along with the vessel. 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.

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

$$\boldsymbol{v} = \begin{bmatrix} u & v & r \end{bmatrix}^T \tag{3.2}$$

In Figure 3.1b the vessel is shown with the direction of the velocities indicated relative to the Body-fixed reference frame.



(a) *Basin-relative reference frame. Borders illustrate basin edges*

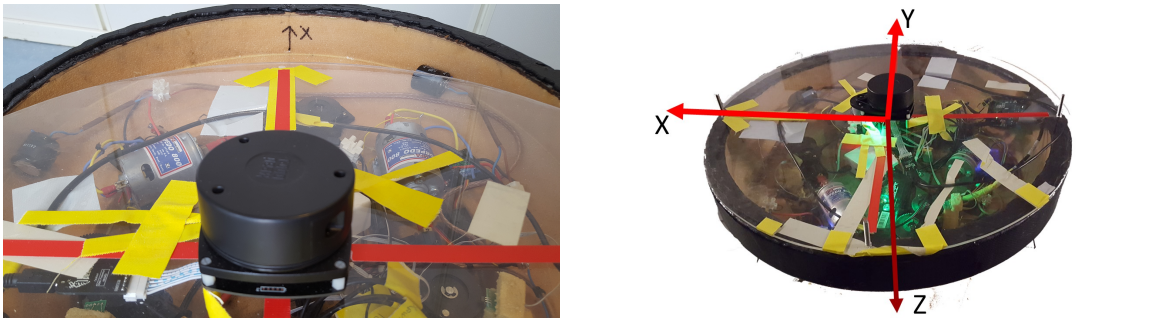
(b) *Body-fixed reference frame.*

Figure 3.1: *The CS Saucer and reference frames*

3.1.3 Transformation from Hector-SLAM Reference Frame to Basin-Relative Reference frame

In the coordinate system generated by the Hector-SLAM Package, the z-axis is pointing upwards, whereas it is pointing downwards in the Basin-relative coordinate system. The origin of the two coordinate systems is assumed to be aligned in in x- and y- direction, but since the z-axes is pointing in opposite directions, the positive y- axes and positive yaw are also pointing in opposite directions. In the implemented system a separate ROS node (*Hector2VesselPos-node*), is responsible for converting from the Hector-SLAM generated reference frame to the Basin-relative reference frame. This node also performs a conversion from quaternions to Euler angles.

When placing the lid on the vessel, it is crucial to align the positive x-axis of the lidar with that of the vessel. Markers on both the lid and the vessel indicate how the lid should be placed on the vessel. This is illustrated in Figure 3.2.



(a) Alignment of lid on the vessel (b) Vessel coordinate system indicated

Figure 3.2: *Lid placement relative to vessel coordinate system*

3.1.4 Transformation Between Vessel Reference Frames

When controlling the vessel in the Basin-relative reference frame, transformations back and forth to the Body-fixed reference frame need to be handled.

The transformation between the reference frames are given as follows:

$$\boldsymbol{v}^{\mathrm{s}} = \mathbf{R}(\psi)\boldsymbol{v}^{\mathrm{b}} \tag{3.3}$$

where $\boldsymbol{v}^{\mathrm{b}}$ is a vector in the Body-fixed reference frame, $\boldsymbol{v}^{\mathrm{s}}$ is a vector in the Basin-relative reference frame, and the transformation matrix $\mathbf{R}(\psi)$ is given by,

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

In order to transform the other way, the inverse $\mathbf{R}^{-1} = \mathbf{R}^T$ is used.

3.2 Equation of Motion

The identification of a mathematical model describing the vessel is essential for the simulations conducted in this thesis. This model is also used in the observer that is implemented to the motion control system of the vessel.

The equation of motion for a vessel at sea can be described in the following form (Fossen, 2011, Eq 6.1):

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

where,

- $\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}^T$ is the body-fixed velocities in surge, sway and yaw.
- $\boldsymbol{\nu}_r$ is the body-fixed velocities relative to local current in surge, sway and yaw. (The current is set as zero in this project such that $\boldsymbol{\nu}$=$\boldsymbol{\nu}_r$)
- $\mathbf{M}_{RB}$ and $\mathbf{M}_A$ is the vessel inertia matrix for the mass and added mass
- $\mathbf{C}_{RB}(\boldsymbol{\nu})$ and $\mathbf{C}_A(\boldsymbol{\nu}_r)$ is the vessel Coriolis centripetal matrix for the rigid body and added mass respectively
- $\mathbf{D}(\boldsymbol{\nu}_r)$ is the nonlinear damping matrix
- $\mathbf{D}$ is linear damping matrix
- $\boldsymbol{g}(\boldsymbol{\eta})$ is the hydro-static restoring matrix
- $\boldsymbol{\tau}_{\text{external}} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ is the external forces acting in surge, sway, and yaw, excluding those mentioned above.

In general rigid bodies operates in six degrees of freedom. These degrees of freedom are the linear motions in surge, sway and heave, and the rotation about these axes. The CS Saucer is assumed to be self-stabilizing by hydrostatic forces in heave, roll and pitch. The rotations in pitch and roll are further considered small, such that the movements in surge sway and yaw are not affected by the configuration in pitch and roll. For this reason, (3.5) is solved for only the three degrees of freedom, surge, sway, and heave.

The derivation of the equation of motion for the vessels was performed in the pre-project (Ueland, 2015). This thesis suffices by only presenting the resulting matrices:

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

$$\mathbf{C} = \mathbf{C}_{RB} + \mathbf{C}_A = \begin{bmatrix} 0 & -9.51r & 0 \\ 9.51r & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.7}$$

$$\mathbf{D}(\boldsymbol{\nu}) = \begin{bmatrix} 7.095|u| & 0 & 0 \\ 0 & 7.095|v| & 0 \\ 0 & 0 & 0.7095|r| \end{bmatrix} \tag{3.8}$$

$$\mathbf{D} = \begin{bmatrix} 1.96 & 0 & 0 \\ 0 & 1.96 & 0 \\ 0 & 0 & 0.196 \end{bmatrix} \tag{3.9}$$

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

There are no hydrostatic restoring forces present in surge, sway, and yaw, and thus, $\boldsymbol{g}(\boldsymbol{\eta}) = \mathbf{0}$. Since there is no wind, current, nor waves in the basin, the external forces $\boldsymbol{\tau}_{\text{external}}$ consists only of the actuated thruster forces in x-direction, y-direction and the moment. These forces are dependent on the commands that are given from the motion control system, which further described in Section 3.1.4 and Section 4.1.3

3.3 Mapping Actuator Input to Thrust Force

3.3.1 Background

In order to achieve decent control of the vessel, it is important to establish a good estimation of the relationship between actuator input and the resulting thrust forces. This is especially important for the CS Saucer which does not have any stabilizing fins and thus is more difficult to stabilize in yaw.

The actuated control input to each thruster u_{pwm} is a PMW-signal, which for the thrusters in use has a valid range (duty-cycle) of [0.0430: 0.0940]. In the Arduino code, a servo-library is utilized for mapping the duty-cycle to integer inputs in the range of [1,180]. It is this integer actuator input that is mapped to corresponding force in the forthcoming sections.

A mapping between actuator inputs and force had been performed prior to this thesis by Idland (2015). When applying this mapping to the implemented controller, only poor control of the vessel was achieved. In particular, the heading of the vessel was difficult to control.

For this reason, it was chosen to perform a new mapping between actuator input and thrust force.

3.3.2 Experimental Setup

The mapping was performed using a laboratory set-up where the vessel was fixed to strain gauges and linear springs. The setup in the lab is provided in Figure 3.3 while a schematic overview of the same setup can be seen in Figure 3.4

When the vessel is fixed in the setup, the springs have a pretension that is transferred to the strain gauges. By keeping the forces small, relative to the stiffness of the springs, the change of force on the strain gauges is expected to reflect the forces that the thrusters are acting on the vessel. These forces are measured through wires connected to the strain gauges. Since the vessel is fixed with pretension, the gauges can measure both positive and negative forces.

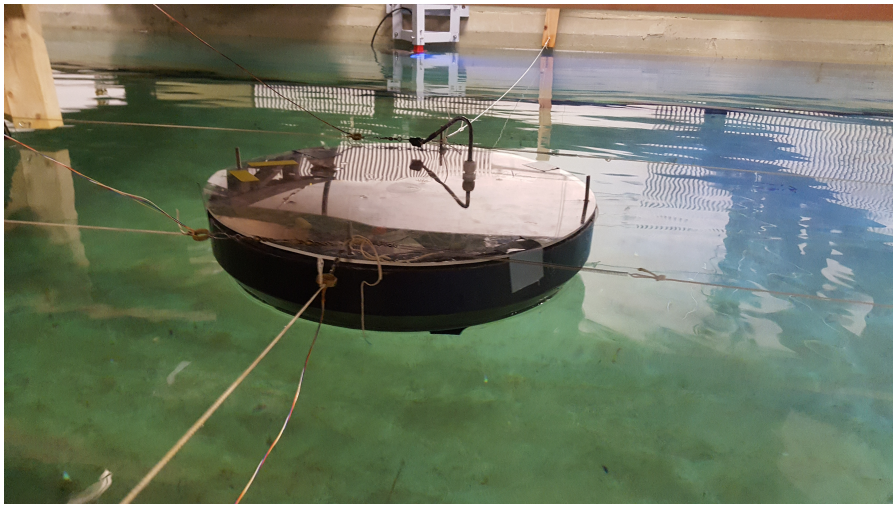


Figure 3.3: *The vessel in the basin rigged for actuator/force testing.*

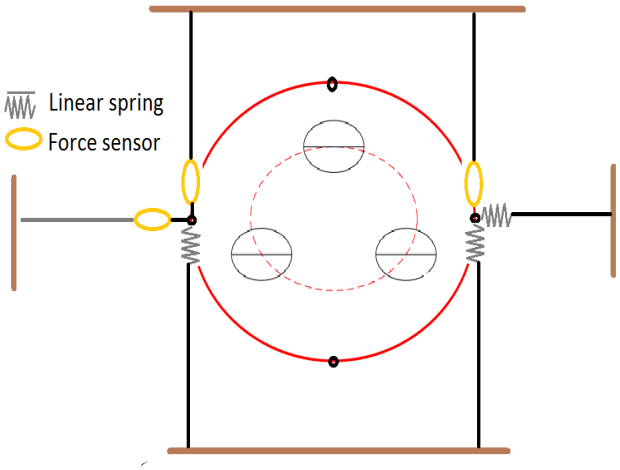


Figure 3.4: *Schematic overview of the vessel rigged for actuator/force testing.*

The test differs from the setup of Idland (2015), by having two suspensions points in x-direction instead of one. The idea is that the two suspensions will take up all moment, and by that make sure that there are no coupling between the total force in x- or y- direction and the moment.

In theory, x- and y- forces and moment can be calculated with this setup, but due to lack of sensitivity in the sensors, it was chosen to align all thrusters in x-direction and only measure total force in this direction.

3.3.3 Mapping Results

During testing, the rotational speed of each motor, battery voltage, force in each strain gauge and input PWM signal was logged. This section only presents the resulting mapping between actuator input and force. The raw data from the experiments can be found in the electronic appendix.

The resulting mapping between actuator input and force can be seen in Figure 3.5. Note that thruster 3 has its propeller blades flipped, which is causing the force to act in the opposite direction. It is evident that the resulting mapping resembles that of a second order function (If starting at the center and either increasing or decreasing actuator input). As seen in the figure, there is a relatively large dead band in the actuator input range of 72-88 were one do not thrust forces.

The data provided in Figure 3.5 is imported as vectors to the control system, where it is used to generate the appropriate mapping between actuator input and thrust force. In the control system, the desired force is mapped to actuator input through first-order interpolation on this dataset.

As a result of the described remapping, the availability of RPM data, and a re-design and tuning the vessels control system, the vessels performance in reference tracking has been significantly improved. Even so, the control of the vessel in yaw is still somewhat problematic, and during operations, the vessels heading typically oscillates around the desired heading.

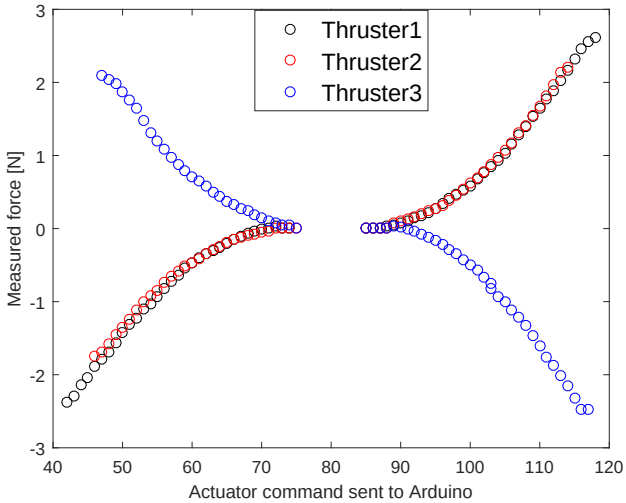


Figure 3.5: *Resulting mapping between actuator input and thrust forces*

Chapter 4

Guidance Navigation and Control

Guidance navigation and control (GNC) refers to the system that processes information from its environment and subsequently controls the movement of a craft. In the work by Fossen (2011), common strategies and algorithms for marine GNC systems are comprehensively reviewed.

Figure 4.1 display the control loop of the implemented system and what messages the individual components communicate with each other through. The gray and blue blocks constitute the GNC system of the vessel that is responsible for making the vessel achieve its control objective. The individual components of the system are reviewed in this chapter.

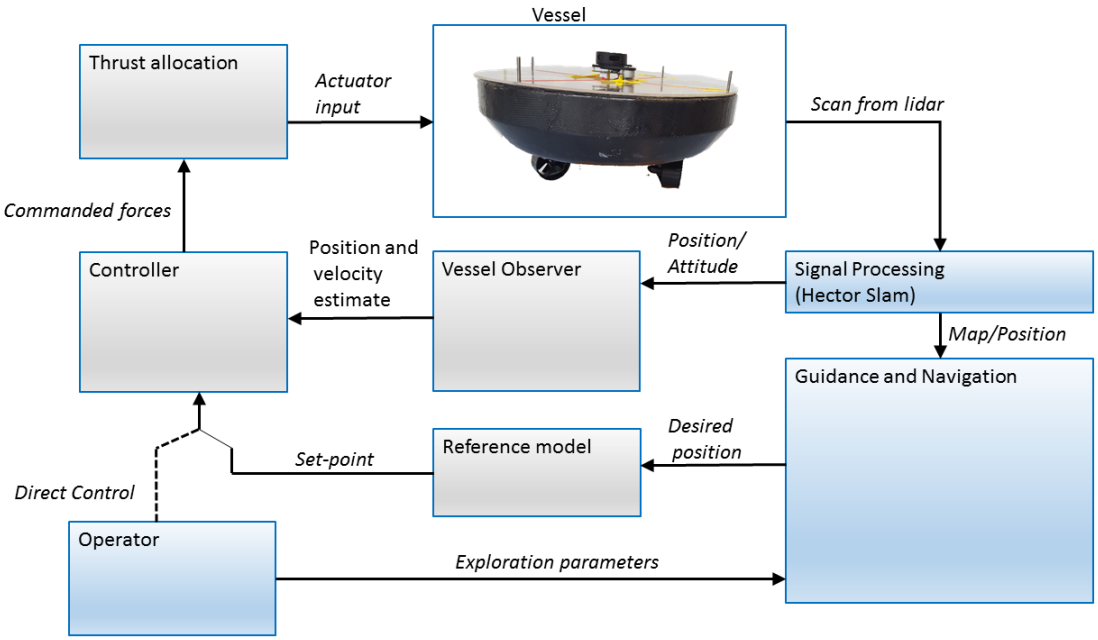


Figure 4.1: *Control loop of the implemented system.*

4.1 Motion Control System

The goal of the implemented motion controller is to make the vessel able to follow a reference track. The gray boxes in Figure 4.1 represent the motion control system of the vessel. In the implemented software the motion control system is implemented as separate node (MotionControl-node).

The reference model and observer are included in the motion control system on a basis of being implemented to the same ROS node. In other works, these blocks are often considered as a part of the guidance and navigation system respectively.

The equations as shown in Section 3.2 define the idealized state space of the system, and thus define the approximate state space one wants to control. Due to the rotation matrix, Coriolis matrix, and the nonlinear damping, this is a nonlinear state-space.

4.1.1 PD-Controller

A PD controller has been implemented on the vessel. In addition to being a popular and recognized controller, the PD controller is quite robust when the full dynamics of the system are unknown. There are still some uncertainties in the dynamical model of the vessel, and the PD controller, therefore, appears to be an adequate choice. The PD controller has one term that is proportional to the error between actual and desired position, and one term that is proportional to the speed of the vessel.

Since the vessel is rapidly re-planning and changing its course during operations, it is not desirable to have an integral term that builds up in between unpredictable course changes. A small integral term, limited by an anti-windup could have avoided this issue. However, the term was not deemed necessary.

The thrust allocation logic requires a body-fixed thrust $\boldsymbol{\tau}$ as input. To find this thrust, the controller first calculates the generalized force in the Basin-relative reference frame, then transforms it to the Body-fixed reference frame through the transformation matrix $\mathbf{R}(\psi)$.

The control law that determines the desired forces in the body-frame is as follows:

$$\boldsymbol{\tau} = \mathbf{R}(\psi)^T(\mathbf{K}_p)(\boldsymbol{\eta}_d - \hat{\boldsymbol{\eta}}) - \mathbf{K}_d\hat{\boldsymbol{\nu}} \tag{4.1}$$

where, $\boldsymbol{\tau}$ is commanded thrust in the body-frame, $\boldsymbol{\eta}_d$ is the desired position, $\hat{\boldsymbol{\eta}}$ is the observer estimated position in the Basin-relative frame, and $\hat{\boldsymbol{\nu}}$ is the observer-estimated velocities in the Body-fixed frame.

The gains of the PD controller are diagonal. Initial values were found by tuning the model in simulations, and subsequently adjusted during testing in the lab. This resulted in the following gains for the PD controller.

$$\mathbf{K}_p = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0.75 \end{bmatrix} \quad \mathbf{K}_d = \begin{bmatrix} 2.1 & 0 & 0 \\ 0 & 2.1 & 0 \\ 0 & 0 & 0.15 \end{bmatrix} \tag{4.2}$$

4.1.2 Reference Model

The controller should, in general, be able to handle any desired reference track in a smooth manner. Therefore, the desired position that the motion controller receives is filtered before it is sent to the PD block. The goal of this filter is to keep the reference signal the PD block receives both smooth, and within the limits of the vessel's capabilities.

More specifically the filter should ensure that the desired position $\boldsymbol{\eta}_d$ is within the limits of what the vessel can follow over time. This in turns smoothens and limits the PD terms and thus improves overall performance in path following.

A popular way to implement such a filter is to use a third order reference system with a low-pass filter in cascade with a mass-damper-spring system (Fossen, 2011, Ch 10.2.1).

This type of cascaded reference filter can be described by the following transfer function.

$$\frac{\text{Reference position}}{\text{Desired position}} = \frac{\boldsymbol{\eta}_{r(i)}}{\boldsymbol{\eta}_{d(i)}} = \frac{\omega_i^2}{(1 + (s/\omega_i))\left(s^2 + 2\zeta_i\omega_i + \omega_i^2\right)} \tag{4.3}$$

for (i=1,2,3) representing position in surge, sway and yaw respectively. The reference position is the setpoint that the PD block receives and the desired position is the setpoint that the motion controller receives from the guidance system.

By also including saturations elements in the filter, the maximum speed and acceleration of the reference signal can be limited. In the implemented reference model this type of saturation is applied on the speed of the reference signal.

The resulting reference model, as implemented to the vessel can be seen in Figure 4.2. The parameters in the filter are tuned to suit the capabilities of the vessel. The chosen values are shown in (4.4), while Figure 4.3 displays how a given time-series of desired position is handled by the filter.

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.9 \\ 1.05 \end{bmatrix} \quad \mathbf{Sat}_{\text{upper}} = \begin{bmatrix} 0.56 \\ 0.56 \\ 1.12 \end{bmatrix} \quad \mathbf{Sat}_{\text{lower}} = \begin{bmatrix} -0.56 \\ -0.56 \\ -1.12 \end{bmatrix} \quad \zeta = 0.8 \quad (4.4)$$

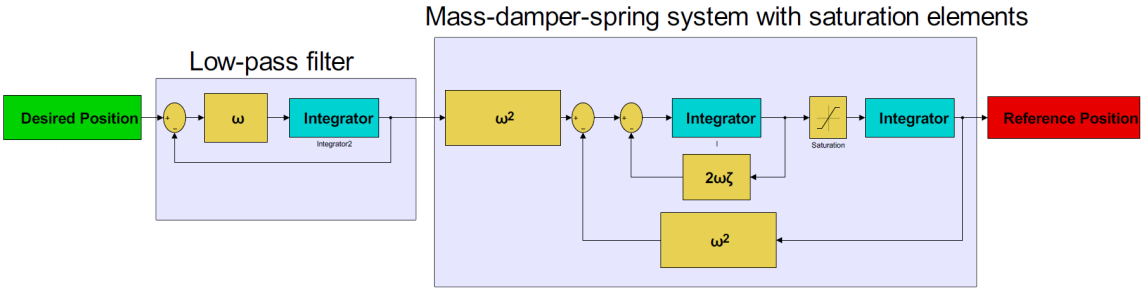


Figure 4.2: *Implemented reference model*

.

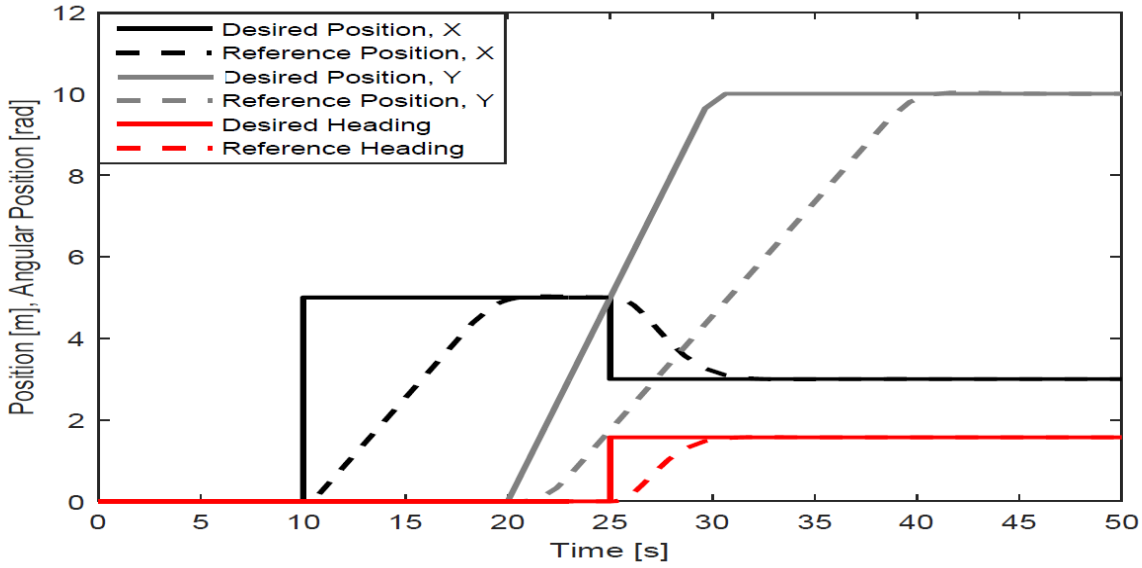


Figure 4.3: *Behaviour of reference model for a time-series of reference positions .*

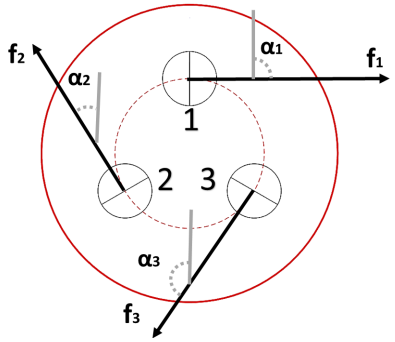


Figure 4.4: *Position and orientation of the thrusters .*

4.1.3 Thrust Allocation

The vessel is equipped with three thrusters, all placed tangentially to a circle about the center of origin with a radius of r_t= 0.138 m. The three thrusters are placed symmetrically on this circle with a spacing of 120 degrees.

Since the thrusters can be rotated, the vessel is over-actuated. In this thesis however, only fixed thruster angles are considered, resulting in a fully actuated system. A review on the use of flexible thruster angles was performed in the pre-project (Ueland, 2015).

Figure 4.4 presents the thruster orientations in their fixed position, where the black arrows that extends from the thrusters illustrate the positive force direction. Relative to the x' axis of the body-frame the three thrusters have the following orientations:

$$\begin{aligned} \alpha_1 &= 90^\circ \\ \alpha_2 &= -30^\circ \\ \alpha_3 &= -150^\circ \end{aligned} \tag{4.5}$$

The decomposed position vector, relative to the vessel's center of origin $\boldsymbol{l}_k = \begin{bmatrix} l_{kx} & l_{ky} \end{bmatrix}^T$ for the k^{th} thruster is given as follows:

$$\boldsymbol{l}_1 = \begin{bmatrix} r_t \\ 0 \end{bmatrix} \qquad \boldsymbol{l}_2 = \begin{bmatrix} r_t \cos(2/3\pi) \\ r_t \sin(2/3\pi) \end{bmatrix} \qquad \boldsymbol{l}_3 = \begin{bmatrix} r_t \cos(4/3\pi) \\ r_t \sin(4/3\pi) \end{bmatrix} \tag{4.6}$$

The decomposes of force vector $\boldsymbol{f}_k = \begin{bmatrix} f_{kx} & f_{ky} \end{bmatrix}^T$ for the k^{th} thruster is given as follows:

$$\boldsymbol{f}_k = \begin{bmatrix} f_k \cos(\alpha_k) \\ f_k \sin(\alpha_k) \end{bmatrix} \tag{4.7}$$

where, f_k and α_k represent the force magnitude and angle of the of the k^{th} thruster respectively.

The corresponding thrust loads for the kth thrusters are thus:

$$\boldsymbol{\tau}_k = \begin{bmatrix} \boldsymbol{f}_k \\ \boldsymbol{l}_k \times \boldsymbol{f}_k \end{bmatrix} = \begin{bmatrix} f_{kx} \\ f_{ky} \\ l_{kx} f_{ky} - l_{ky} f_{kx} \end{bmatrix} \tag{4.8}$$

where $\boldsymbol{\tau}$ is the thrust force in the body-frame of the vessel.

By inserting the values for $\boldsymbol{f}_k$ and $\boldsymbol{l}_k$, and by simplifying the resulting expression by the use of trigonometric relations, the formulas for the thrusts reduce to:

$$\boldsymbol{\tau_1} = \begin{bmatrix} f_1 \cos(\alpha_1) \\ f_1 \sin(\alpha_1) \\ f_1 r_t \end{bmatrix} \qquad x\boldsymbol{\tau_2} = \begin{bmatrix} f_2 \cos(\alpha_2) \\ f_2 \sin(\alpha_2) \\ f_2 r_t \end{bmatrix} \qquad \boldsymbol{\tau_3} = \begin{bmatrix} f_3 \cos(\alpha_3) \\ f_3 \sin(\alpha_3) \\ f_3 r_t \end{bmatrix} \tag{4.9}$$

With the generalized force vector $\boldsymbol{\tau} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$, and the individual thrust force vector $\boldsymbol{f} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ one now gets the following relationship between local thrust force and body-fixed forces on the vessel.

$$\boldsymbol{\tau} = \mathbf{T}\boldsymbol{f} \tag{4.10}$$

where the thrust allocation matrix $\mathbf{T}$ for fixed thruster angels is given as:

$$\mathbf{T} = \begin{bmatrix} 0 & \sin(2\pi/3) & \sin(4\pi/3) \\ 1 & \cos(2\pi/3) & \cos(4\pi/3) \\ r_t & r_t & r_t \end{bmatrix} \tag{4.11}$$

The desired forces on each thruster can now be determined from the desired body-fixed forces by applying the inverse of the thrust allocation matrix.

$$\boldsymbol{f} = \boldsymbol{T}^{-1}\boldsymbol{\tau} \tag{4.12}$$

The resulting local thrust force vector $\boldsymbol{f}$ is finally mapped to appropriate actuator inputs as described in Section 3.3.

4.1.4 Force Saturations

The reference model described in Section 4.1.2 should limit the forces acting on the vessel, such that the vessel is always able to follow the given reference signal. Even so, for safety measures, a saturation logic that limits the thruster forces acting on the body has been implemented according to the following logics:

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

$$X_{\text{sat}} = \begin{cases} c_k X, & \text{for } c_k < 1 \\ X, & \text{for } c_k \geq 1 \end{cases}$$

$$Y_{\text{sat}} = \begin{cases} c_k Y, & \text{for } c_k < 1 \\ Y, & \text{for } c_k \geq 1 \end{cases}$$

$$Z_{\text{sat}} = \begin{cases} \text{sign}(Z) T_{\max}, & \text{for } |Z| \geq T_{\max} \\ Z, & \text{for } |Z| < T_{\max} \end{cases} \tag{4.13}$$

where, [X,Y,Z] is commanded force in surge, sway and yaw

This logic ensures that the forces acting on the vessel are evenly saturated and that the saturation does not alter the orientation of the applied force vector.

4.1.5 Observer Design

An observer responsible for filtering out noisy data, and for estimating the body-fixed velocities of the vessel is implemented to the motion controller. This observer is a version of the nonlinear passive observer described by Fossen (2011, Ch 11.4). In addition to reconstructing and estimating the position and velocities, this observer can perform dead reckoning. However, the implemented system does not detect loss of signal, and the observer can thus not be expected to perform dead reckoning in its current form.

The observer suggested by Fossen is shown in Figure 4.5. As opposed to this version, the partition of frequencies into wave dependent and wave independent frequencies has not been performed in this project. This is for the simple reason that there are no waves in the basin during operations.

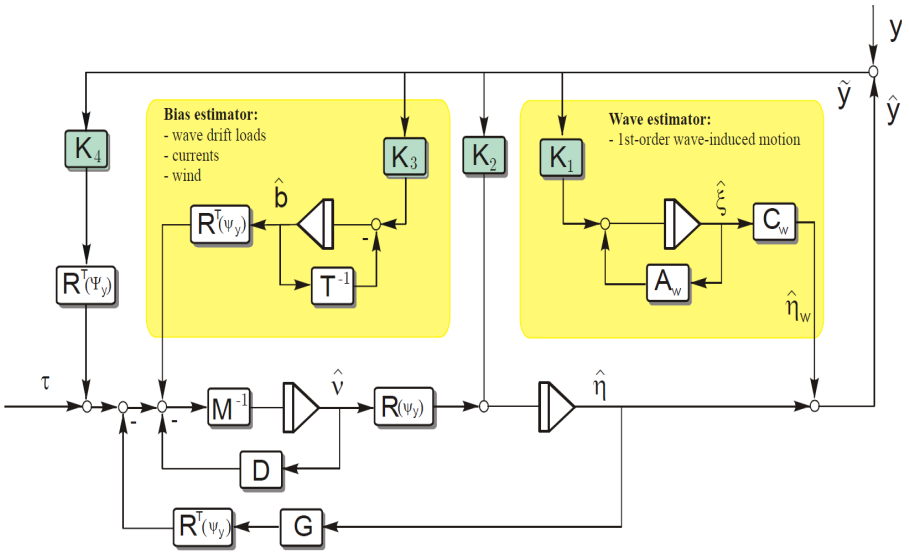


Figure 4.5: *General block description of the nonlinear passive observer. Courtesy of Fossen (2011)*

For appropriate choice of gains, the nonlinear passive observer guarantees global convergence of all estimation errors (including the bias term). The term passivity implies that the phase of the error dynamics is limited by 90 degrees, which yield good stability properties. The stability of the observer dynamics could be proved using Lyapunov stability theory (Fossen, 2011, Ch 11.3).

The observer is based on a simplified design model, where unmodeled dynamics is accounted for by the bias term $\boldsymbol{b}$:

$$\dot{\boldsymbol{\eta}} = \mathbf{R}(\psi)\boldsymbol{\nu} \tag{4.14a}$$

$$\dot{\boldsymbol{b}} = \text{-}\mathbf{T}^{-1}\boldsymbol{b} \tag{4.14b}$$

$$\mathbf{M}\dot{\boldsymbol{\nu}} = -\mathbf{D}\boldsymbol{\nu} + \mathbf{R}^T(\psi)\boldsymbol{b} + \boldsymbol{\tau} \tag{4.14c}$$

$$\boldsymbol{y} = \boldsymbol{\eta} \tag{4.14d}$$

where,

- Equation (4.14a) is the rotation between body-fixed and basin-relative velocities
- Equation (4.14b) is the bias model which accounts for slowly varying forces and moments due to second order wave loads, currents and winds. This term also account for unmodeled dynamics. $\mathbf{T_b}$ is a diagonal time constant matrix.

- Equation (4.14c) is the body-fixed equation of maneuvering.
- Equation (4.14d) is the vessel response.

Now defining the error dynamics as $\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta} - \hat{\boldsymbol{\eta}}$, $\tilde{\boldsymbol{b}} = \boldsymbol{b} - \hat{\boldsymbol{b}}$, $\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu} - \hat{\boldsymbol{\nu}}$ and $\tilde{\boldsymbol{y}} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ and applying injections terms, one gets the model implemented in Figure 4.5 (excluding the wave dynamics)

$$\dot{\boldsymbol{\eta}} = \mathbf{R}(\psi)\hat{\boldsymbol{\nu}} + \mathbf{K}_2\tilde{\boldsymbol{y}} \tag{4.15a}$$

$$\dot{\boldsymbol{b}} = -\mathbf{T}^{-1}\hat{\boldsymbol{b}} + \mathbf{K}_3\tilde{\boldsymbol{y}} \tag{4.15b}$$

$$\mathbf{M}\dot{\hat{\boldsymbol{\nu}}} = -\mathbf{D}\hat{\boldsymbol{\nu}} + \mathbf{R}^T(\psi)\hat{\boldsymbol{b}} + \boldsymbol{\tau} + \mathbf{R}^T(y_3)\mathbf{K}_4\tilde{\boldsymbol{y}} \tag{4.15c}$$

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{\eta}} \tag{4.15d}$$

Note how $\mathbf{K}_1$ is skipped in these equations. This is done in order to keep the syntax equal to that of Fossen (2011), where $\mathbf{K}_1$ is reserved for the wave dependent dynamics.

The parameters $\mathbf{T}$, $\mathbf{K}_2$, $\mathbf{K}_3$ and $\mathbf{K}_4$ are diagonal structured matrices. Through the guidelines presented by Sørensen (2013, Ch 8.2) and tuning, the following values have been found:

$$\mathbf{T} = \text{diag}(0.1, 0.1, 0.1) \tag{4.16a}$$

$$\mathbf{K}_2 = \text{diag}(0.25, 0.25, 0.2) \tag{4.16b}$$

$$\mathbf{K}_3 = \text{diag}(0.2, 0.2, 0.15) \tag{4.16c}$$

$$\mathbf{K}_4 = \text{diag}(5, 5, 0.5) \tag{4.16d}$$

As an alternative to the nonlinear passive observer, the Extended Kalman filter could have been implemented. One advantage of using the nonlinear passive observer as opposed to the Extended Kalman filter is that there are far fewer parameters to tune. The Extended Kalman filter on marine systems is presented in the work by (Sørensen, 2013, Ch 8). Chapter 6 of the same work, further presents the general purposes of an observer on a marine control system.

In the implemented observer in, the choice has been made to include the nonlinear dynamics $\mathbf{C}(\hat{\boldsymbol{\nu}})\hat{\boldsymbol{\nu}}$ and $\mathbf{D}(\hat{\boldsymbol{\nu}})\hat{\boldsymbol{\nu}}$ to the equation of maneuvering (4.15c).

4.2 Processing of Map

4.2.1 Map Generation

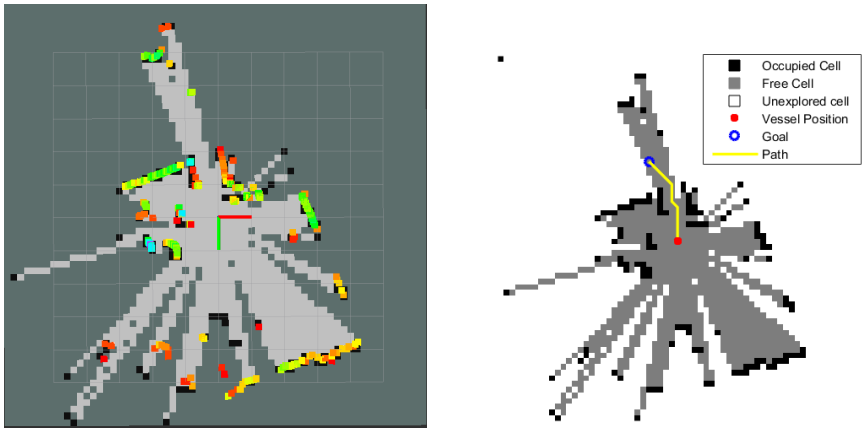
The applied Hector-SLAM package produces both an occupancy grid and the vessel's position within it. By adjusting parameters in the Hector-SLAM launch file, one may choose both arbitrary grid size and resolution of the occupancy grid. In general, larger grid sizes are more computationally expensive and demand more of the operator computer.

In the trials performed in this thesis, the resolution is set to either 0.1m or 0.2m, while the grid-size is set to either (128x128) or (256x256).

Once the Hector-SLAM package has generated the map, it is sent to the MATLAB node, where each cell in the occupancy grid is interpreted as either free, occupied or unexplored. This is performed by using the following threshold function:

$$\text{Cell status} = \begin{cases} \text{Explored and occupied,} & \text{for: cell value} > 50 \\ \text{Explored and free,} & \text{for: } 0 \leq \text{cell value} \leq 50 \\ \text{Unexplored,} & \text{for: cell value} = -1 \end{cases} \tag{4.17}$$

In Figure 4.6 the operator view of the map in the native ROS tool rviz is compared to the map as represented in MATLAB.



(a) *Hector-SLAM* (b) *MATLAB*

Figure 4.6: *Map Visualization*

4.2.2 Online Map Processing

In the implemented code, the map is processed online through four steps, before the path algorithms are applied. In this section, each of these steps is explained by the help of the example provided in Figure 4.7.

In addition to the four steps discussed in this section, Figure 4.7 visualizes the subsequent generation of a path by the path planner (which is explained in Section 4.3.2). Thus, the figure displays all the main steps the system applies during one iteration of path generation.

1. **Import Map from Hector-SLAM to MATLAB**
   The first step is to import the map to MATLAB using the ROS subscriber/publisher architecture. Here the threshold function described in Section 4.2.1 is introduced, and the data representation of the map is converted from vector to matrix format. The example map imported to MATLAB is provided in Figure 4.7a.

2. **Inflate Map**
   The vessel is represented by an x- and y- coordinate corresponding to its position in the occupancy grid. In order not to collide, all neighboring cells within the area that the vessel extends over to need to be free. In addition, cells within a safety distance around the vessel should be free, such that it has room to maneuver. The safety distance does not need to be particularly large, as the implemented path planner prefer paths that keep a distance to nearby objects (see Section 4.3.2.4).

   To ensure that the vessel has room to maneuver with some safety distance, all cells that have a distance less than a predefined inflation radius to an occupied cell are considered inaccessible and thus labelled as occupied. This process is called inflating the map, where the inflation radius is a parameter that can be adjusted by the operator, that by default is set to 0.4 m.

   The example map, after objects are inflated can be seen in Figure 4.7b.

3. **Reduce Map to Reachable Cells**
   Not all grid cells in the occupancy are in reality accessible by the vessel. There are two sources of this. The first is that the occupancy grid may have inconsistencies. This can be seen in the example map, where some rays are extending through identified obstacles in the upper left part of Figure 4.7a. The second reason is that the inflation process may close passages to cells that previously were within the reachable space of the vessel.

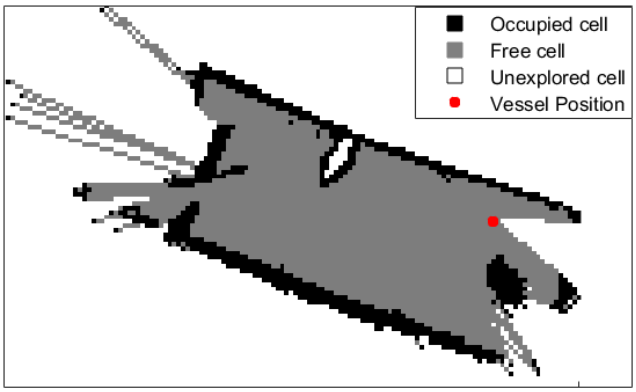
   The example map reduced to only reachable cells is shown in Figure 4.7c.

4. **Assume Non-Visible Cells Within a Distance of 3m as Explored and Free**
   As pointed out by Grabowski et al. (2003), due to specular reflection, emitted rays from the lidar that strike an adjacent object with a shallow angle, can be reflected away from the lidar. The effect of this is that the lidar does not always receive the reflected rays, even if they have hit an object. The lidar cannot decipher between this event (and other erroneous events) and the case where there are no obstacles within the maximum range of the lidar. For this reason, the Hector-SLAM algorithms do not update the map in directions where no reflected lidar rays are detected.

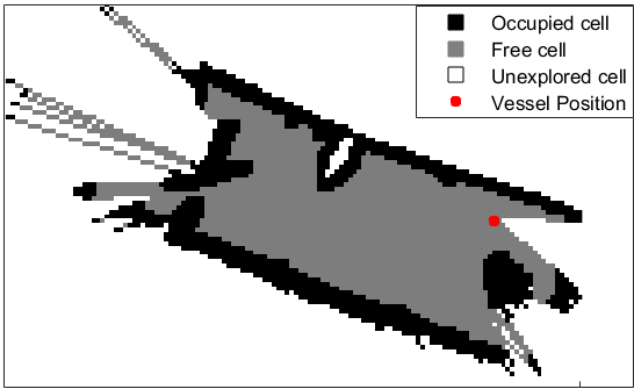
   In the experiments performed in the basin, this effect is evident in sections where the lidar scan extends towards the length direction of the basin. In Figure 4.7 the effect can be seen to the right of the indicated vessel position.

   Implemented to the MATLAB code is, therefore, a lidar emulator that temporary updates the map in the directions where no reflected rays have been detected. This update is performed by assigning cells within these sections as free out to a range of 3 meters.

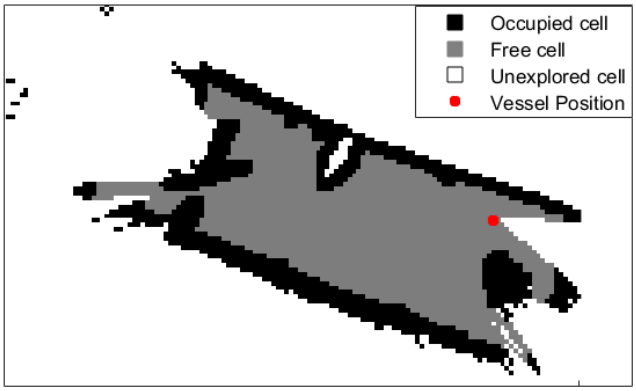
   If there are objects in the vessel's environment that the lidar has problems recognizing, the method is likely to increase the likelihood of collisions. In the controlled lab, however, the lidar is able to efficiently scan most obstacles, and the implementation did not cause any collisions. It does, however, make the vessel able to explore the basin more efficiently and faster. If the vessel is operating in environments that contain objects that are more difficult for the lidar to scan, it should be reconsidered if this particular operation should be performed. Figure 4.7d display how the described procedure alters the example map.



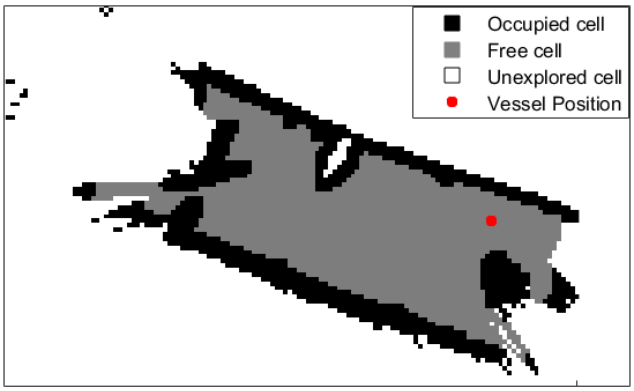
(a) *Map as imported from Hector-SLAM*



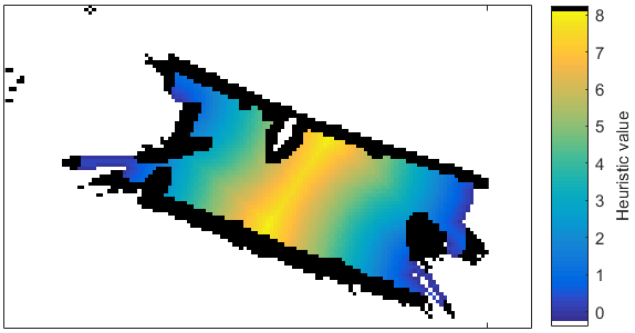
(b) *Inflated map*



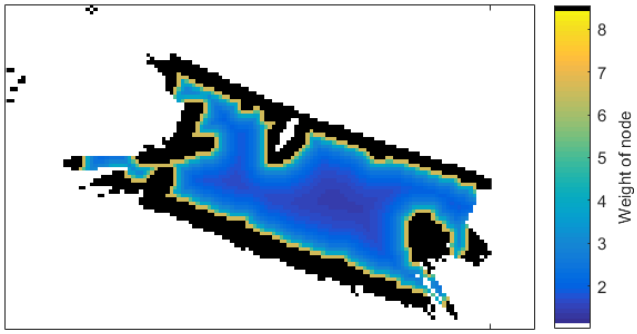
(c) *Map reduced to reachable cells*



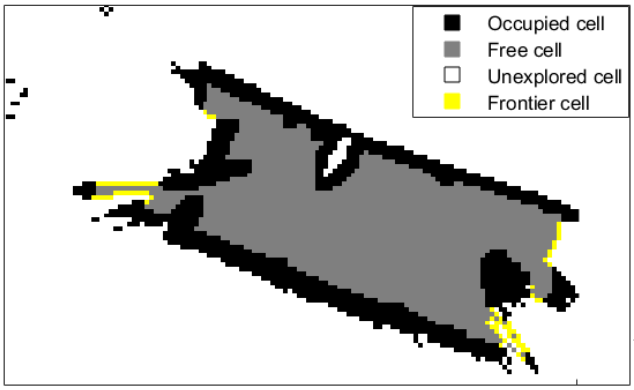
(d) *Non-visible cells within a distance of 3 m assumed explored and free*



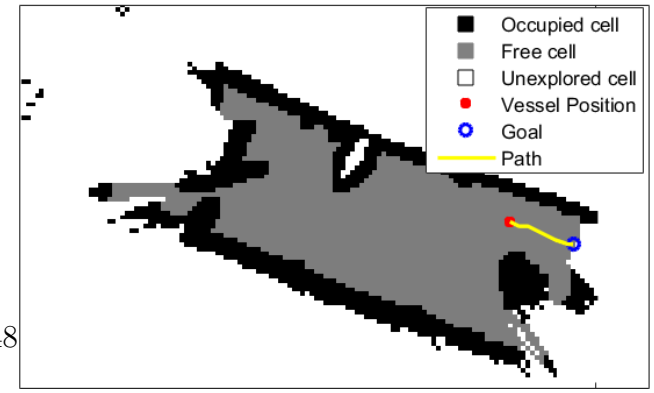
(e) *Heuristic value for pathfinder*



(f) *Weight of cells for pathfinder*



(g) *Frontiers indicated*



(h) *Chosen path*

Figure 4.7: *Map processing and path generation stages*

4.3 Guidance And Navigation

4.3.1 Exploration Strategies

This section investigates how the exploration strategies introduced in Section 1.2.5 is implemented to the system. The performance of both methods are further reviewed in subsequent chapters. Which exploration strategy the vessel shall use during operations can be altered online by a parameter shift.

4.3.1.1 Frontier-Based Exploration

The following steps summarize the implemented Frontier-based exploration algorithm:

Implemented Frontier-based exploration algorithm

1. Identify frontiers in the occupancy grid. In this step each explored cell that has an unexplored neighbour cell is considered a separate frontier
2. Plan path to the closest frontier
3. Allow for the vessel to start following the path and repeat the process

Figure 4.8 illustrate how the simulated vessel explores a scenario. In this example, a low resolution was used on a relatively small map. This was done to so that individual frontier-cells are easy to see and to perform the full exploration in relative few moves. As is evident from the figure, the vessel efficiently moves towards the closest frontiers until the whole map is explored, whence it returns to its initial position.

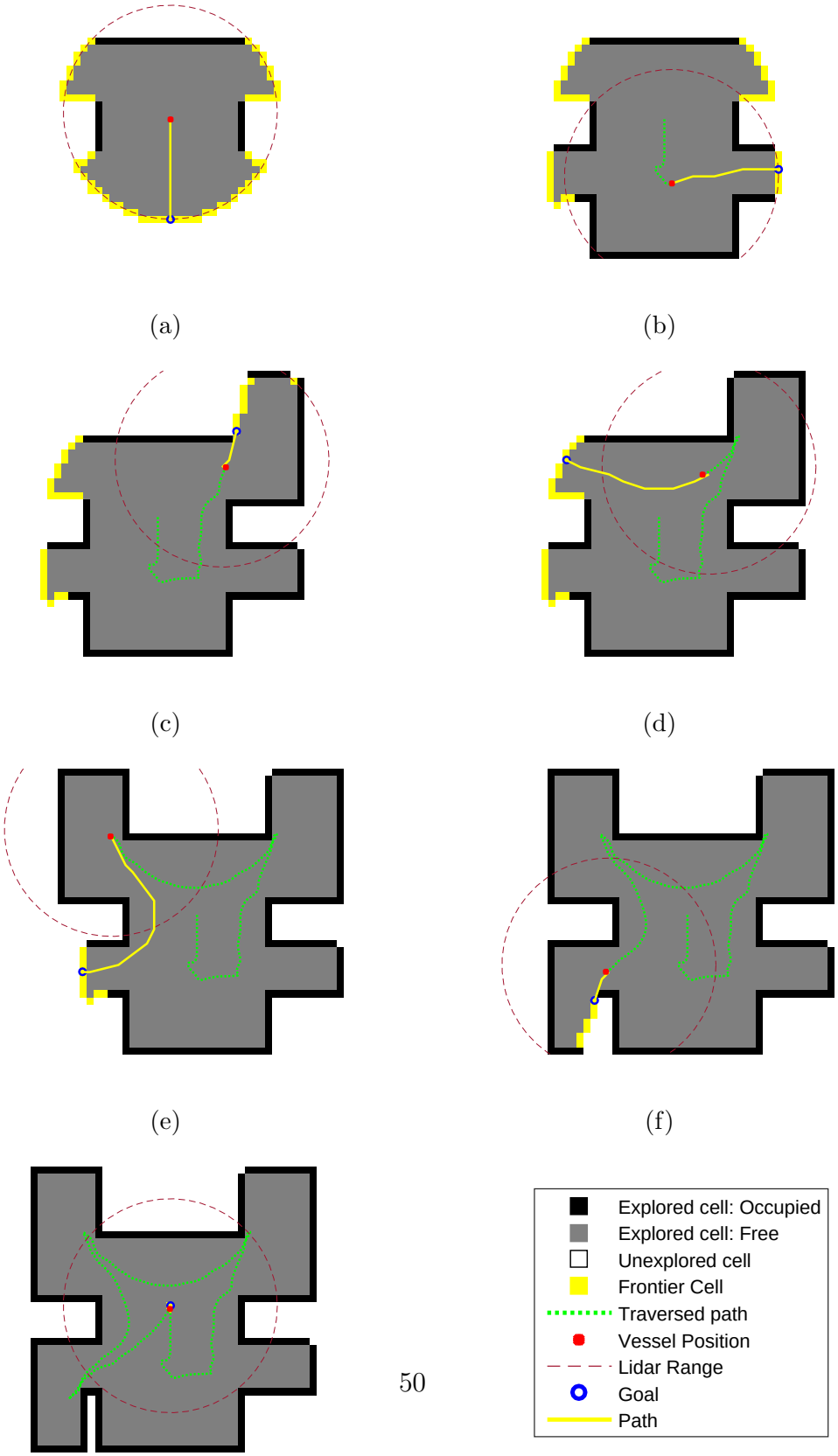


Figure 4.8: *Frontier-based simulated exploration stages*

4.3.1.2 Gap-Based Exploration

The exploration strategy explained in this section is better described as a one based on the identification of discontinuities in the lidars environment, than a one that tries to replicate the Gap navigation tree exploration strategy as it is described in Section 1.2.5.2.

One of the distinct ways that the implemented method differs from the referenced literature is that it does not utilize the Gap navigation tree, where each the gap is set into a structure with children and parents. As opposed to investigating gaps in an orderly manner according to the logic of the Gap navigation tree, the implemented method always navigates to the nearest unexplored gap. For this reason, the thesis refrains from classifying the implemented method as a Gap navigation tree exploration. Instead, the implemented method is referred to as Gap-based exploration.

The following steps summarize the implemented Gap-based exploration algorithm:

Gap-based exploration algorithm

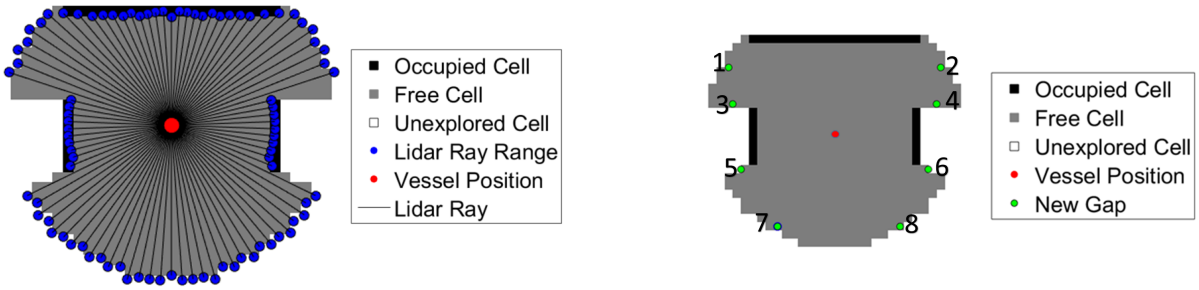
1. Identify new gaps based on the lidar scan.
2. Mark gaps that do not have any unexplored cells in their vicinity as explored
3. Plan path to the closest gap
4. Allow for the vessel to start following the path and repeat the process

In the implemented method, the identification and assessment of gaps are performed in each new iteration that the exploration node performs. As a result, a large number of gaps are quickly identified, many of which represents the same discontinuities in the map. Though this does not cause any immediate problems, it would be more convenient to work with fewer gaps. One possible development of the code may, therefore, be to group neighboring gaps together.

In the current implementation, due to the nonideal maps and limited lidar range, it cannot be guaranteed that the vessel will explore all accessible areas. However, since the identification of gaps are performed in each new iteration, a complete exploration it is deemed likely.

The implemented rules for maintaining and editing gaps are as follows:

- **Gap Identification**
  New gaps should be identified where there is a discontinuity in the map. This



(a) *Lidar rays emulated from lidar position* (b) *New gaps identified*

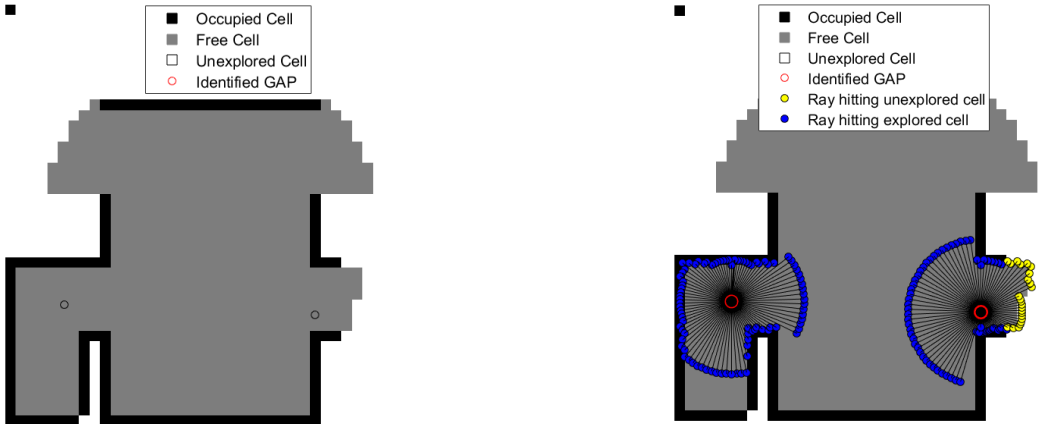
Figure 4.9: *Visualization of gap identification. (In the actual code the angular increment between rays is smaller)*

is performed by sending out rays from the vessel's position in the map, as can be seen in Figure 4.9a. Figure 4.9b display how eight new gaps are identified based on the emulated rays. The rules at which new gaps are placed are as follows:

- If the perceived length between two adjacent rays differs in length by more than a predefined distance, a gap is identified. The gap is now placed between the two endpoints. In Figure 4.9b the gaps 3-6 are identified in this manner.
- In sections where the rays do not hit any objects within its range, a gap is placed in the middle of the section. In Figure 4.9b the gaps 1, 2, 7 and 8 are identified based on this rule
- For a new gap to be placed it must have unexplored area its vicinity.

- **Marking Gaps as Explored**
  The gaps are marked as explored once there is no longer any unexplored areas in their vicinity. This operation is demonstrated in Figure 4.10. In this figure, the left gap is in the vicinity of only explored cells and can be marked as checked. The right gap is in the vicinity of unexplored cells and needs to be investigated further.

Figure 4.11 visually shows how the simulated Gap-based exploration strategy functions. The only difference between this simulation and the one illustrated in Figure 4.8 is that a different exploration strategy is applied. Comparing the two simulations it evident that both methods were able to explore the scenario efficiently and it is not evident from these examples which that performs best.



(a) *Two Gap candidates marked for investigation* (b) *Gaps are Examined*

Figure 4.10: Examination of Gaps

4.3.1.3 Condition for Updating Path

In both of the described exploration strategies the path is replanned each time the exploration node has performed an iteration. However, the computation time causes a delay, where the vessel's position is shifted at the end of an iteration from what it was at the start of it.

In order to avoid the situation where this delay causes the vessel to oscillate between goal positions in opposite directions the following logic has been implemented:

Condition for updating path logic
-If the cell corresponding to the goal position or previous gap is not yet explored or examined:

-Then, the produced path is only updated if the difference in orientation between r_{new} and r_{old} is less than 90 degrees, where r_{new} and r_{old} is the orientation of the first segment of the new and old path respectively,

4.3.1.4 Finishing Procedure

Unless otherwise specified, at the time that no more frontiers or gaps are accessible the vessel returns to its initial position where it will await further instruction from the operator.

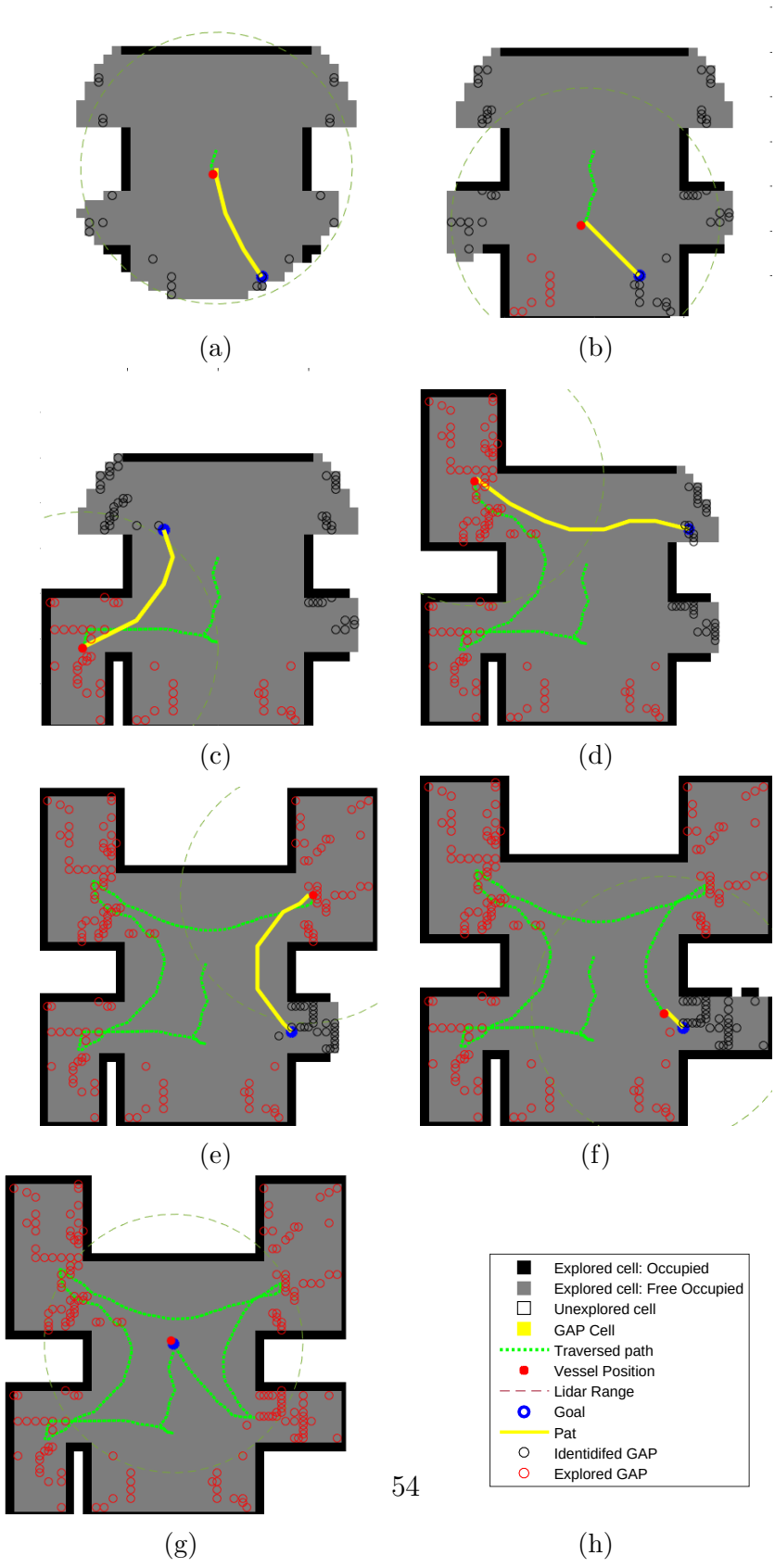


Figure 4.11: *Gap-based simulated exploration stages*

4.3.2 Path Planner

The A* search algorithm is an algorithm that is relatively simple to implement, recognized as efficient, and one that enjoys great popularity within the robotics community. For these reasons it this is the algorithm that is utilized for planning paths between locations in the area where the vessel operates. When applying the algorithm to the occupancy grid used in this thesis, the center of each grid cell is considered as a separate node that the path can cross.

The A* search algorithm applies two sets of lists for keeping track of data during a search. These lists are the Open list that keeps track of nodes that are candidates for further examination, and the Closed list that keeps track of examined nodes. Both the Open and Closed list contain information about the cost of reaching a node and its preceding node.

When the algorithm chooses a new node for examination, it selects the node in the Open list that appears to be the most promising, which is equivalent with the node that has the lowest f-value. The f-value is defined as follows:

$$f(s) = g(s) + h(s) \tag{4.18}$$

where $g(s)$ is the cost the pathfinder has used to reach Open node s from the start, and $h(s)$ is the Heuristic cost, which is an estimate of the cost to reach the goal from node s.

In grid-based graphs that represent the geometry of the map graphs, it is suitable to set the Heuristic value equal to the Euclidean distance from start to the goal node. Note that this never over-predicts the actual cost of reaching the goal from a particular node. This is important since the A* search algorithm is guaranteed to find the shortest path in the graph if the Heuristic does not over-predict the cost of getting to the goal. If all h-values are set to zero, the method reduces to that of the Dijkstra's algorithm, which is a special case of the A* search algorithm.

Figure 4.12 illustrates how the Heuristic value is automatically generated for a given the goal, which is marked red. As many other figures this chapter it has been generated by using data from the trials in the basin. The resulting values of the Heuristic can in this figure, thus be directly be related to the distances in the basin.

The A* search algorithm is complete, meaning that if there exists a path to the goal node it will find it. The method is further admissible (finds a best path) if the Heuristic does not over-predict the actual minimum cost of reaching the goal. Since the method solves the problem by investigating the most promising nodes first, it is classified as a best-first-search algorithm.

The following operations describe how the algorithm find a the path, step by step:

A* Search Algorithm

1. Pop the node in the Open list that has the lowest f-value. This node now becomes a part of the Closed list.
2. If the node is a destination node, retrace the path backwards to find the path from start to goal.
3. Examine the neighbouring nodes that are not Closed.
4. If the node has not yet received a g-value, or the new g-value is lower than the old, then update it. When updating a g-value, record the parent node. Update at the same time the f-values.
5. Add the neighbouring and non-blocked nodes to the Open list. Add the examined node to the Closed list. Repeat.

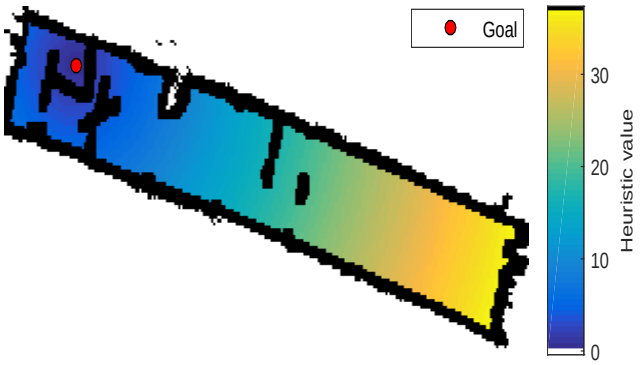
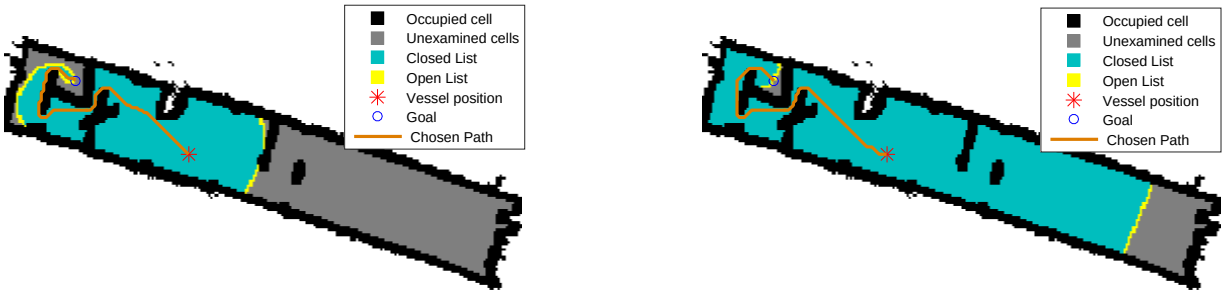


Figure 4.12: *Heuristic value of nodes in example map.*

Figure 4.13a illustrates which nodes that have been investigated to find the shortest path from start to goal in a given map by the use of the A* search algorithm. For comparison, the same result using the Dijkstra search algorithm (where the Heuristic is set to zero) is included in Figure 4.13b. Although the resulting path differs, the length of the generated paths are identical, and thus both methods have found a shortest path. Due to the best-first strategy of the A* search algorithm, it opened much fewer nodes to perform the operation.



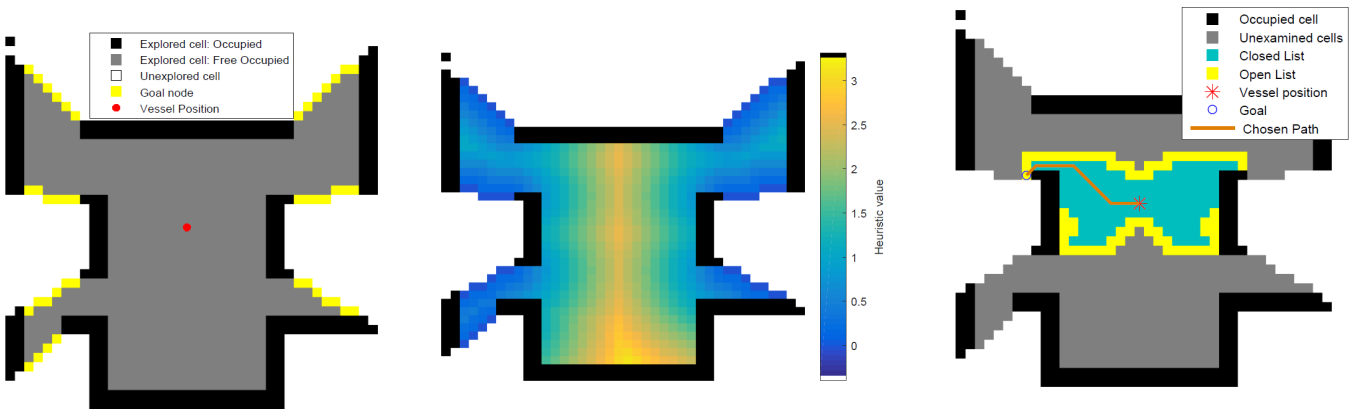
(a) *A* search algorithm* (b) *Dijkstra search algorithm*

Figure 4.13: *Path planer comparison: open and closed nodes*

4.3.2.1 Multiple Goal Nodes

The exploration methods applied can yield multiple goal candidates, which the search algorithm should be able to handle. The pathfinder should therefore find the shortest path to any of the potential goal candidates. In the implemented algorithm this is realized by adjusting the Heuristic $h(s)$ such that it equals the Euclidean distance from cell s to the nearest goal node. The first goal candidate the method opens during a search is now identified as the closest goal node, from where the path is retraced back to start.

Figures 4.14a and 4.14b illustrate how the Heuristic is generated with multiple goal nodes, while Figure 4.14c illustrates how nodes subsequently are opened to find the shortest path to a goal node.



(a) *Goal nodes indicated*

(b) *Heuristic value of nodes*

(c) *Node status when path is found*

Figure 4.14: *A* search algorithm applied to multiple goal nodes*

4.3.2.2 Node Connection Distance

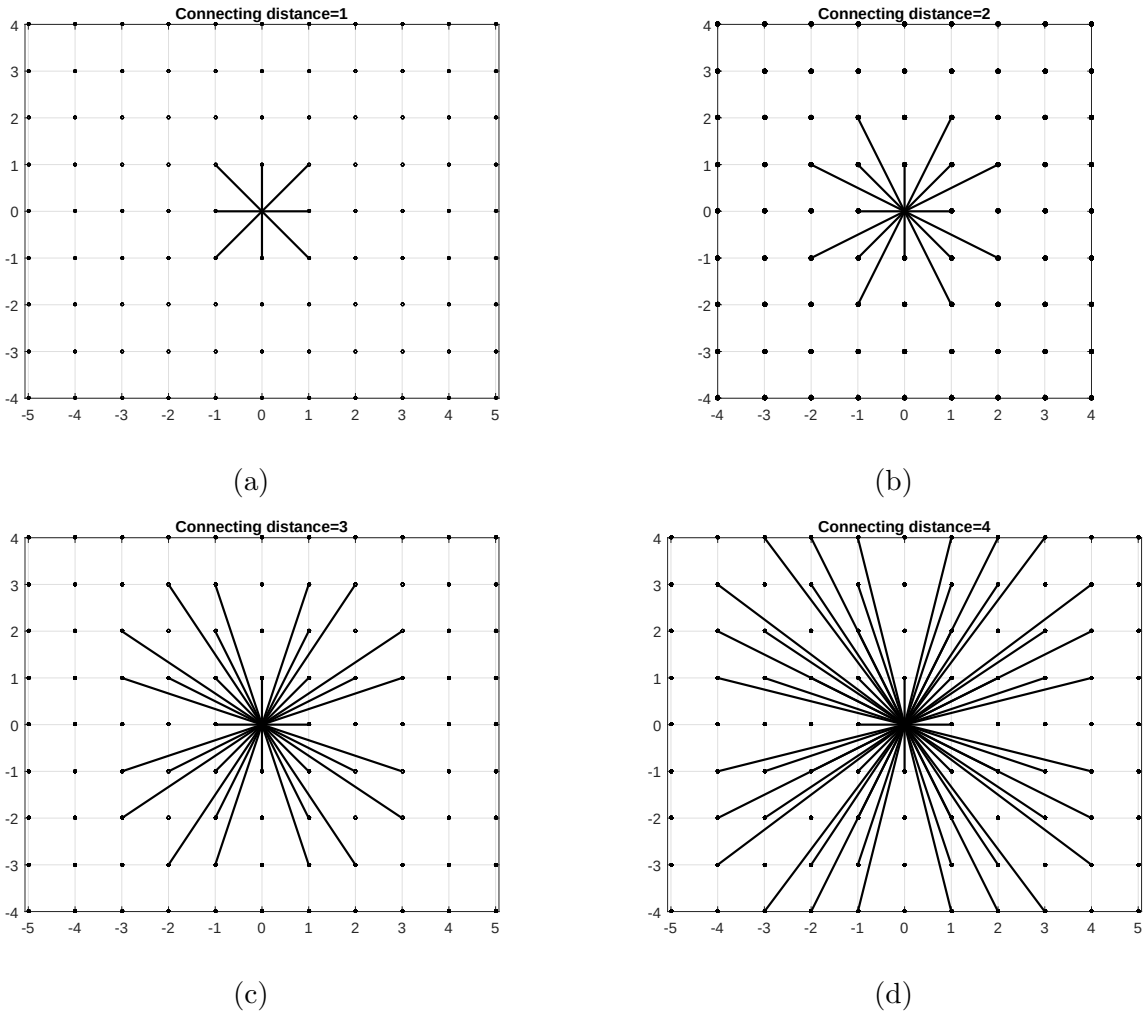
In the standard A* search in occupancy grids, only the eight neighboring tiles are investigated when expanding paths from a node. This restricts the orientation of the planned path to eight directions, in increments of 45 degrees, which again leads to suboptimal paths.

A way to circumvent the restriction is to allow each node to connect to nodes that are more than one tile away. Figure 4.15 illustrate which nodes that are investigated when expanding the path, for connecting-distances between 1 and 4. Each connecting line in this figure represents a possible heading that the final path can take. As is evident from the figure, increasing the connecting-distance quickly increases the number of possible directions. A connecting-distance of two yields 16 possible orientations, while three and four yields 32 and 54 possible orientations respectively.

Figure 4.16 illustrate how the calculated paths differ when using a connecting-distance of one and four. As expected, the calculated path using a connecting-distance of four is both shorter and smoother than when the connecting-distance is one.

While larger connecting-distances increase the number of possible orientations, and in general lead to better and shorter paths, it quickly increases the complexity of calculations and thus also computation times. In the implemented system, the connecting-distance is an integer parameter that may be changed by the operator

during operations. The default value for the parameter is set to four, which is a value found to yield suitable paths, within reasonable computation time.



(a) (b) (c) (d)

Figure 4.15: *Node connections using different connecting-distances*

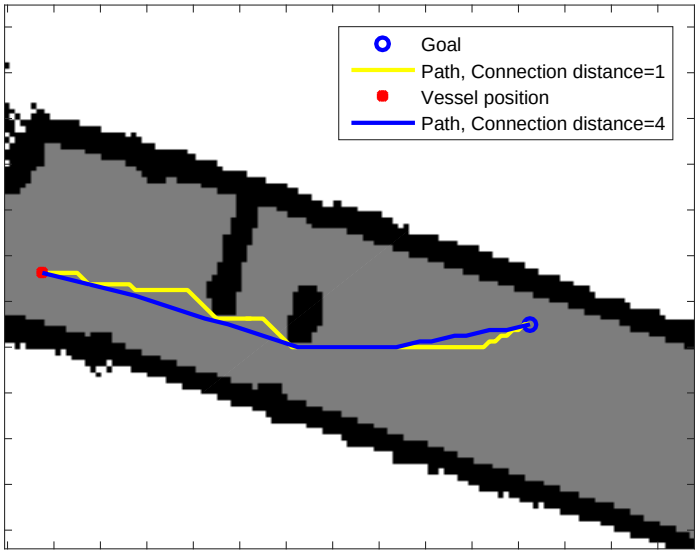


Figure 4.16: *Comparison of path found using a connecting-distance of one and of four.*

4.3.2.3 Focussing Exploration Toward Unexplored Destination

In some cases, it is preferable to focus the exploration towards a destination outside of the explored area. This is not necessarily the same as desiring to reach a point as fast as possible, as it also keeps some of the rapid exploration properties.

If the only goal was to reach the unexplored point as quickly as possible, a more direct method would be to apply the A* search algorithm to the full map, not considering whether cells are explored or unexplored. Then whichever frontier the resulting path is passing through, is the goal which the vessel should maneuver to.

The implemented method considers both the cost of reaching a frontier and the Euclidean distance from the chosen frontier to the set exploration goal. No cells outside of the explored map are assessed. This is implemented by adding a cost of reaching a frontier-cell that relates to the Euclidean distance between the frontier and the goal cell, denoted g'(s_f).

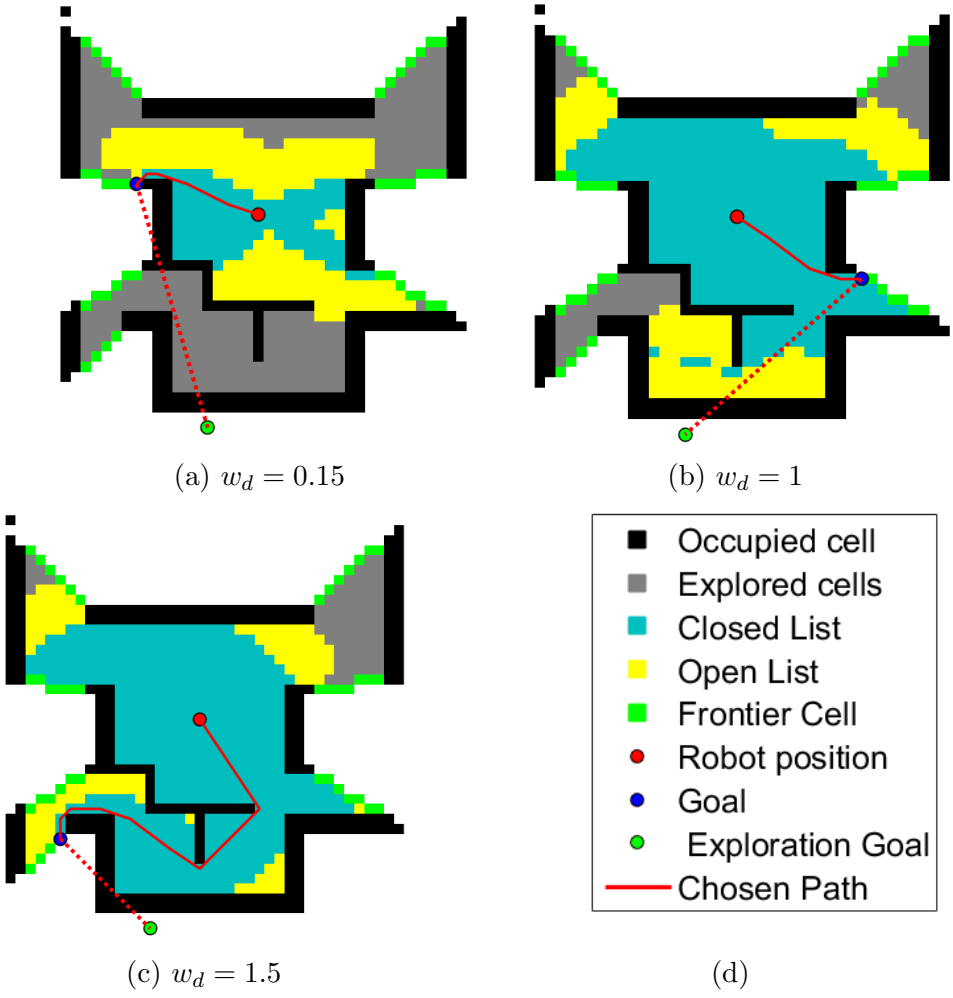
The weight on the distance between frontier and exploration goal, w_d is now introduced, which regulates how to weight the importance of the cost g'(s_f). A large value of w_d means that it is important that the chosen frontier is close to the

exploration goal, while a low value, means that it is more important that the path from start to the particular frontier is short.

The extra cost g'(s_f) of travelling to the frontier-cell s_f is now given as:

$$g'(s_f) = w_d \, d_{goal}(s_f) \tag{4.19}$$

Where d_{goal} is the Euclidian distance from the frontier-cell to the exploration goal.



(a) $w_d = 0.15$ (b) $w_d = 1$

(c) $w_d = 1.5$ (d)

Figure 4.17: *Chosen path to global exploration goal using different weigths, w_d*

Now that the cost of reaching the goal has changed, so should the Heuristic. This is performed by setting the Heuristic of each cell equal to the lowest value found when combining the cost traveling to a frontier with the extra cost of reaching that frontier. The Heuristic value of node s can thus be expressed as follow:

$$h(s) = \min(d_{s_f} + g'(s_{\mathrm{f}})), \quad \text{For all frontier-cells } s_f \tag{4.20}$$

where s is the investigated cell, s_f represent a frontier-cells, $g'(s_{\mathrm{f}})$ is the extra cost of reaching the frontier-cell, and d_{s_f} is the Euclidean distance from the vessel's position to the considered frontier-cell.

Figure 4.17 displays how the implemented A* search algorithm performs using different weights on w_d. The value of w_d needs to be tuned according to the operator's preferences. Testing performed in this thesis revealed that a value of w_d between one and two seems reasonable.

The operator can set a global exploration goal during operations by interacting with the map window. This is further reviewed in Section 4.4.

4.3.2.4 Weighting of the Cost-map

Thus far, the cost of passing from one node to another using the A* search algorithm has been assumed identical to the Euclidian distance between the two nodes. In the following section, a weighting of the nodes based on their distance to nearby objects is introduced.

The weighting is implemented such that the closer a cell is to an occupied cell, the higher the cost of passing it. There are several reasons why this is convenient, one of which is that the vessel now maneuvers further away from objects, which decrease the chance of collision. Also, due to the lidar being a further away from walls, it can scan a wider portion of the walls. Finally, since the path now does not take the shortest route around corners, the resulting paths are smoother and contains less sharp turns.

The weight $w(s)$ on node s is set according to the following formula:

$$\mathrm{w(s)} = 1 + \frac{5}{0.1 + \mathrm{d_{obj}}} \tag{4.21}$$

where $\mathrm{d_{obj}}$ is the Euclidian distance from the node to the closest occupied cell. Note how the weight is always larger than 1, which ensures that the Heuristic value still never overestimate the true cost of moving to the goal, and thus still is admissible. Figure 4.18 illustrates the weight of the cells in the occupancy grid using this formula for a scenario in the basin.

A simple way of implementing the weight of the cost of travelling between two nodes that demands few operations is to only account for the weight in the two connected nodes. With this approach, the cost of moving between the two directly connected nodes A and B is given by the following formula

$$g_{AB} = \frac{w(B)+w(A)}{2} d_{AB} \tag{4.22}$$

where w(A), w(B), d_{AB} are the weight of the two nodes and the distance between them respectively.

As can be seen from the formula, only the first and the last node are assessed to calculate the cost of moving from one node to another. When the connecting-distance is larger than one, this means that some of the cells that are passed are not evaluated. For this reason, an approach that accounts for the cost of each node in the connection has been implemented. In this approach the cost of moving from node A to B is as follows:

$$g_{AB} = \sum_{s=1}^{k} d(s)w(s) \tag{4.23}$$

where d is the distance the path traverse over node s, and k is the number of nodes that the connection passes. The formula is exemplified for a connection passing six cells in Figure 4.19

In the implemented code, the length segments that the path covers in a connection are automatically generated each time a new connection is assessed, which is quite computationally expensive. To reduce the computational demands, it would be more advantageous apply a pre-generated table that contains these distances as a function of vertical and horizontal movement in the connection.

Both of the two mentioned approaches for calculating the cost in the weighted map have been implemented to the system, where the method of only evaluating the first and last node is significantly faster. The two methods of calculating cost can be toggled online.

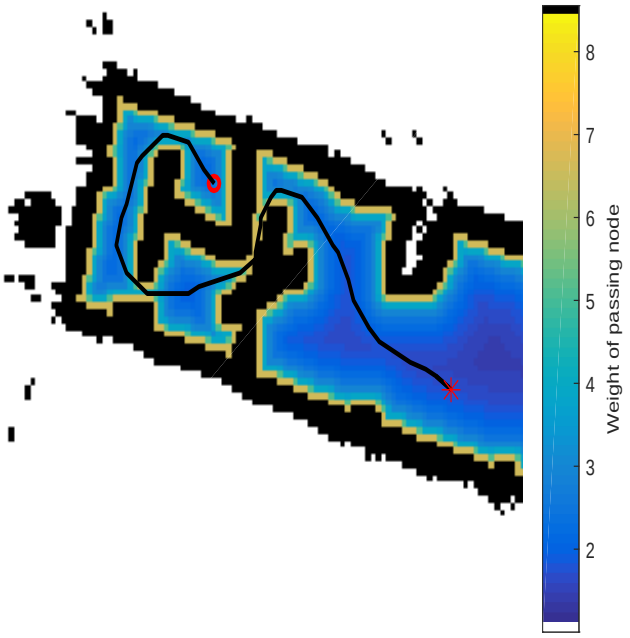


Figure 4.18: *Planned path in the weighted grid.*

Figure 4.20 compares how the generated path differs using the two methods of incorporating the weight. A path is calculated both using a connecting-distance of four and ten, and for comparison, the calculated path using no weighting is included. For a connecting-distance of ten it is clear that the resulting path when weighing only the first and last cell in a connection is closer to the wall than desired. However, based on these figures (and other testing performed by the author) it appears that the simple method of calculating weights are sufficient for a connecting-distance of four. If computation speed is no issue, it is still advised to use the method that accounts for all weights along a connection.

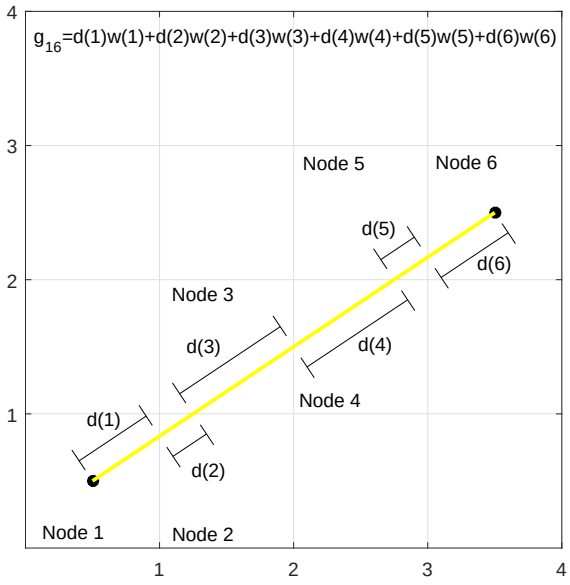
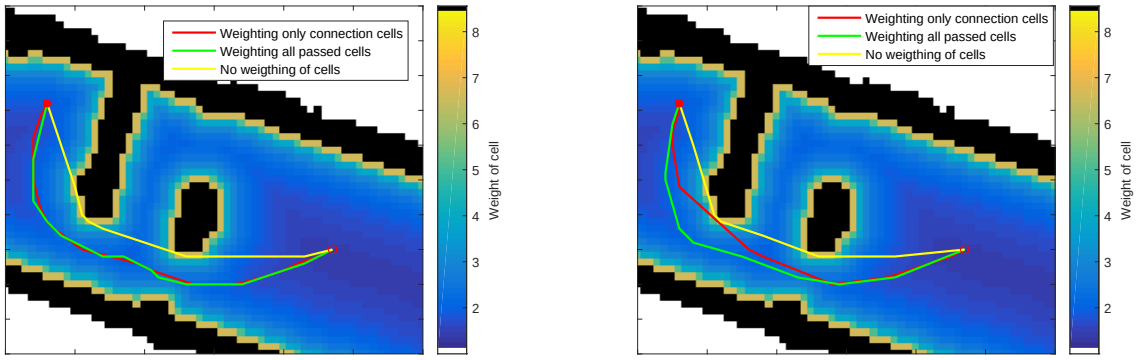


Figure 4.19: *Cost of moving from one node to another in the weighted costmap. d(s) and w(s) represent the distance and weight on each node respectively.*



(a) *Connecting-distance 4* (b) *Connecting-distance 10*

Figure 4.20: *Planned path using different weighting strategies*

.

4.3.2.5 Reuse of Data Between Iterations

In the implemented path planner, the path is recalculated from scratch in each iteration. It would be more advantageous if the method was able to reuse the information obtained in previous iterations, which more sophisticated algorithms indeed can do.

One algorithm that can perform this is the D* search algorithm, which is an extension of the A* search algorithm that is able to reuse information it has obtained in previous iterations. In this method, only the nodes that are affected by changes in the map or pose of the vessel, and their descendants are reassessed in each new iteration. Since only the map in the vicinity of the vessel thus needs to be updated, it is convenient to start from the goal node and calculate the backward when applying the D* method. In this way as few nodes as possible need to be updated.

When only a relatively small area around the vessel's position needs to be updated, the D* method is recognized being much faster than the A* search algorithm (Stentz et al., 1995). In the current system, however, several nodes are set as possible goal nodes, which somewhat complicates the use of the D* algorithm. The idea of reusing information from previous iterations should still be possible to apply, for instance by locking onto goal nodes once they are identified.

The idea of reusing information from previous iterations also applies to the generation of the weighted map, and the Heuristic, which in the implemented system do not reuse information obtained in prior iterations.

4.3.2.6 Video Demonstration of Pathplanner

The author has created a video demonstrating the search algorithms discussed in this section. The video is appended in the electronic attachment. Besides, it is available online (Ueland, 2016b)

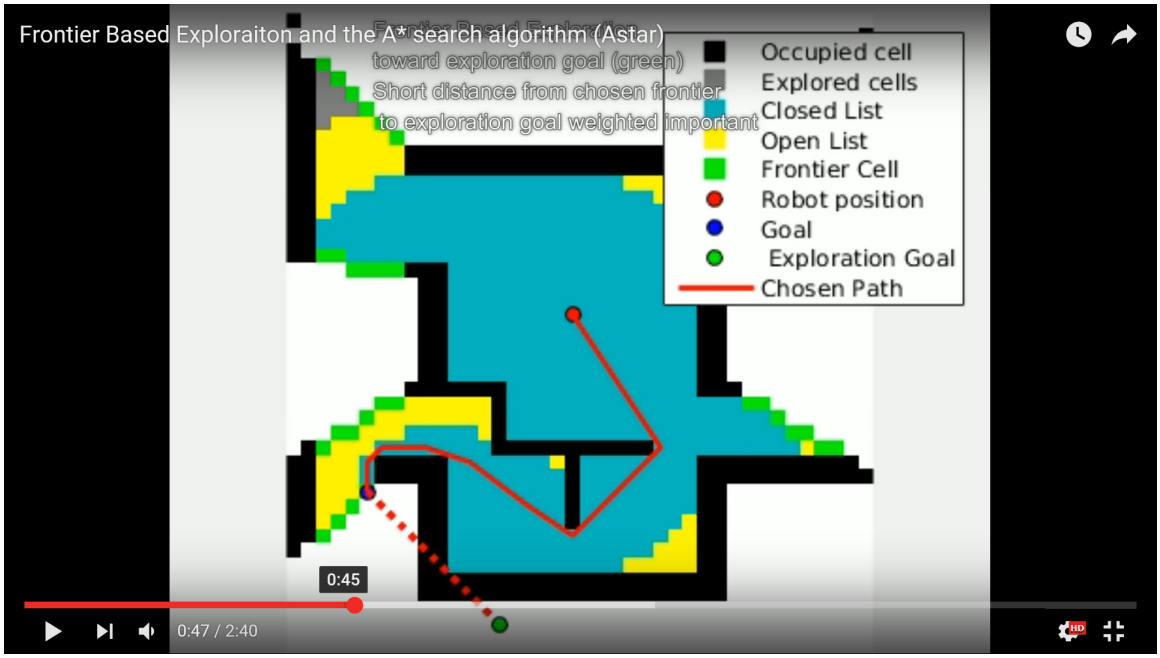


Figure 4.21: *Screen shot of video showing implemented path planner strategies.*

4.3.3 Velocity Control Law

The velocity of the vessel is controlled by adjusting the setpoint that the motion controller receives. This procedure is in this thesis labeled as a velocity control law and is discussed in this section.

4.3.3.1 Setpoint Generation

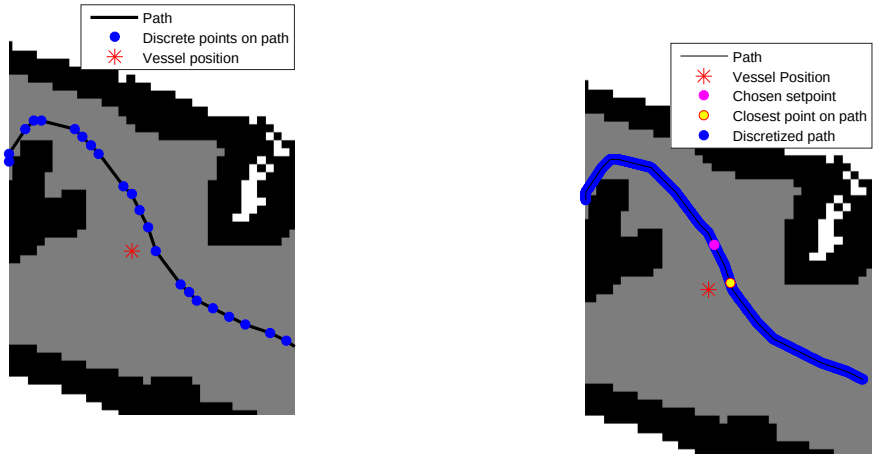
A list consisting of the first 128 discrete positions in the planned path are after each iteration sent from the exploration node to the ROS-framework. Based on this list, a separate ROS node generates set points for the controller. The ROS node that performs this operation is running independently on the exploration node, which allows the commanded setpoints to be updated with a frequency of 20 Hz, even if the path planner only updates the chosen path every few second. The implemented node (Path2Setpoint-node) designated for this procedure performs the following three operations:

1. **Rediscretizing Path**
   The imported vector that represents the optimal path has a relatively large distance between each point. Also, the distance between each discrete point is not uniform. The optimal path is, therefore, rediscretized to a new step-size, which by default is set to 0.01m. Figure 4.22a and 4.22b illustrate how the discrete vector representing the path is represented before and after this operation. Note that in the rediscretized version the points are so close that they appear to constitute a solid, thick line.

2. **Identify Closest Point on Chosen Path**
   Due to the dynamics of the vessel, one cannot expect its position to be aligned with that of the planned path. The discrete point on the path that is closest to the vessel, therefore, needs to be identified. This operation is performed by comparing the current position of the vessel with the re-discretized path. In Figure 4.22b the closest point is identified as indicated by a yellow point.

3. **Find Setpoint on the Chosen Path**
   The point identified as the closest to the vessel's position is now used as a reference point from where to calculate the new setpoint. The setpoint is found by iterating n discrete points forward in the path from the identified closest position. Where,

$$n = \text{round}\Big(\frac{\text{Setpoint distance}}{\text{Stepsize}}\Big). \tag{4.24}$$



(a) *Descretized path as generated by the exploration and pathfinder node*

(b) *Rediscretized path. Closest path point and chosen setpoint indicated*

Figure 4.22: *Generation of setpoint*

If there are less than n points left in the path vector, the last point in the vector is chosen.

For the example displayed in Figure 4.22b the stepsize is 0.01 and the setpoint distance is 1m. As a result, the indicated setpoint is the 100th consecutive point after the yellow one.

4.3.3.2 Adapting Setpoint Distance

The setpoint distance is automatically adjusted based on the distance from the vessel's position to the nearest object. The nearer the objects, the lower value of the parameter. This scheme has several advantageous effects on the vessel's behavior. Firstly it ensures that the proportional term of the PD controller, which is directly related to the vessel's distance to the setpoint is low when the vessel is near objects, while it is high when there are no nearby objects. At the same time, the derivative term is not directly affected by the change of setpoint. This means that the vessel is able to adapt its speed quickly as the setpoint distance changes.

Another advantageous effect is that when there are no nearby objects, the vessel does not need to aligning itself on the path. Instead, it aims at a position further ahead in the path which results in both smoother and shorter trajectories.

The distance to the nearest object is extracted directly from the lidar data-cloud in a separate node (*Scan2SetPointDist-node*) and not from the generated map. This means that objects that have not yet been incorporated into the map, which is typically the case for dynamical objects, are taken into consideration. The effect is a decreased probability of collision.

Figure 4.23 shows how the node changes the setpoint distance according to distance to nearby objects. This mapping is tuned according to the capabilities of the vessel. As evident the function is truncated at a distance to objects of 0.1 m and 3 m. Mathematically the chosen relationship can be expressed by the following formula:

$$\text{Setpoint distance} = \begin{cases} 0.1 & \text{,for } d_{min} \leq 0.3 \\ d_{min} - 0.2 & \text{,for } 0.3 < d_{min} < 1 \\ 2d_{min} - 1.2 & \text{,for } 1 \leq d_{min} < 2.1 \\ 3 & \text{,for } d_{min} \geq 3 \end{cases} \tag{4.25}$$

where d_{min} is the shortest distance that to an object, as recognized by the lidar.

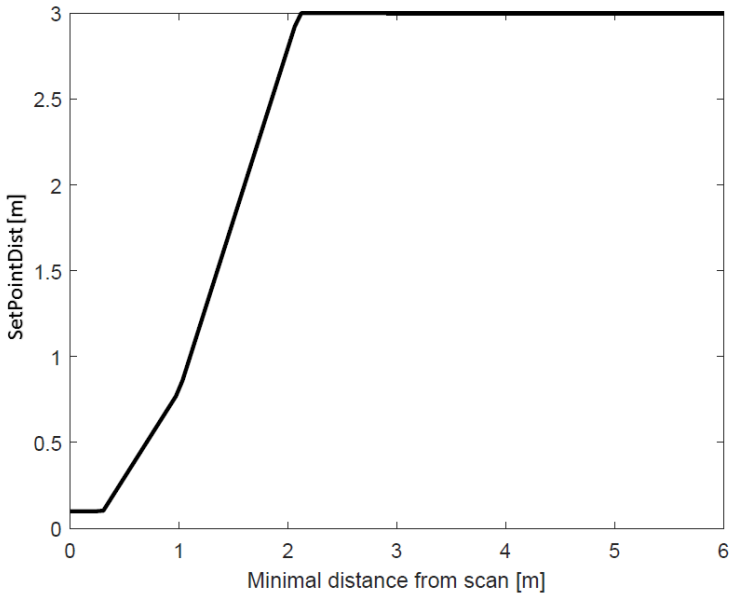


Figure 4.23: *The setpoint distance, as a function of the minimal distance registered by lidar.*

4.4 Operator Interaction

During operations, the operator has several means of interfacing with the system, as is explained in this section.

4.4.1 Interactive Map Window

Once the exploration node is run in MATLAB, an interactive window automatically appears. This window contains a figure that visually displays the map, vessel position, chosen path and path destination. During operation, the window is updated each time the exploration node has performed one iteration.

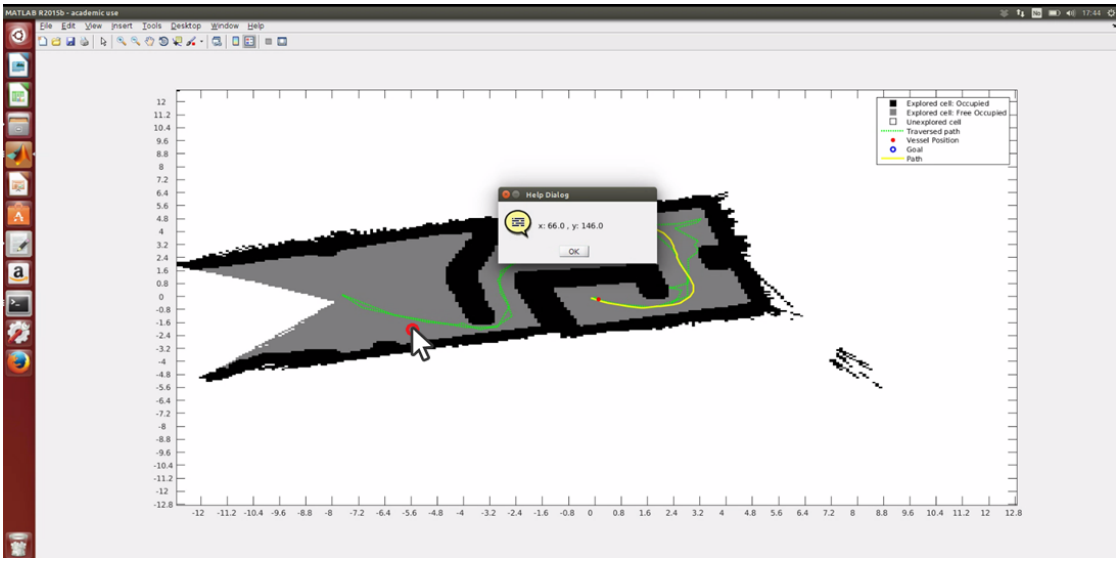
The user can interact with this window and by left-clicking, the following exploration modes can be chosen:

- **Focussing Exploration Toward Unexplored Destination**
  If the operator clicks on a position in the unexplored map, the vessel focus exploration towards a global setpoint (as described in Section 4.3.2.3).

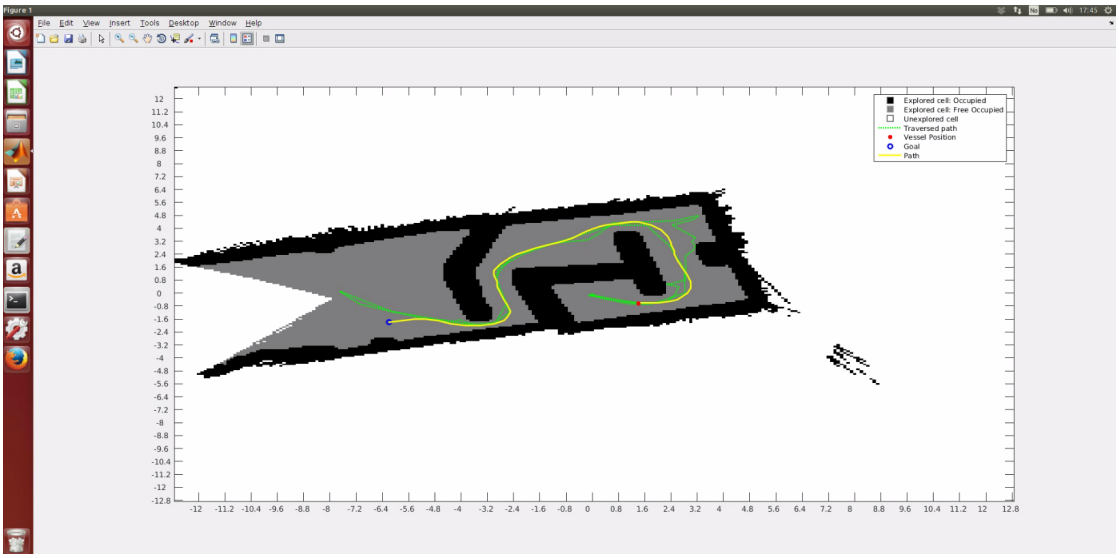
- **Autonomous Exploration**
  If the operator clicks on the top left corner of the map, the vessel returns to autonomously explore the basin according to the chosen exploration strategy.

- **Navigate to Known Location**
  If the operator clicks on an already explored position, the system computes a path to this location, and subsequently navigates to it.

Figure 4.24 displays the user interface on the operator computer. In Figure 4.24a, the operator has clicked on an explored position, while Figure 4.24b displays how the system responds by planning a path to the chosen position.



(a) *Operator has clicked on a known location on the map. X and y refer to cell placement in the grid of the identified click*



(b) *The path-planner has found a path to the desired position*

Figure 4.24: *Example of user interaction in the map.*

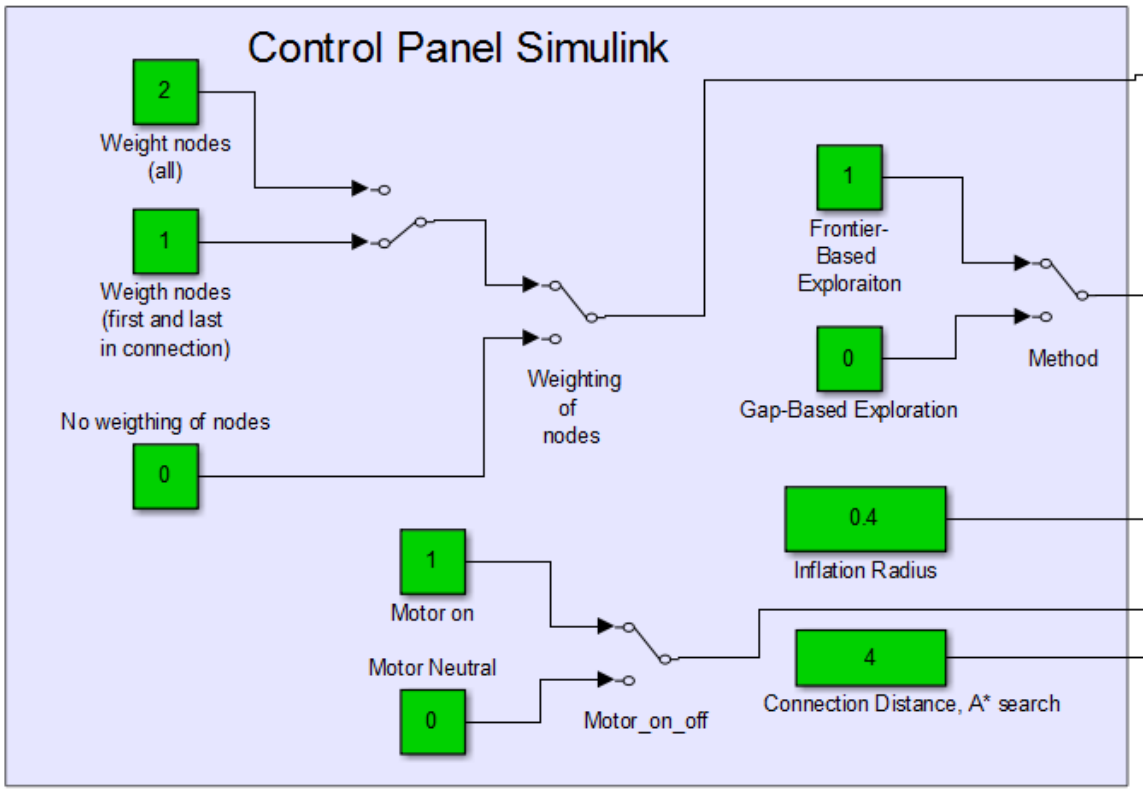


Figure 4.25: *Simulink-Control Panel, Extracted from Exploration Node.*

4.4.2 Parameter Setting

The exploration node running in Simulink has several parameters that may be adjusted by the operator. The setting of these parameters can be performed online via the Simulink control panel of the exploration node, as seen in Figure 4.25.

An overview over all parameters that are adjustable during operations and their function can be found in Table D.2.

Table 4.1: *Parameters adjustable by operator during operations*

Parameter	Explanation	Valid Range
Exploration Method	Switching strategy for exploration. (See Section 4.3.1)	0=Gap-Based 1=Frontier-Based
Connecting-Distance	Max connecting-distance between neighbouring nodes in the A* search algorithm. (See Section 4.3.2.2)	Integers (Should be kept low for reasonable computation times)
Weighting of nodes	Weighting the distance to the closest object as part of the cost of visiting each node. (See Section 4.3.2.4)	0=No Weighting 1=Weight first and last node in connection 2=Weight all nodes in connection
Exploration mode	Autonomous guidance or control to a specific point on the map (See Section 4.4.1)	Clicking in the map interface
Inflation Radius	Radius around a grid-cell that need to be free for the cell to be deemed accessible (See Section 4.2.2)	Reasonably set a little larger then the extent of the vessel
Controller gains	Tuning of motion controller gains. Can only be tuned in the Simulink version of the motion control node	Tune to suit vessel characteristics
Motor_on_off	Switching thruster power on or off. Used if the C++ compiled version of the MotionControl node is used	0=Off 1=Neutral Position

4.5 Software Architecture, Overview

In the implemented ROS architecture there is a number of nodes that perform different tasks. In addition to the nodes reviewed in this chapter, these nodes include a lidar driver, Hector-SLAM nodes, and an Arduino node.

An overview the network of nodes and topics in the ROS architecture can be seen in Figure 4.26. This figure also displays the physical units each node is run on. Only the topics that are vital for the control of the vessel are included in the figure. Thus, topics that are responsible for parameter-setting or monitoring the system are not included. A full overview of all nodes and topics present in the system, and their function can be found in Appendix D.

It should be noted that during most of the experiments it was chosen to run the MotionController node in Simulink on a second operator computer instead of using the C++ compiled version. This was done in order to easily make adjustments in the code, adjust controller gains and for toggling individual motors on and off.

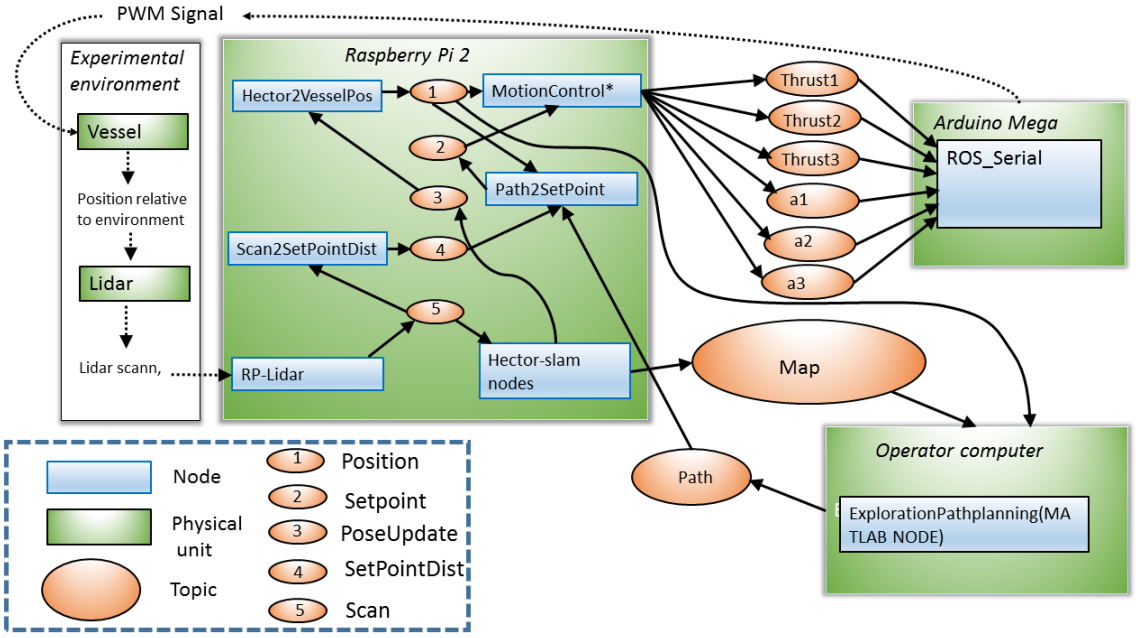


Figure 4.26: *Node/topic interactions in the implemented system.*

Chapter 5

Simulations

5.1 Simulated Nodes

The architecture of ROS makes it relatively easy to replace real hardware with simulated nodes, that interface with the system through the same type of messages that the physical device it replaces would. In the implemented model, two simulator nodes are applied. One node is the mapping simulator node (MappingSimulator-node), where the lidar and subsequent map generation process is simulated. The other node is the vessel simulator node (VesselSimulator-node) that simulates the vessels dynamics.

Figure 5.1 schematically shows the simulated system in the ROS subscriber/publisher architecture. Comparing this system to non-simulated architecture seen in Figure 4.26, gives an overview of what parts of the system that is simulated.

In the Figure 5.1, the red and blue boxes represent nodes and topics that are run in the same manner in the simulations as they are in real experiments. Using many of the same software components makes the simulations more similar to the physical tests and makes it easier to filter out bugs in the overall system. Besides it means that the user interface on the operator computer is identical in both cases.

5.1.1 Mapping Simulator Node

The mapping simulator node is responsible for simulating the map generation process. It contains logic for simulating the lidar, together with a structure for storing and updating a map of the vessel's environment. This means that both the

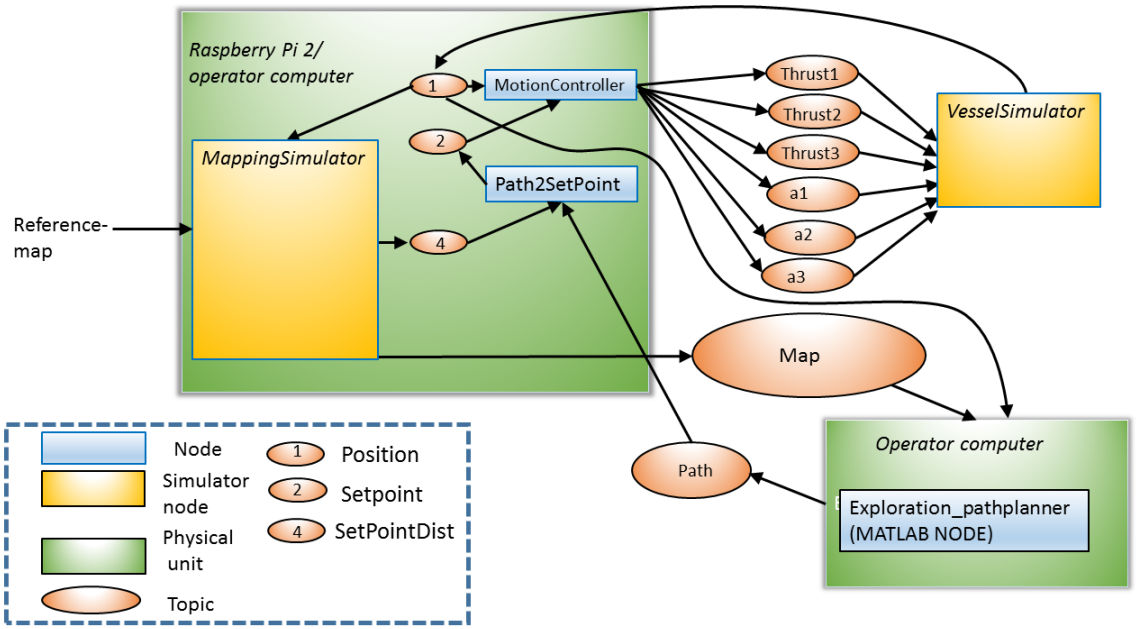


Figure 5.1: *Node/topic interactions during simulations*

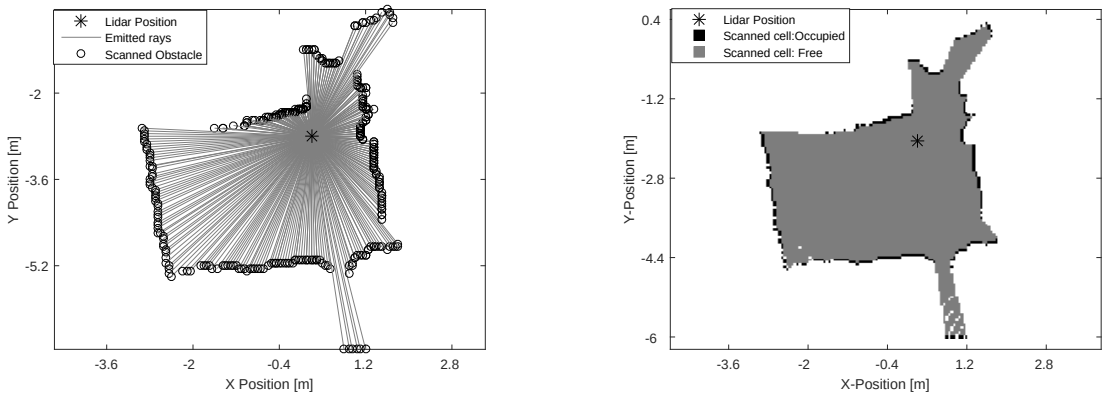
lidar node (RPlidar-node), and the subsequent mapping performed by the Hector-SLAM nodes are simulated through this node. In addition, the node simulates the logic for finding setpoints distances (Scan2SetPointDist-node).

The node applies two mapping arrays during simulations, both of which represents occupancy grids. One is the constant pre-generated Reference-map that represents the environment that the vessel operates in. The other is the dynamic Explored-map array that represents how much of the Reference-map that the vessel has explored.

The node has the position and heading of the vessel as input, and is implemented with a separate lidar emulator that emits rays from the vessel's position relative to the Reference-map. This emulator evaluates which cells in the vessel's vicinity that should be updated, a process which is illustrated in Figure 5.2. The Explored-map array is continuously updated as the lidar emulator explores new areas. Also, the lidar emulator recognizes the distance to the nearest objects in the map and uses the logic presented in Section 4.3.3.2 to generate the SetPointDist parameter.

Two versions of the mapping simulator node have been designed. One version that does not update cells in sections where the lidar emulated rays does not hit any objects within the lidar range, and one that updates cells in these sections.

The first method is in line with how the Hector-SLAM algorithms update the map



(a) The lidar emulator sending out rays that hit objects in the Reference-map

(b) Updated Explored-map

Figure 5.2: *Lidar emulator*

in physical experiments, as explained in Section 4.2.2. It is this method that is used in this section. An example of the use of the second version, that updates the map in areas where the lidar rays do not hit any objects can be seen in Figure 4.8 and 4.11.

5.1.2 Vessel Simulator Node

The vessel simulator node simulates the dynamics of the vessel, and how it responds to actuated inputs on the motors. It does so by solving the equation of motion (3.5) for $\dot{\boldsymbol{\nu}}$ as follows:

$$\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}(-\mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}\boldsymbol{\nu} + \boldsymbol{\tau}) \tag{5.1}$$

Where the body-fixed input force $\boldsymbol{\tau}$ is found by first interpolating actuator input to local thrust force, using the relationship obtained in Section 3.3, and then by multiplying with the thrust allocation matrix found in Section 4.1.3 .

To obtain the position vector $\boldsymbol{\eta}$ in the Basin-relative reference frame, the resulting integrated velocity vector $\boldsymbol{\nu}$ is multiplied with the rotation matrix and integrated once again. The following equation shows this transformation:

$$\boldsymbol{\eta} = \boldsymbol{\eta}_0 + \int_{t_0}^{t} \mathbf{R}(\psi)\boldsymbol{\nu} dt \tag{5.2}$$

where $\boldsymbol{\eta}$ and $\boldsymbol{\eta}_0$ is the vessel position at time t and t_0 in the Basin-relative reference frame.

In effect, the vessel simulator node replaces the physical vessel and the Arduino node. Besides, the node generates the position of the vessel, which otherwise would have been obtained through the Hector-SLAM nodes.

5.2 Simulations Performed

In simulations performed in this section, the Reference-map that is used is an occupancy grid that has been generated through physical operations in the basin. The map used has a grid size of (256x256), and a resolution is 0.2m. The simulations are carried out in soft real time, meaning that time elapsed in simulations follows computer time.

In this section, the following three simulations are presented;

- **Simulation-1, Autonomous Exploration of Basin by the use of the Frontier-Based Exploration Strategy**
  In this simulation, the vessel autonomously explores the map using the Frontier-based exploration strategy. Snapshots from the simulation are shown in Figure 5.3. The exploration is straightforward and appears to be quite efficient. The vessel starts off by exploring frontiers in the left half of the basin until it is fully explored. It subsequently investigates the right half of the basin. After exploring the whole map it returns to its initial position.
- **Simulation-2, Autonomous Exploration of Basin by use of the Gap-Based Exploration Strategy**
  In this simulation, the vessel autonomously explores the map using the Gap-based exploration strategy. Snapshots from the simulation are shown in Figure 5.4. The behavior of the vessel during exploration is very similar to that of the Frontier-based exploration strategy; The vessel starts off by exploring the left side of the basin, before exploring the right side and finally returning to its initial position.
- **Simulation-3, Focussing Exploration Toward Unexplored Destination**
  In this simulation, the vessel starts off in its default autonomous exploration mode. This is shown in Figure 5.5a. Prior to Figure 5.5b, the operator has utilized the interactive map and clicked the location where the green marker has appeared. Since this location is unexplored, the system starts to focus exploration towards it, as explained in Section 4.4.1. In Figure 5.5d the vessel has reached the operator setpoint, where it awaits further instructions.

5.2.0.1 Simulation-1

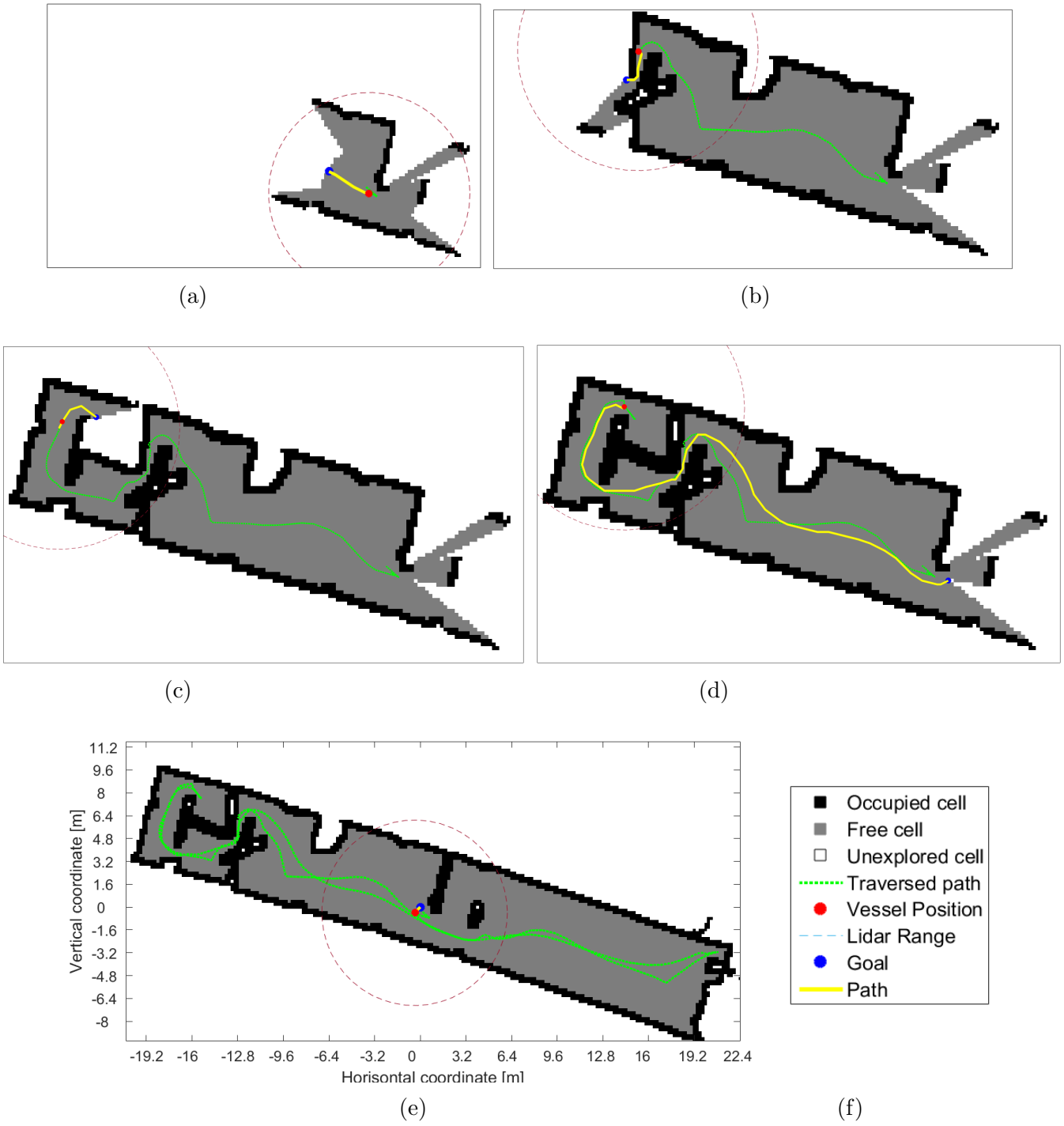


Figure 5.3: ***Simulation-1*** *is demonstrating the Frontier-based exploration strategy*

5.2.0.2 Simulation-2

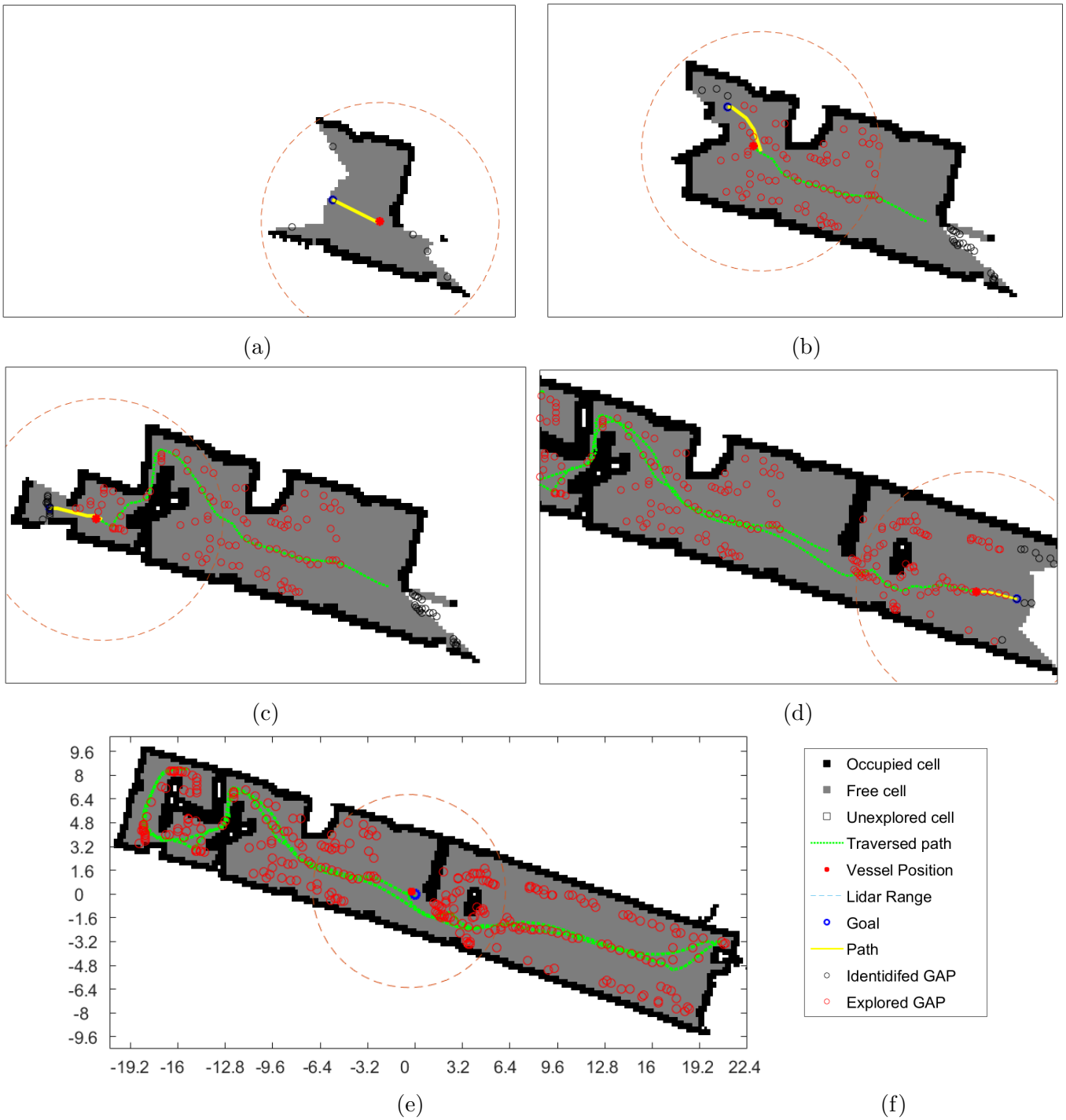
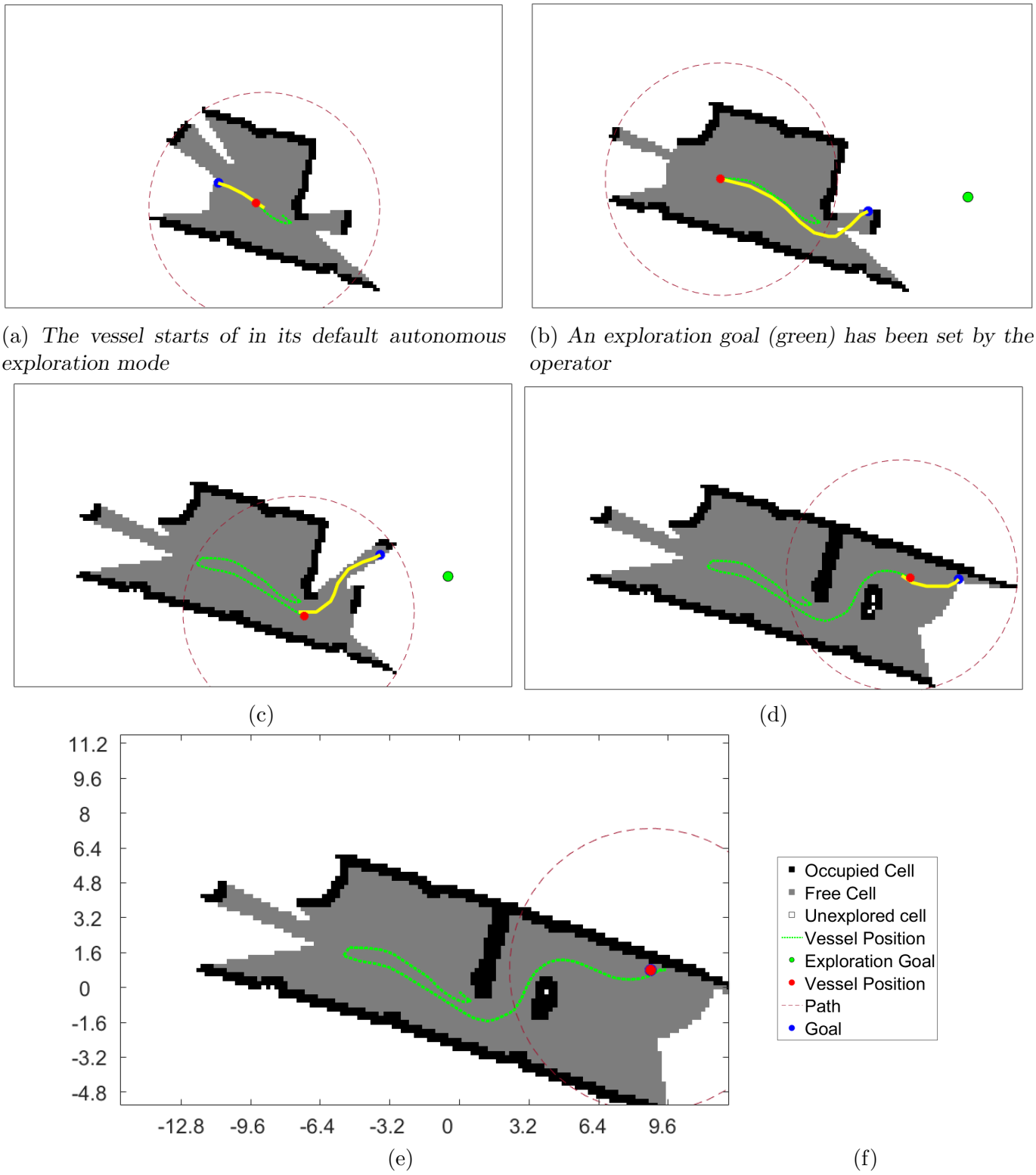


Figure 5.4: ***Simulation-2*** *is demonstrating the Gap-based exploration strategy*

5.2.0.3 Simulation-3



(a) *The vessel starts of in its default autonomous exploration mode*

(b) *An exploration goal (green) has been set by the operator*

(c)

(d)

(e)

(f)

Figure 5.5: ***Simulation-3*** *is demonstrating exploration towards a userdefined exploration goal*

.

5.3 Simulator Discussion

5.3.1 About Simulator

During development of the system, the simulator has been used a lot. This has involved using it to test algorithms and to identify bugs. Without the availability of the simulator, it is unlikely that a system with the same level of capabilities could have been designed in the same period of time, especially considering that the availability of the MC Lab was limited.

The simulations performed in this thesis can be classified as software-in-the-loop (SIL) simulations. Since the system is divided into loosely connected nodes, transforming it into a Hardware-in-the-loop simulator (HIL), which has stricter requirements does not require many steps. HIL testing would involve running the vessel simulator node on an independent platform that ensures real time, and should include the Raspberry Pi 2 and Arduino in the loop.

By changing the dynamics of the vessel simulator node, the simulator can be adapted to fit the dynamics of other vessels. It is therefore, the author's belief that the simulator can be useful for other applications than the exploration objective seen in this thesis. One example where it can be useful is for the simulation of two vessels approaching each other in a small corridor. In this case, one could investigate how the two vessels should interact and behave when passing each other. In this example, separate ROS-nodes should represent the GNC system of each vessel.

5.3.2 Simulator Versus Real System

Although most of the message parsing, several nodes, and the operator interface is identical in both cases, the simulator cannot be expected to fully replicate the physical system. In this regard, this section reviews some of the most important shortcomings of the simulator.

Due to only being approximations, both the dynamics of the simulated vessel and the thruster responses to actuator inputs are expected to deviate from the physical system, which is a result of imperfect modeling, noise, and biases. The effect is that the simulated vessel is more stable in reference tracking than the physical vessel.

Another noticeable difference is the quality of the lidar scan. Unlike in the physical system, the Mapping simulator node never experiences imperfections and sees new

objects immediately when they come within range of the lidar. The effect is that the vessel never needs to investigate objects closer in order to identify them. It can thus perform exploration more efficiently than the real system.

The ability of the vessel to follow calculated paths in the environment is verified in the next chapter. This means that strategies functioning in the simulator can be expected to also work in the basin, as long as they generate reasonable paths for the system to follow. This is true even if the vessel is considerably less stable in path-following in the physical experiments than what it is in the simulations. This observation means that the simulator efficiently can be used to test new exploration strategies.

5.3.3 Analysis of Simulation Performed

The simulations performed in thesis all proved successfully and were able to demonstrate that the two methods of exploration functioned as desired. In addition, the implementation of the method for focusing exploration towards a desired location proved successful.

Comparing the two methods, it is clear that they behave quite similar, and it is difficult to conclude which one is the better. Since the gaps are only spawned in the vicinity of frontiers, it can, in fact, be argued that the implemented Gap-based exploration scheme is a version of the Frontier-based exploration strategy. In order to adequately judge which method is the better, further testing should be performed in more complex environments. In addition they both methods should be tested through physical experiments.

Although both methods appeared to function well, it is the Frontier-based exploration strategy that has been chosen as the preferred one, and that is in focus during the experiments performed. This is due to its simplicity and it guaranteeing complete exploration.

Chapter 6

Experimental Results

6.1 Tools for Post Processing of Data

6.1.1 Recording of Data

The following means of recording data were utilized during experiments:

- **Filming**
  The operations performed in this chapter were filmed with either one or two cameras, while at the same time, the operator computer had enabled screen recording. In addition to providing a convenient method of presenting the results, the generated videos facilitate for analysis the vessel's behaviour. This analysis can, for instance, involve a comparison of the video from the operator screen and the video of the vessel.

- **RosBag Recording**

  During experiments, the ROS tool RosBag (ROS-community, e) is utilized for recording of data. By running the RosBag tool during an operation, all messages, that any topic receives is saved with a timestamp. In the implemented system, all information about the system that is deemed relevant for post-processing is routed through topics and is thus readily available for analysis after experiments.

  The generated RosBag-files can be played back in the ROS environment, however, it has been chosen to import the files to MATLAB for post processing. In MATLAB, the value of characteristic parameters such as vessel speed and thruster inputs are available as the vessel is traversing the map.

- **Online Saving Data of Exploration Node**

  The exploration node run in MATLAB saves the input variables that its main script receives in each iteration. After the exploration node has been stopped, the main script for exploration can be run with the stored input variables from previous iterations. By analysing how the scripts process this data, unexpected behaviour and bugs can be identified and fixed.

6.1.2 Animation Tool

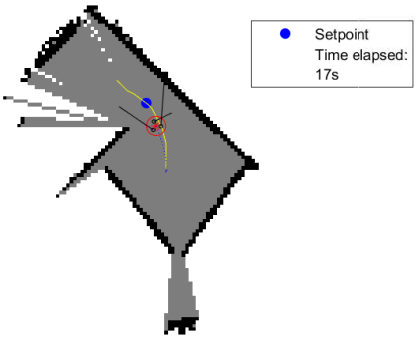
A script has been created in MATLAB (*RosbagReplay.m*), that reads RosBag data and automatically generates a video file that animates the exploration process. Specifically, this animation shows the setpoint of the vessel, planned path, path-goal, thruster input, vessel position and heading of the vessel as it traverses through the map.

In the production of the animation, a new frame is generated for every tenth message that has been processed. Since most topics are publishing approximately 20 messages per second, the animation is both more detailed, and is updated at a much higher frequency than the interactive window displayed online during operations.

Due to the amount of data, the generation of an animation file is quite time-extensive. As an example, the video file produced from Experiment-2 was generated by processing 250.158 messages and took few hours to produce.

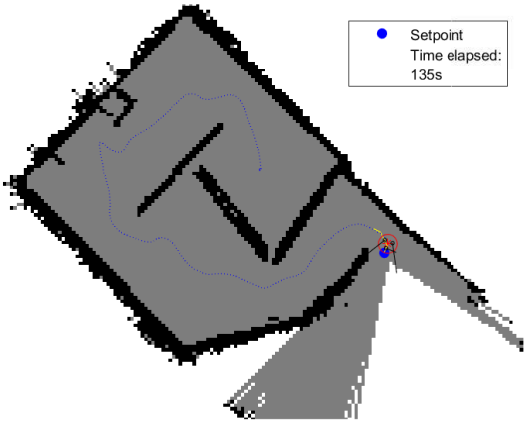
Figure 6.1 presents some snapshots of an animation produced by processing the data of Experiment-2. Note that the black lines which represent the thrust, are directly proportional to the actuated input signal and thus gives an overview over how the three thrusters interact with the system during operations.

It is concluded that the animations give useful insight to the exploration process and that it, in general, is helpful for analyzing the experiments. The visualization of the setpoint is especially helpful as it tells where the motion controller is aiming at any time.

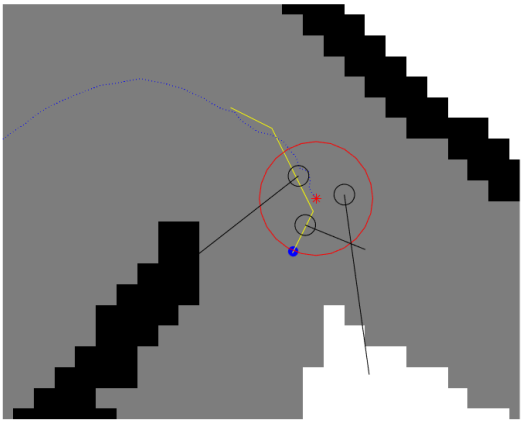


(a)

(b)



(c)



(d) *Zoomed in, focus on the vessel*

Figure 6.1: *Snapshots of the animation generated from Experiment-2*

6.2 Experiments

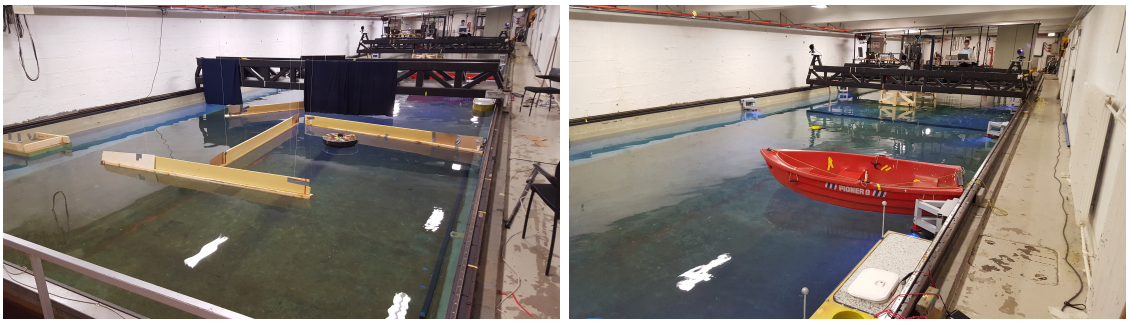
The following section will present relevant results from the experiments performed in the Marine Cybernetics Laboratory. Due to limited availability of the laboratory, only two full experiments, both using the Frontier-based exploration strategy were well documented. In both scenarios, static objects were introduced, as described in Section 2.1.2.2.

The experimental setup in the two experiments is quite similar, but not equal. In Experiment-1, the vessel starts in the middle of the basin and explores the full basin, while in Experiment-2 the vessel starts in the labyrinth-like section and explores towards the center. The obstacles in the basin are also set up slightly differently.

6.2.1 Experiment-1

Frontier-Based Exploration, Exploring the Whole Basin
In this experiment, the whole basin was successfully explored. The vessel mostly behaved as expected, but a few incremental adjustments were still performed in the algorithms prior to the next experiment. This involved adjustments the algorithms that decide where the setpoint should be placed.

The experiment was performed using an occupancy grid size of (256x256) and a resolution of 0.2 m. Images of the basin as set up in the experiment can be seen in Figure 6.2, while the fully explored basin from the trial can be seen in Figure 6.3.



(a) *Labyrinth section* (b) *Middle section*

Figure 6.2: *Laboratory setup in Experiment-1*

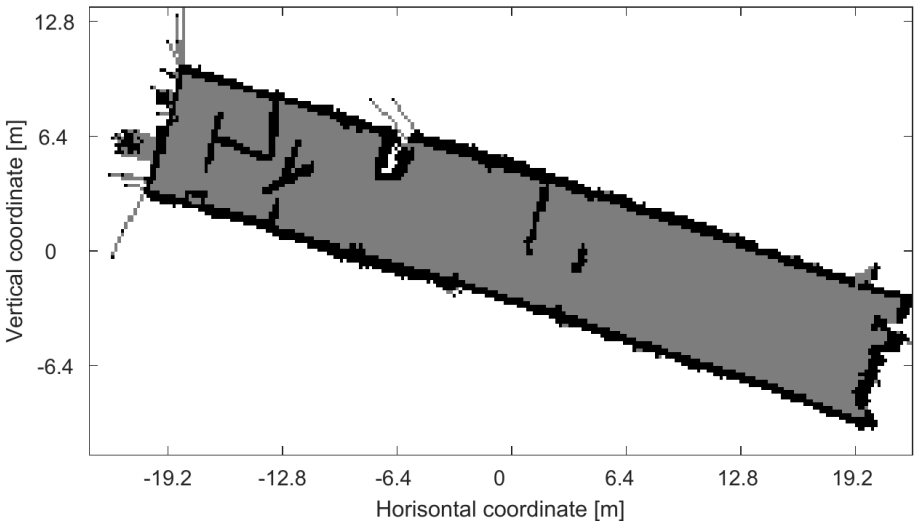


Figure 6.3: *Explored map of the basin, Experiment-1*

.

6.2.2 Experiment-2

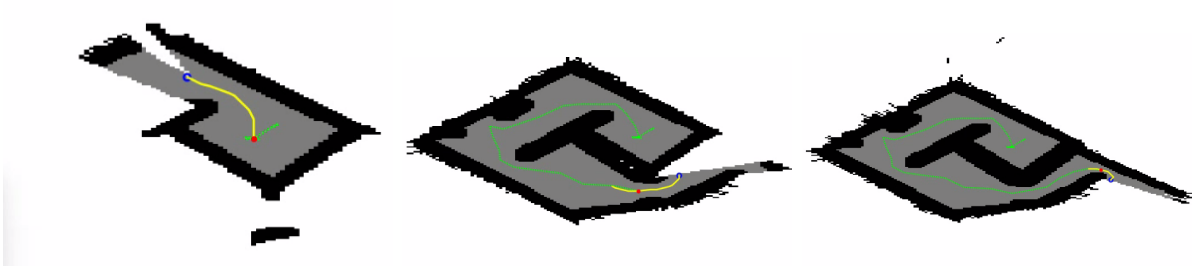
Frontier-Based Exploration, Exploring of a Section of the Basin
This is the final experiment that was performed with the vessel in this project, and the one that was best documented. The experiment yielded very satisfying results, with the vessel exploring the scenario without problems.

The experiment was performed using an occupancy grid size of (256x256) and a resolution of 0.1 m. Figure 6.4 illustrates how the vessel explored the scenario. The two first rows show the operator interface and the vessel in the laboratory respectively. The last row shows snapshots from the animation produced by post-processing the data.

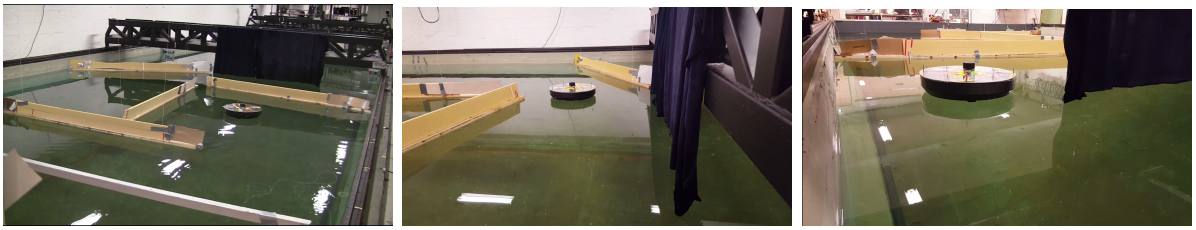
6.2.2.1 Speed Analysis

During Experiment-2, the vessel adapted its setpoint, and thus also its speed according to the distance to hinders. This helped to ensure that the vessel never came close to crashing, while at the same time efficiently exploring the basin.

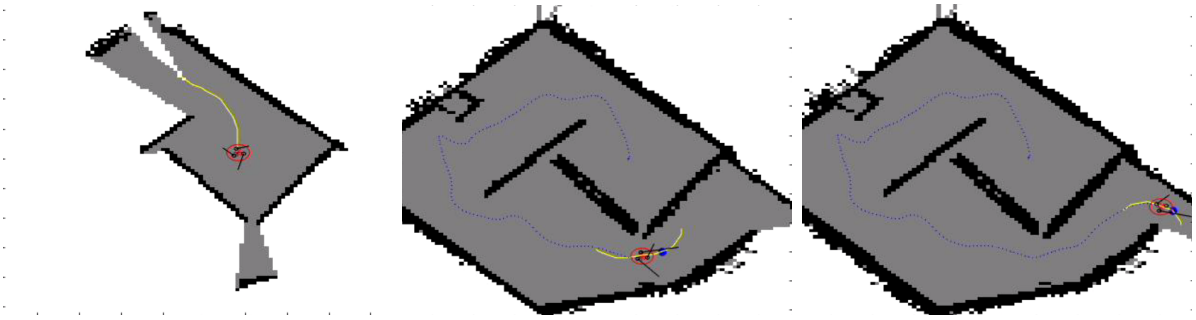
The speed of the vessel is plotted in Figure 6.5. Although the speed also varies as a natural result of the vessel changing directions, it is clear that the vessel is adjusting its speed according to the distance to nearby objects. In particular, this is evident when the vessel passes the last narrow passage. In this section, the



(a) *Computer interface, step A* (b) *Computer interface, step B* (c) *Computer interface, step C*



(d) *Image of operation, step A* (e) *Image of operation, step B* (f) *Image of operation, step C*



(g) *Post produced animation, step A* (h) *Post produced animation, step B* (i) *Post produced animation, step C*

Figure 6.4: *Snapshots from Experiment-2.*

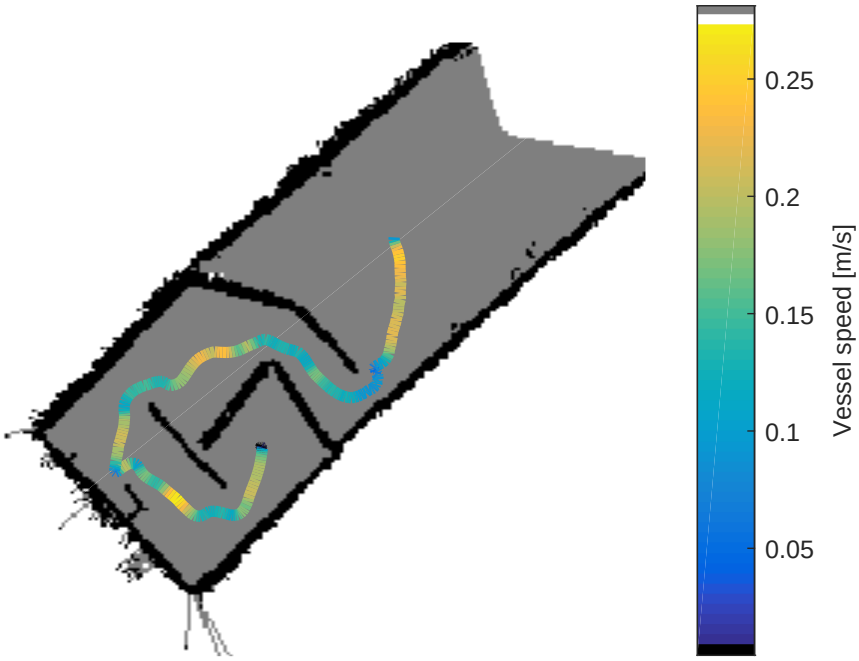


Figure 6.5: *Vessel speed as estimated by the observer, Experiment-2.*

vessel slows down to about 0.07 m/s. Once the vessel has passed it and gets into an open area it quickly speeds up to about 0.25 m/s.

6.2.2.2 Video

A video displaying the vessel operating in the basin during Experiment-2 is appended to the electronic attachment. This video is also available online (Ueland, 2016a)

The video is produced by merging the film from two cameras and the film generated by the enabled screen recording the operator computer. At the end of the video, the exploration process is reviewed through an animation, produced as described in Section 6.1.2.

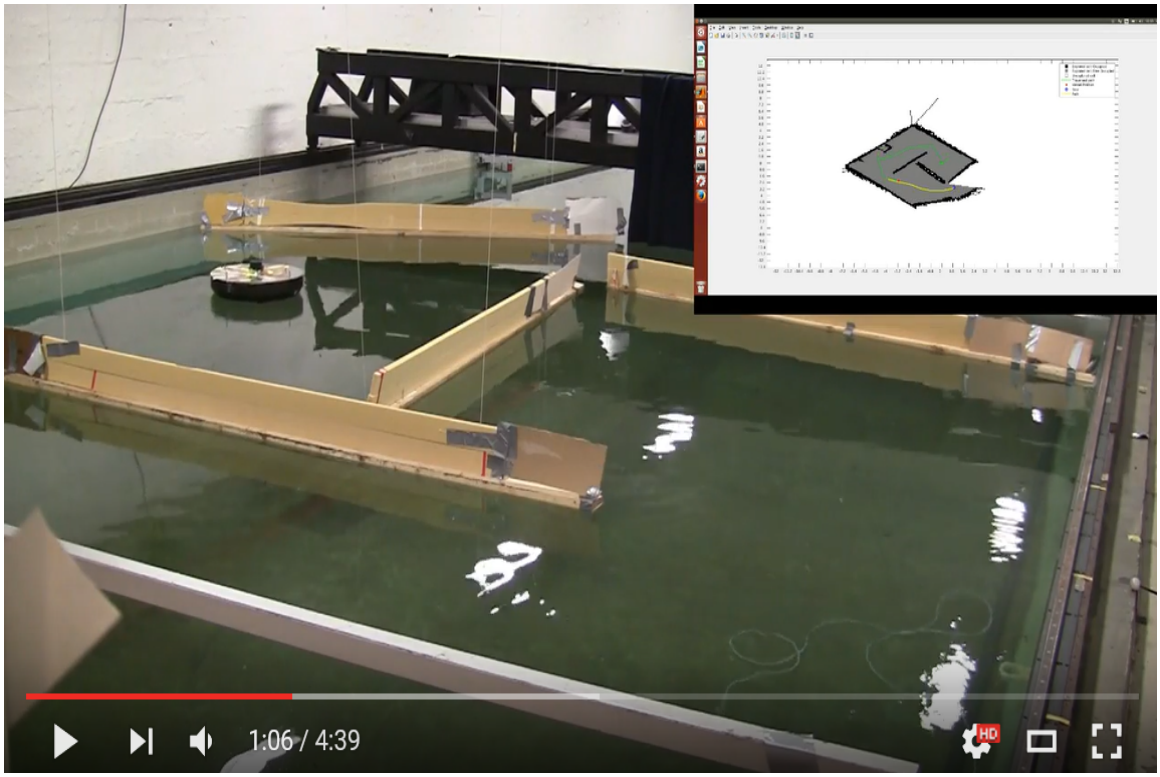


Figure 6.6: *Screenshot of video showing Experiment-2, as posted on Youtube.*

6.3 Discussion

In summary, the experiments proved successful, and the vessel was able to explore the basin autonomously. The speed regulation of the vessel also proved efficient. As is evident in the generated video, the operator was able to successfully utilize the explored map for path-planning to destinations within it.

Although the SLAM algorithms are able to recognize most objects successfully, it had some issues in recognizing the wall in the very left of the basin. This wall is a part of the basin's wave generator that has a smooth, non-vertical surface. The effect can be seen in Figure 6.4, where after halfway crossing the width of the basin, the vessel takes a detour and gets quite close to the wall, before it identifies it as an obstacle and moves on. Although it did not cause any problems in this experiment, it indicates that the vessel may have challenges in scanning certain types of objects.

Even though considerable effort has been made on improving it, the vessels stability and performance in reference tracking is still relatively poor. For this reason, the gains of the controller have been kept relatively low, resulting in slow operating speeds ranging from 0 to 0.3 m/s. It is the author's view that the vessels relatively poor performance in reference tracking currently are the biggest bottleneck in the overall exploring process. With better control of the vessel, the gains could have been set higher and the basin could have been explored faster.

A known challenge using SLAM algorithms is the localization of crafts in corridors where there are no features it can use as a reference. For this reason, it was expected that the algorithms could face problems in the sections of the basin where there are few hinders. As can be seen in right side of Figure 6.3, this was not the case, and the vessel explored well, also in these areas.

It is important to note that the successful exploration is partly a result of having a controlled environment in the basin. If there, for example, were waves present in the experiments, the lidar would be tilted up and down, in which case the SLAM algorithms is expected to face problems.

Chapter 7

Conclusions

The experimental platform, the CS Saucer, has been extensively upgraded during this thesis. This has involved changing the software platform to LabVIEW to ROS, which included replacing the NI myRIO unit with a Raspberry Pi 2, Wireless WiFi adapter and an Arduino. The implementation of ROS on the vessel makes the vessel much more versatile and is believed to be of advantage for future academic projects performed on the vessel.

The ability of the vessel to follow reference tracks has also been significantly improved. This is a result of a remapping of the thrust/force relationship, installation of RPM sensors, and a redesign of the control system including tuning of gains. Even with these improvements, the heading of the vessel still oscillates during operations. This is particularly the case when the commanded forces are large.

The use of ROS with a combination of nodes written in C++, nodes run in Simulink and C++ nodes generated from Simulink proved efficient. Indeed, the thesis has verified that the utilization of MATLAB and Simulink together with ROS functions well, and in particular it showed the advantage of being able to run a relatively advanced ROS node from MATLAB on the operator computer. This setup is believed to be of great advantage for future students in the MC Lab, where students, in general, are well educated in the use of MATLAB.

The A* search algorithm has been implemented for pathfinding. This algorithm is implemented in an efficient manner and is more advanced than its basic form in that it allows for longer connecting-distances between nodes and applies a weighted map. The paths generated are in general of a satisfying character and both and avoids sharp turns and keep a distance to walls if possible.

Two methods of exploration have been implemented to the system, namely the

Frontier-based exploration strategy and the Gap-based exploration strategy. Simulations showed that they were quite similar in performance, but due to the simplicity and it guaranteeing complete exploration, the Fronter-based exploration strategy was the preferred one, which was documented through experiments.

The exploration node, responsible for generating paths for the system to follow may use up to a few seconds to perform an iteration. Separate nodes that iterate with a much higher frequency are responsible for generating setpoints based on the calculated path. This scheme results in a steady stream of updated setpoints and is in general found to be successful.

The simulation model developed proved crucial for rapid development and troubleshooting of the system. In fact, given the limited time available in the laboratory, it is hard to conceive how the system could be created the with the same level of performance without it. The simulation model is believed to be useful also for other projects involving path planning for vessels.

The performed experiments were successful, and the vessel explored the basin efficiently, without significant problems. The vessel was able to regulate its speed according to the distance to nearby objects. In the performed experiment the vessel speed varied from around 0.05 m/s in the narrowest passages to above 0.25 m/s in the open area. The reason that the velocities are not even higher is that the gains of the motion controller had to be limited due to a relatively poor control of the vessel.

The utilized Hector-SLAM algorithms yielded precise localization and mapping of the vessel in its environment. The lidar was able to scan all objects present in the basin successfully, even if not all objects were recognized instantly. The range of the lidar, which was approximately equal the width of the basin appeared to be sufficient for the tested scenarios.

Summarized, the implemented system merge suitable exploration strategies, SLAM algorithms, a path planning strategy, motion controller, and a velocity control law to a well-functioning guidance, navigation, and control system. The system is able to perform the objectives that this thesis set out to solve. An additional benefit of the generated system is that it is constructed by the use of low-cost components.

Chapter 8

Further work

As of now the system has only been tested in a static environment. It is a future goal that the system should handle dynamic objects. The biggest challenge in this regard is believed to be assessing how the SLAM algorithms should handle dynamical objects. The SLAM algorithms are using their knowledge of the map for localization, and if considerable parts of the environment in the vicinity of the vessel changes, it will face problems of localizing the vessel.

Ideas for overcoming this challenge involve making sure that dynamical objects are recognized, making sure that only relatively small parts of the map changes, or to identify static features in the map that can be used as a reference for positioning.

The author would advise implementing the tracking system available in the laboratory to the system. The position data from this system could be implemented as odometry data to the SLAM algorithms, and thus, hopefully, making sure that the system can identify the vessel's position, even in dynamical environments. An accurate position estimate from the tracking system would also be useful for analysis of the SLAM algorithm's performance.

If the map generation, and in particular the localization of the vessel when facing dynamical objects are handled, deflection strategies for avoiding objects should be implemented. By using the lidar scan directly, dynamical objects within the vicinity of the vessel can be identified. Subsequently, a node that deflects the path around identified objects, before later merging it to the global path could be implemented.

It would be interesting if the vessel was fully autonomous without the need of the operator computer. If this is to be performed, the author's advice is to implement

the exploration node in C++ or Python from scratch. The script in MATLAB do not use any advanced built-in functions, and it can, therefore, with some effort, be implemented to C++ in a similar manner. If this were the case, a Simulink node could still be used as an interface with the operator, while the vessel itself could explore the basin fully autonomously independent on the operator computer.

Schemes, where algorithms are able to reuse information from previous iterations could probably increase the computational efficiency of the system. At the current stage, however, the operator computer is able to compute all path within reasonable times, and there does not appear to be a need for these kinds of improvements. However, if the exploration node is implemented on the Raspberry Pi 2, which has much less computational power, the implementation of these schemes should be considered.

The author recommends making an effort of further improving the vessels performance in reference-tracking. This could, for example, involve applying the RPM data for low-level control of the actuators. In this case, a scheme for reducing the noise of the RPM data need to be implemented. Another tactic that can be considered is to utilize the vessels ability to rotate its thrusters, which could facilitate for more advanced motion control schemes.

As a final note, the author advises setting up a system in the ROS-tool Rrt-Gui for simple control of parameters and gains in the system.

Bibliography

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Ferguson, M. Rplidar, source code. Retrieved 10th of March 2016, from: `http://wiki.ros.org/rplidar`.

Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.

Ford, L. R. (1956). Network flow theory.

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

Geraerts, R. and Overmars, M. H. (2004). A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V*, pages 43–57. Springer.

Gerkey, B. (2015). gmapping. Online accessed 10th of December 2015 from `http://wiki.ros.org/gmapping`.

Grabowski, R., Khosla, P., and Choset, H. (2003). Autonomous exploration via regions of interest. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1691–1696. IEEE.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

Idland, T. K. (2015). Design, construction, and control of marine cybership "c/s saucer". Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580.

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., and von Stryk, O. (2014a). Hector open source modules for autonomous mapping and navigation with rescue robots. In *RoboCup 2013: Robot World Cup XVII*, pages 624–631. Springer.

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., von Stryk, O., and Klingauf, U. (2014b). Robocuprescue 2014-robot league team hector darmstadt (germany). Technical report, tech. rep., Technische Universität Darmstadt.

Kohlbrecher, S., Von Stryk, O., Meyer, J., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 155–160. IEEE.

Kohlbrecher S, M. J. (2015). hector slam. Online accessed 10th of December 2015 from `http://wiki.ros.org/hector_slam`.

Leedekerken, J. C., Fallon, M. F., and Leonard, J. J. (2014). Mapping complex marine environments with autonomous surface craft. In *Experimental Robotics*, pages 525–539. Springer.

Leonard, J. J. and Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee.

Liquid-Robotics (2015). The wave glider. Retrieved 16th of May 2016, from: `http://www.liquid-robotics.com`.

Moravec, H. P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121. IEEE.

Mullane, J., Vo, B.-N., Adams, M. D., and Vo, B.-T. (2011). A random-finite-set approach to bayesian slam. *Robotics, IEEE Transactions on*, 27(2):268–282.

MUNIN (2016). Munin unmanned ship web page. Retrieved 10th of May 2016, from: `http://www.unmanned-ship.org/munin/`.

Nasir, R. and Elnagar, A. (2015). Gap navigation trees for discovering unknown environments. *Intelligent Control and Automation*, 6(4):229.

Naval-technology (n.d). fleet-class-common-unmanned-surface-vessel. Retrieved 10th of May 2016, from: http://www.naval-technology.com/projects/fleet-class-common-unmanned-surface-vessel-cusv/.

Neato-Robotics (n.d.). Neato-robotics-vacuum-cleaner. Retrieved 10th of May 2016, from: https://www.neatorobotics.com/.

Newman, P., Bosse, M., and Leonard, J. (2003). Autonomous feature-based exploration. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 1234–1240. IEEE.

NICommunityPostA (2016). Controlling a usb lidar using myrio and the 'classy' state machine. Retrieved 13th of March 2016, from, https://decibel.ni.com/content/docs/DOC-35698.

NTNU/ime/labs (n.d). Marine-cybernetics-lab, ntnu. Retrieved 10th of May 2016, from: https://www.ntnu.edu/amos/mclab.

O'Kane, J. M. (2014). A gentle introduction to ros.

Rescue-Robot-League. Rescue-robot-league. Online accessed 10th of March 2016 from https://www.robocupgermanopen.de/en/major/rescue.

Robotshop (2015). Rp-lidar. Retrieved 10th of December 2015, from http://www.robotshop.com/en/rplidar-360-laser-scanner.html.

ROS-community. Camera tools. Retrieved 10th of March 2016, from: http://wiki.ros.org/Sensors/Cameras.

ROS-community. Imu tools source code. Retrieved 10th of March 2016, from:: http://wiki.ros.org/imu_tools.

ROS-community. Introduction to ros. Retrieved 10th of March 2016, from: http://wiki.ros.org/ROS/Introduction.

ROS-community. Multiple-machines-ros. Retrieved 10th of March 2016, from: http://wiki.ros.org/ROS/Tutorials/MultipleMachines.

ROS-community. Ros-community, rosbag. Retrieved 10th of March 2016, from: http://wiki.ros.org/rosbag.

ROS-community. Ros-community, rosserial source code. Retrieved 10th of March 2016, from: http://wiki.ros.org/rosserial.

ROS-community. Ros-community, tf. Retrieved 10th of March 2016, from: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF.

ROS-community. Ros tutorials. Retrieved 10th of March 2016, from: http://wiki.ros.org/ROS/Tutorials.

Sharoni, R. (2016). Marine inverted pendulum. Unpublished master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S., and Younes, H. (2000). Coordination for multi-robot exploration and mapping. In *AAAI/IAAI*, pages 852–858.

Sørensen, A. J. (2013). *Marine Control Systems. Propulsion and Motion Control of Ships and Ocean Structures*. Department of Marine Technology, Norwegian University of Science and Technology, Trondheim, Norway, 3 edition.

Stachniss, C., Grisetti, G., and Burgard, W. (2005). Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and Systems*, volume 2, pages 65–72.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE.

Stentz, A. et al. (1995). The focussed d* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659.

Tovar, B., Guilamo, L., and LaValle, S. M. (2004). Gap navigation trees: Minimal representation for visibility-based tasks. In *Algorithmic Foundations of Robotics VI*, pages 425–440. Springer.

Ueland (2016a). Autonomous exploration by the use of lidar on a marine surface vessel. Online accessed 30th of June 2016 from: https://www.youtube.com/watch?v=BUihBGbhfDA.

Ueland (2016b). Frontier based exploraiton and the a* search algorithm (astar). Online accessed 30th of June 2016 from https://www.youtube.com/watch?v=AKwn3y9LC-g.

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

Vallabha, G. . (2010). Real time pacer for simulink. Retrieved December, 2015 from MATLAB Central File Exchange http://www.mathworks.com/matlabcentral/fileexchange/29107-real-time-pacer-for-simulink.

Woo, J., Seo, I., Lee, J., Park, J., Park, A., Kim, M., Jung, Y., Park, J., You, R., Choi, H., et al. (2014). Autonomous surface vehicle: Macs.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE.

Appendix A

Electronic Attachments

The files in this appendix are included in electronically submitted versions. The software components that are utilized during deployment or simulations is also publicly available on GitHub through the following repository:
`https://github.com/NTNU-MCS/CS_Saucer_ROS`

A.1 Parameter Generation Files

VesselParametersSet.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.

Mmp256.mat/map128.mat/map40
Files containing stored MATLAB workpaces. For simple setting the dimensions of the Simulink nodes.

Posedata.mat
Files containing stored MATLAB workpaces. For initializing of simulations.

A.2 ROS Nodes that are Launched During Deployment

A.2.1 Exploration_Pathplanner node

Exploration_pathplanner.slx
Simulink model Responsible for running the exploration node. Utilizes a series of subscripts. Subscribes to position and map, and publishes a path to the ROS architecture. To be run on operator computer.

A.2.1.1 Exploration_pathplanner scripts

- **ExplorationMain.m**
  Main script for exploration and pathplanning. Responsible for generating paths according to implemented guidance and exploration strategy.
- **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 there are no objects in direction where the lidar rays are not reflected.
- **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**
  Examines whether a gap can be marked as explored or not. Only used if the Gap-based exploration strategy is applied.

A.2.2 Path2SetPoint node

Path2SetPoint.slx
Simulink model used to generate the node in C++.
Generates appropriate setpoint on the planned path.

Path2SetPoint node folder
To be run as a regular ROS node.

A.2.3 Scan2SetPointDist

Scan2SetPointDist.slx
Simulink model used to generate the node in C++.
Finds the closest object based on lidar scan. Contains logic for finding setpoint distance.

Scan2SetPointDist node folder
To be run as a regular ROS node.

A.2.4 Hector2VesselPos

Hector2VesselPos.slx
Simulink model used to generate the node in C++.
Transforms position vector from lidar coordinate system to that of the vessel. Includes a quaternion transformation.

Hector2VesselPos node folder
To be run as a regular ROS node.

A.2.5 MotionController

Motion_Controller.slx
The control system of the vessel, responsible for controlling the vessel to the desired setpoint. This is performed by publishing appropriate signals to the actuators. It is adviced 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++.

MotionControllercompile node folder
To be run as a regular ROS node.

A.2.6 Arduino Code

CS_SaucerThrustRPMVoltage.inu
Arduino code responsible for publishing PWM signals to actuators and for monitoring revolution speeds and 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.

A.2.7 Hector-SLAM nodes

Open source nodes that are used for SLAM.
See `http://wiki.ros.org/hector_SLAM`

A.2.8 RPLidar node

Open source nodes that are used as driver for the RP-lidar.
See `http://wiki.ros.org/RPLidar`.

A.2.9 ROS_Serial node

Open source package for the Arduino. See `http://wiki.ros.org/rosserial`

A.3 Simulator Nodes

A.3.1 Vessel Simulator node

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

Vessel Simulator node folder
To be run as a regular ROS node.

A.3.2 Mapping Simulator node

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

MappingSimulatorUpdateAll.slx
Version of the mapping simulator where map is updated also in sections where lidar rays are not reflected

Mapping Simulator node folder
To be run as a regular ROS node.

A.4 Launch Files

Res01.launch
Launches RPLidar and Hector SLAM nodes for mapping with a resolution of 0.1m, and a gridsize (256x256)

Res02.launch
Launches RPLidar and Hector SLAM nodes for mapping with a resolution of 0.2m and a gridsize of (256x256)

Simulation.launch
Launches simulator nodes

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

A.5 Other

A.5.1 Bag Postprocessing script

bag_replay.m
Script that read data from the Bag_files. Generates an animation video

A.5.2 Astar animation generation file

Animate_AStar.m
Script that generates a path in the map, , while at the same time generating a animation file that show how it investigates the map. Generates video of the type as seen in the start of the video referenced in 4.3.2.6

A.5.3 Real time pacer

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

A.6 Raw Data for Mapping Actuator Input To Force Vector

FORCETHRUSTERMAPPING folder.
Contains the raw data and scripts used to perform the mapping from actuator input to force. (See section 3.3)

Appendix B

Software Set-Up and Installation

This manual is intended for use at NTNU, and especially for students that want to use ROS as their software framework for projects in the Marine Cybernetics Laboratory at NTNU. Though specifically intended for students working in the Marine Cybernetics Laboratory, it is hoped that it might also be useful for other readers.

The goal of the manual is to describe the steps needed to set up the ROS framework 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 can use 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 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 of ROS may be needed for students setting up their own system.

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 does not use the same amount of time on the same issues.

Note that due to the high level of development of ROS and the robotics community, the manual will probably need to be adapted to future versions. For example, Raspberry Pi 3 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.

B.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 the computer (Micro-SD/SD adapter or Micro-SD/USB adapter
- Hardware for interfacing the RP2 (monitor, ethernet-cable or WiFi adapter, HDMI-cable, mouse and keyboard).

B.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]

B.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 is 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]
- 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 (excluding the overclocking part): `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/Installation/UbuntuARM]

B.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:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src$
$ catkin_init_workspace
```

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

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

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

```
$ source gedit ~/catkin_ws/build/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

B.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 the bridged WiFi settings.

Now connect both Raspberry Pi 2 and Laptop to the MC Lab network and note their IP addresses. You can find the IPs by the command:

```
$ ifconfig
```

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

Unit	Username	IP
RP2	ubuntu	192.168.132
Laptop	einar	192.168.232

Now edit the hosts file

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

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

```
192.168.0.132 ubuntu
192.168.0.232 einar
```

Now edit the bashrc file

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

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

```
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 you no longer wich to have RP2 as the ROS master.

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

In the particular setup used in this thesis, the RP2 did not receive enough power when all components were connected. For this reason, the current limit was changed on the RP2. This was performed by adding the following line to /boot/-config.txt

```
1 max_usb_current=1
```

B.3.1 Arduino on ROS

In order to get the Arduino to ROS one should install Arduino IDE and the Rosserial package to the 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.

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 into the Arduino IDE from the operator computer as follows:

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

For the motor controllers, the built in ROS-Servo example is good. See tutorial on this source:
`http://wiki.ros.org/rosserial_arduino/Tutorials/Servo`

B.4 RP lidar and Hector-SLAM in ROS

Install the RPLidar drivar through the following commands:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/robopeak/rplidar_ros
$ cd ~/catkin_ws
$ 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.

```
$ 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:

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

Or check out either one of these sources:
`http://blog.zhaw.ch/icclab/rplidar/`
`http://wiki.ros.org/rplidar]`

Get the Hector-Slam and TF package

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

Now locate the Hector_mapping launch file

```
$ roscd hector\_mapping
$ cd launch
$ gedit mapping_default
```

Add the following line (which is the same as assuming that the lidar is placed at the vessels center of origin)

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

Also adjust parameters names of the launch file as explained in `http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot`

Now test if Hector_SLAM is able to process the Lidar data by running the rp_lidar and Hector_SLAM nodes.

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

Appendix C

Launch Manual

C.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. Also, see Figure C.1 for a coupling diagram of the Arduino.
2. Connect the battery to the system and the lidar to the Raspberry Pi 2. Place the lid as instructed in Section 3.1.3.
3. Connect to the MC Lab network on the operator computer.
4. SSH into the RP2 and launch the RP2 nodes (the mapping nodes are launched later).

```
$ ssh ubuntu@ubuntu
$ cd catkin_ws/src
$ roslaunch LaunchNoLidar.launch
```

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

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

```
$ ssh ubuntu@ubuntu
$ cd catkin_ws/src
$ roslaunch res01.launch
```

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

```
rosinit('ubuntu') (workspace commando)
load Frontier256.mat (workspace commando)
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 behaviours or errors than the C++ compiled version of the motion controller yields.

```
rosinit('ubuntu')
run VesselParametersSet.m
run motioncontroll.slx
```

   Option b SSH into the RP2 and run the motion_controller node

```
$ ssh ubuntu@ubuntu
$ cd catkin_ws/src
$ 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 MC Lab

8. Turn the motors on and monitor exploration

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`

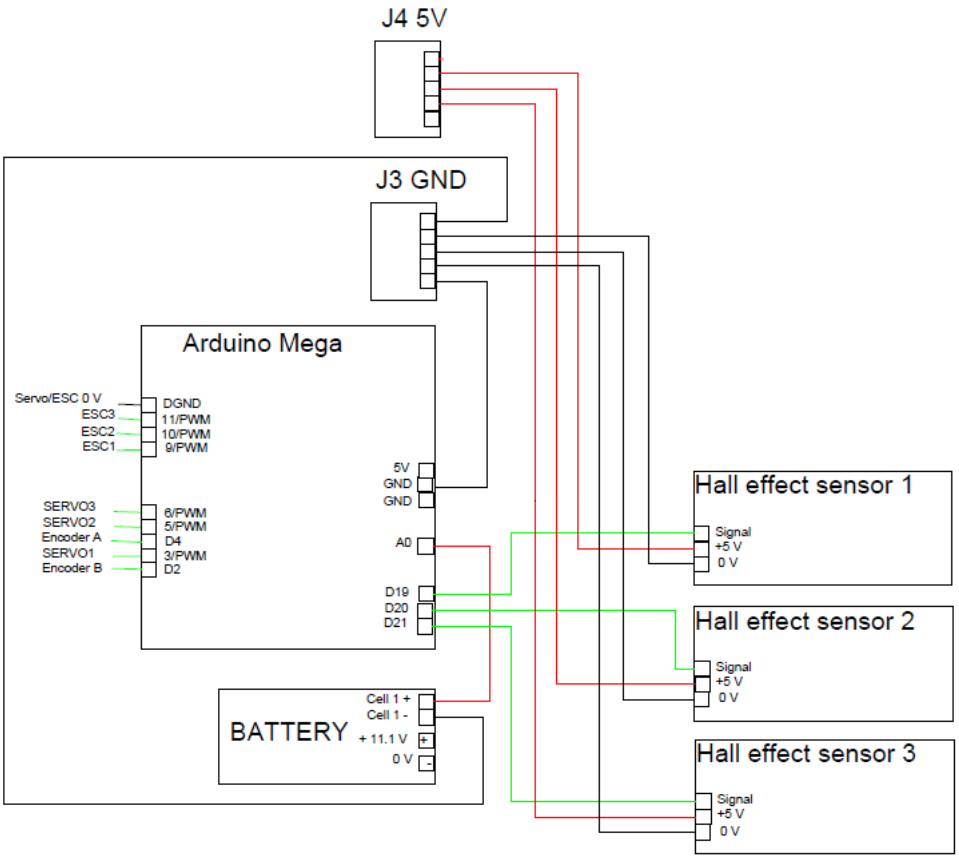


Figure C.1: *Wiring diagram of the Arduino Mega. Courtesy of Sharoni (2016).*

C.2 Perform simulations

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 gridsized, 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 simulator.launch
4 rosinit (MATLAB workspace)
5  Setstuff256 (running script from MATLAB workspace)
6  run Exploration_pathplanner.slx
```

Appendix D

ROS Architecture Overview

Table D.1: *Nodes in the system explained*

Node name	Node function	Topics Subscribed to	Topics Published to	Described in
base_to_laser_broadcaster	Coordinate transformation between base-link of the robot and lidar position.	-	tf	ROS-community (g)
Exploration_pathplanner	Responsible for generating path according to implemented guidance and exploration strategy	map Position	Motor_On_Off Path GAPS UNFGAPS	Section 4.4, 4.3.2 and 4.3.1.
Hector2VesselPos	Transformation of position vector from Hector-SLAM to vessel coordinates. Includes a quaternion transformation	Poseupdate	Position	Section 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	Kohlbrecher S (2015)
motioncontroller	Responsible for controlling the vessel to the desired setpoint. This is performed by publishing appropriate signals to the actuators.	Setpoint Position Motor_on_off	VesselSpeed nu Thrust1,Thrust2 Thrust3 a1,a2,a3	Section 4.1
Path2SetPoint	Generates an appropriate setpoint on the planned path	SetPointDist Path	SetPoint	Section 4.3.3.1
rplidarNode	Driver for the lidar. Generates range data in the 2D plane as registered by the lidar scann	none	scan	Ferguson(n.d.)
Scan2SetPointDist	Find the closest object based on lidar scan. Contains logic for finding setpoint distance	Scan	SetPointDist ScanDist	Section 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 BatterEncoderThrustRPM	ROS-community (f)

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

Topic	Topic Explained
a1 a2 a3	Actuator input for angle of thrusters, interpreted and transmitted by the Arduino
BatterEncoderThrustRPM	Measured revolution signal from each motor and measured voltage of battery. Interpreted by the Arduino.
GAPS	List containing gaps that are candidates for investigation.
map	Occupancy grid represented as a vector. Also contains properties such as the resolution. The occupancy grid represents the environment that the lidar has explored.
Motor_on_off	Toggle motor status between on and neutral. Only used on the C++ compiled version of the motion controller node
nu	Body-fixed speed in surge, sway and yaw, as estimated by the observer
Path	Vector of length 256 that represents discrete points in the planned path. Index 1:128 is represent x positions, index 129:256 represent y positions.
PoseUpdate	Position/Attitude of lidar as estimated by the Hector SLAM package.
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.
SetPoint	Setpoint of vessel in surge, sway and yaw
SetPointDist	Distance for setpoint ahead in the path from the point on the path that is closest to the vessel.
Thrust1, Thrust2, Thrust3	Actuator input for thrusters. Interpreted and transmitted by the Arduino
UNFGAPS	List containing gaps that have been examined.
VesselSpeed	Speed of the vessel in the direction it is travelling, as estimated by the observer

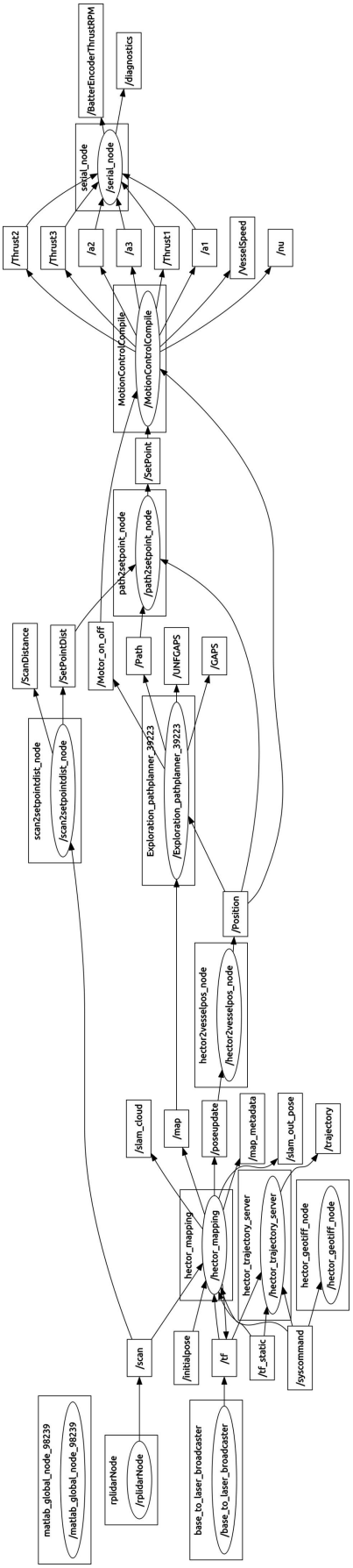


Figure D.1: *Overview over topics and nodes in the system as generated by running the native rqt_graph tool in ROS during operation.*